Provenance for Interactive Visualizations

Fotis Psallidas
Computer Science, Columbia University
fotis@cs.columbia.edu

Eugene Wu
Computer Science, Columbia University
ewu@cs.columbia.edu

ABSTRACT
We highlight the connections between data provenance and interactive visualizations. To do so, we first incrementally add interactions to a visualization and show how these interactions are readily expressible in terms of provenance. We then describe how an interactive visualization system that natively supports provenance can be easily extended with novel interactions.

ACM Reference Format:

1 INTRODUCTION
Interactive data visualizations enable users to rapidly recognize important patterns within the data, by leveraging the powerful capabilities of the human perceptual system, and to identify and explore salient relationships that are not readily evident from a static visualization. As such, they constitute a cornerstone in many human-in-the-loop data analysis and management systems across domains including data exploration and decision-support [39, 41], knowledge exploration [3, 37], debugging and analysis of machine learning and statistical models [49, 54, 55], interactive data cleaning [27, 28, 60, 61] and profiling [15, 40], to name a few.

The increasing importance and ubiquity of interactive visualization tools, along with the massively increasing scale of modern datasets, has seen a convergence between the visualization and database communities. Visualization systems incorporate data processing capabilities such as filtering, grouping, aggregation, ordering, and scaling in order to compute data summaries that are further rendered on the screen. However, increasing dataset sizes has caused data processing to become a core bottleneck that impedes interaction responsiveness and is detrimental to the overall data analysis and exploration of end-users [19, 22, 53].

To this end, recent work in both communities has proposed systems to combine query processing and visualization functionality within a single framework. For instance, Reactive Vega [51] draws upon stream query processing and declarative languages (i.e., Vega-lite [50]) to model the data processing, visualization, and interaction processes within a unified dataflow framework. Similarly, the Data Visualization Management System [16, 63] proposes a relational abstraction to model interactive visualizations as relational workflows that map database relations and relations of user events to marks and, ultimately, pixels on the screen. For instance, consider the multi-view interactive visualization of Figure 1: Figure 1 visualizes a breakdown of delayed flights [38] coupled with a crossfilter interaction technique [12]. Each chart renders the output of a count aggregation query of delayed flights grouped by different attributes: by state (A), airline (B), departure delay (C), date (D), month (E), and year (F). Thus, the visualization may be modeled as a large relational workflow composed of these aggregations, along with visualization workflows to map the results to visual marks (e.g., rectangles, circles, or polygons) which, in turn, are mapped to pixels on the screen. Crossfilter interactions let users select data in any of the views and see the other views update to show the statistics represented by the selected subsets.

Drawing the connection between relational workflow processing and interactive visualization not only improves the productivity of developers by introducing higher level languages to express visualizations, but has led to a rich area of performance-oriented research. Recent research efforts adapt query optimization techniques to the visualization domain and develop novel techniques inspired by unique characteristics of visualizations. These include adapting columnar execution [28], perception- and visualization-aware online aggregation [2, 29, 42, 47], speculative exploration sampling [26], and visualization prefetching [6], to name a few. Notably, most of this work has been focused on speeding up specific visualization interactions or specific classes of database queries.

In this paper, we build on this convergence by highlighting the connection between data provenance [9] and visualization interactions. Provenance broadly describes the process by which data artifacts are created and transformed. In the context of a relational workflow, it both describes the sequence of operators that transformed input relations to result relations (coarse-grained provenance), as well as relationships between individual input and output records of the workflow (fine-grained provenance or lineage).

The use of provenance in visual analytics is not new. Previous efforts leveraged coarse-grained provenance of data, interactions, and
visualizations in the form of histories that can be used to support collaborative communication, replication and reproducibility, action recovery, sense-making, and meta-analysis (see survey [46] and tutorial [24]). Unfortunately, the role of fine-grained provenance in interactive visualization has been less explored. We believe that a major factor is performance: the overhead to track fine-grained provenance can slow fast query processing engines by multiple orders of magnitude and cripple interaction response times.

To this end, recent work demonstrated a fine-grained provenance-enabled relational engine [44, 45] that is fast enough, and incurs sufficiently low overhead, to out-perform specialized interactive visualization systems on cross-filtering benchmarks and maintain sub-100ms interaction times on a 123.5M row flight dataset. These results illustrate the feasibility of expressing interactive visualizations using high-level provenance constructs, while also benefiting from fast execution engines. Following this, the purpose of this paper is to explore two questions: how can leveraging provenance concepts make it easier to build existing interactive visualizations?, and does taking a provenance perspective enable new interactions and visualization interfaces that are otherwise challenging to express?

The rest of the paper is split into two sections. Section 2 introduces the connections between interactive visualizations (and when possible, interactive applications in general) and provenance concepts. To do so, we start with a trivial non-interactive visualization, and incrementally endow it with different types of interactions commonly found in the information visualization literature. For each, we will describe how it is currently constructed, draw its connection with provenance, and remark on details regarding performance or semantics. Section 3 builds upon this perspective by exploring how expressing and implementing interactive visualizations on top of provenance-enabled visualization engines can leverage existing provenance analysis research and greatly extend the expressive power of interactive visualizations.

2 INTERACTION AS PROVENANCE

Interactive visualizations can be modeled as workflows that map between the data and the pixel space. User interactions can be viewed as dynamically transforming the workflows, or rapidly creating new workflows, and ultimately cause changes in the pixel space [63]. In this section, and along this conceptual model, we illustrate the connections between provenance—in particular, fine-grained provenance between individual input and output records—and visualization interactions. (We use provenance and fine-grained provenance interchangeably, and clearly state when we refer to coarse-grained or other forms of provenance semantics.) To better explain the connections, we progressively build interactive visualizations over the following database schema of delayed flights:

```
<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>ontime</td>
<td>fid, y, m, d, h, adelay, ddelay, src_apid, dst_apid, alid</td>
</tr>
<tr>
<td>airlines</td>
<td>alid, name, active</td>
</tr>
<tr>
<td>airports</td>
<td>apid, name, lat, lon, elevation, city, state, country</td>
</tr>
<tr>
<td>shapes</td>
<td>state, polygons[]</td>
</tr>
</tbody>
</table>
```

**Listing 1: Example Database Schema**

The ontime table records the arrival and departure delays of each flight (i.e., adelay and ddelay, respectively) from a source airport with id src_apid to a destination airport with id dst_apid along with the departure time of the flight (i.e., year, month, day, and hour) and the carrier that operated the flight alid. The airports table records the id of each airport (apid) along with its name, latitude (lat), longitude (lon), elevation, city, state, and country. The airlines table stores the id of an airline (alid) along with its name and whether or not the airline is still active. Airports and airlines serve as dimensions tables to the fact table ontime. Finally, the shapes table records an array of polygons that corresponds to the geographical bounds of each state in the US.

**Initial Static Visualization.** Let us start by building a static visualization to depict the number of flights for active airlines per state as a heatmap, similar to the one in Figure 1(A). In purely relational terms, we can specify this visualization as follows:

```
-- Data Processing
Q1 = SELECT COUNT(*) AS cnt, AVG(ddelay) AS avg_ddelay, AVG(adelay) AS avg_adelay, state
FROM ontime, airports, airlines
WHERE ontime.alid = airlines.alid AND ontime.src_apid = airports.apid AND airlines.active = 'Y'
GROUP BY state
-- Visualization
S = SELECT MIN(cnt) AS mi, MAX(cnt) AS mx FROM Q1
P = render_map(S)
```

**Listing 2: Example of a static visualization.**

Figure 2a depicts the workflow described by the above queries. Q1 specifies the data processing part of the visualization and consists of a join between the ontime, airlines, and airports relations followed by a filter on active airlines and a group-by state count aggregation. (Q1 also computes the average departure and arrival delays per state that we use later in interactions.) M constitutes part of the visualization workflow that transforms the output of Q1 into attributes of polygon marks (i.e., geometry and color of each polygon). color() is syntactic sugar for an equation that maps each count value to an output range of green hues, where the input range is computed by S as the minimum and maximum counts from Q1. Finally, the polygons are rendered on the screen using a mark-specific render_map() shim. We omit further details for space considerations and refer interested readers to prior work in relational specifications of visualization workflows [16, 63].

Under this model, the visualization application is a (possibly complex) relational view V that maps the input database in data space to rendered marks in pixel space. Next, we elaborate on the connections of common interactive capabilities and provenance concepts by building on this static visualization example.

**Interactive Selections.** One of the fundamental building blocks of visualization management systems is the ability to interactively reference visual marks by clicking, lassoing, or other types of selection operations [50, 56, 58]. Although users interact with visual marks, the intention is typically to manipulate the underlying data represented by the visual marks rather than the marks themselves. To this end, visualization research has developed many techniques to invert selections in pixel space to declarative selection queries in the input data space [13, 21, 33, 37, 50].
The predominant forms of selection are item/group selection and range selection. Consider the map in Figure 2b. Item and group selection may correspond to clicking on one or more states, where the selection is a set of states. The intention is to identify the input records associated with the selected states. Range selection may correspond to drawing a bounding box (dashed red box). This may be interpreted as group selection, where the set of states corresponds to the state polygons that intersect with the box. However, the intention may also be to translate the bounding box into a predicate over `lat`, `lon` attributes in the shapes polygons. The latter representation can be attractive because the selection can be further manipulated and relaxed to, say, add additional predicates (e.g., `adelay > 5min`), modify the predicate clauses (e.g., increase the `lon` range), or remove unnecessary clauses [21].

Connection with Provenance: All of the above selection types are variants of a common provenance operation known as backward trace, which identifies input records that contribute to specified output records. Different backward trace implementation techniques correspond to the above selection semantics.

Visualization systems typically support range selection when the visualization workflow consists of rescaling data attributes to visual variables (e.g., `COUNT` to y pixel position). Since the scaling operations are typically invertible, it is simple to say, rescale a bounding box’s coordinates from `y_min` to `y_max` to be in terms of `COUNT`. Provenance research generalizes this by computing the workflow’s inverse function `V_1^(-1)()`. This can be done through weak inverse functions [59], deriving provenance predicates from relational workflows [25], or by explicitly annotating each operator with an inverse function [62]. Expressing range selections as backward trace helps extend its support to visualizations that perform complex data processing, as well as rendering.

Item (or group) selections identify the specific input records that correspond to the user’s selection in pixel space. Visualization systems typically implement this by annotating records as they flow through the visualization workflow so that the output is annotated with the input records [8]. However, annotations [7, 36] are only one mechanism to answer fine-grained provenance queries. They can also be computed by evaluating the provenance predicates above, or by explicitly materializing input-to-output record dependency information as explicit index data structures when executing the visualization workflow [45, 62].

A note on semantics: One subtle point is that provenance systems may support different types of provenance semantics, and visualization developers should be aware of these semantics. For instance, assume we select outputs of Q1 and want the corresponding airlines from the `airlines` relation. We typically only want the set of airlines, rather than the bag of every copy of the airlines that were used to derive the selection. In this case, visualization toolkits should demand “which-provenance” semantics [17] as opposed to general transformation provenance semantics that return each airline tuple as many times as it contributes to selected outputs. (See [9, 17, 25] for an introduction to different provenance semantics.)

Tooltips and Details-on-Demand. A common use case once a user has performed a selection is to show detailed information, or summarizations, about the selected data. Tooltips and details-on-demand are popular examples of this paradigm.

Tooltips render information (say, in a modal pop-up) that contains information about the provenance of the selected marks. For instance, when users select states in Figure 2a, they may want to see additional attributes per state such as the average arrival and departure delays (i.e., `avg_adelay` and `avg_ddelay`, respectively).

Details-on-demand go beyond tooltips by retrieving and further processing user selections. For instance, when hovering over a state, the visualization may update to show a detailed list of airports operating in the state. Another form of details-on-demand is to semantically zoom into the user’s range selection. For instance, the user may select states with a range selection on the map. In response, the visualization updates to zoom into the range and show, say, detailed city-level breakdowns of counts of delayed flights.

Connection with Provenance: These functionalities are often implemented as standalone features in a visualization system. However, they can be easily expressed as queries that take the backward trace of the user’s selection as input. We illustrate this in Figure 2c. The user selection in the visualization is traced back to the input records, then a second visualization workflow `V_2` (often expressed as a SQL query) computes statistics about the provenance and renders them as details. The primary distinction between the above examples is the definition of `V_2`, which we illustrate in Listing 3 below:

```
-- Tooltips
T = SELECT avg_ddelay, avg_adelay
   FROM backward_trace(selected, Q1)

-- Details-on-demand
D = SELECT * FROM backward_trace(selected, airports);
Z = SELECT COUNT(*), city
   FROM backward_trace(selected, ontime) A1,
           backward_trace(selected, airports) A2
   WHERE A1.alid = A2.alid
   GROUP BY city;
```

Listing 3: Examples of tooltips and details-on-demand
The tooltip query $T$ traces the provenance of the user’s selected states to the output of Q1, and returns the average departure and arrival delays. The details-on-demand shows two queries. $D$ retrieves the list of airports within the selected states. $Z$ performs the drill-down from state to city-level statistics, for the selected states. It does this by joining on time records and airports for the selected states, and re-computes the number of delays for each city.

**A note on performance**: Joins, such as the one in the query $Z$ above, are common in visualizations. To avoid potentially expensive join execution costs, it is common practice for visualization systems and developers to first denormalize relations ahead of time. The visualization is then implemented over the denormalized relation. However, denormalization is only one possible join optimization and comes with several costs. It introduces redundancy, is time- and space-consuming to construct, and in many cases not even required. Furthermore, this focus on denormalization is an example of violating physical data independence [11] and impedes rapid visualization development. For instance, developers may spend considerable time writing application code to essentially denormalize on time=airport$s and compute the per-city count. Later, they may want to iterate on the visualization design and try showing, say, other statistics or grouping by elevation. However, they may be reluctant to incur the same engineering cost to try another design. This is because each design change implies the time- and space-consuming process of reconstructing the denormalized relation.

In contrast, expressing this logic in terms of provenance and relational operations enables rapid design iteration by offloading implementation to the visualization engine. Furthermore, recent work [45] suggests that workflows composed of provenance and relational operations can be optimized to ensure interactive response times by, say, materializing efficient join indexes adaptively, partially denormalizing the database, and pre-computing statistics.

![Figure 3: Linking and Cross-filtering](image)

**Multi-View Linking.** Linking is a common class of interactions where selections in one view update other views. Prominent examples include linked brushing and cross-filtering.

**Linked brushing:** Suppose we render a scatterplot of the average arrival (y-axis) and departure (x-axis) delays for each state, as computed by Q1 in Listing 2. Consider the visualization in Figure 3a. Linked brushing may let users select states on the map (red box) to highlight the corresponding delay information for each selected state in the scatter plot (red circles), and vice versa.

**Cross-filtering:** Cross-filtering is used to explore correlated statistics across multiple visualization views [12]. In the common setup, each view is the result of an aggregation query over different combinations of input attributes (e.g., each view in Figure 1). Selecting marks in one view recomputes the aggregation queries over the subset of input records represented by the selection, and updates the views accordingly. Figure 3b illustrates a simple example where selecting a set of states updates the counts of flights per carrier.

**Connection to Provenance:** Linked brushing is precisely backward tracing from the states to the input state records, followed by forward tracing to highlight the states in the scatterplot. Cross-filtering is expressed as backward tracing followed by refreshing the other views by executing the queries (e.g., $V_1$ in Figure 3b) over the provenance. The difference is based on the forward tracing operation. In this example, linked brushing traces the subset to the output marks, whereas cross-filtering recomputes the views for the output marks.

**A note on semantics:** To better highlight the importance of the provenance literature in the domain of interactive applications, note that the update procedure corresponds to a common provenance operation, known as selective refresh in the provenance literature. Selective refresh may not always update the same target outputs, for instance if the workflow contains a one-to-many operator followed by two non-monotonic aggregation operators [25]. The notion of unsafe selective refresh, and recent techniques to address it [10], highlight the value of leveraging the provenance literature to ensure correctness in interactive visualizations.

**A note on performance:** Crossfilter is an important yet computationally expensive interaction technique. The visualization community has begun adopting dense [32] and sparse [31] data cubes to support cross-filtering at interactive speeds. Unfortunately, building such data structures requires considerable offline time—from hours to the ontime flights dataset. This “cold-start” problem [5] makes it challenging for developers to rapidly build and test complex interactive visualizations, and makes it difficult to load a dataset in a visualization engine and immediately start cross-filtering.

Recent work [45] on fast fine-grained provenance engines shows that it is possible to construct whole or partial data cubes for cross-filter provenance queries in interactive time. In addition, provenance metadata can be represented in efficient index data structures that accelerate backward and forward provenance tracing lookups. These forward and backward indexes are precisely the indexes to support incremental view updates on deletion.

### 3 PROVENANCE-SUPPORTED INTERACTION

Section 2 described how core visualization interactions can be succinctly expressed in terms of provenance. This means that a visualization engine that is engineered to support provenance querying can readily add support for such interactions. Developers can then declaratively specify interactive visualizations and rely for their optimization on the underlying provenance-enabled visualization engine. In this section, we look beyond existing interactive visualization features, and examine *new functionality that may be possible* with the capabilities of such a provenance-enabled engine.

**Advanced Provenance Analysis.** To begin, we first highlight a rich area of provenance analysis techniques, such as interactive query specification [1]: what-if analysis [4, 14]; and result explanation [60] among others, that already exists. These techniques are a natural fit with a provenance-enabled visualization engine (Figure 4). First, their inputs consist of provenance metadata and user-provided information that can be naturally elicited through a
visualization interface. Second, their outputs are often in the form of predicates, records, or queries that can be naturally rendered in a visualization. Furthermore, they can be integrated as a function over the provenance result in a similar way to cross-filtering in Figure 3b. We illustrate a few examples of such integration below.

Data Explanation: Outlier explanation techniques [48, 60, 61] take as input anomalies in the visualized data, the query used to generate the visualization, and return simple predicates that are most “responsible” for those errors. Figure 4 shows how this is integrated into an interactive visualization. The user selects anomalies in the scatter plot on the left (A). Then, the analysis procedure uses $V_1$ and the fine-grained provenance of the selected points to generate a predicate explanation. Rather than print the explanation in textual form, it can be deeply integrated into the visualization itself. The example visualization recomputes the query $V_1$ over a subset of the input identified by the explanation and renders it as an overlay (B).

Why-not Analysis: Non-existence of anticipated query results play a detrimental role on the overall data exploration and analysis. For instance, if the state of California was missing in the map plot of Figure 1, then the user may be confused. Similarly, if the user complains that the COUNT of delayed flights should be higher for a specific carrier (perhaps by resizing a bar to be higher), then the user is questioning the absence of delayed flights in the visualization. Although the algorithms for generating these explanations [1, 30] may differ, the way they can be integrated into, and presented within, the visualization are similar to the preceding example.

Multi-application Linking. Visualizations contain multiple views in order to present patterns between important combinations of attributes (Figure 1). Cross-view interactions such as linked brushing and cross-filtering are powerful because they help the user identify relationships between patterns.

In terms of functionality, they combine record-level backward tracing from selections in the visualization with forward tracing to (and refreshing of) visualizations dependent on shared input data. By expressing these interactions in terms of provenance it becomes clear that the backward and forward tracing operations need not be coupled, nor even be implemented within the same visualization application. As long as different applications process the same dataset, and support backward and/or forward tracing functionality, then linking and cross-filtering across multiple applications is possible.

Figure 4: Before and after of an advanced provenance analysis. (a) the user selects outliers in the initial visualization (shown on the left), and (b) the results of the predicate explanation update the visualization (shown on the right). In practice, the visualization will update in place.

Figure 5: Provenance can enable linking and cross-filtering across different applications.

Figure 5 illustrates linking between the running visualization example with external applications such as search and user profile management. The user may use a form-based search interface to find recent flights through Miami. This result set is fundamentally the result of a query workflow over the data store but presented as a text- and image-based web application. By tracing these search results back to the input data (the red rectangle over airlines represents a subset of the relation), they can also be traced forward to update the visualization application (depicted by the red arrows). Furthermore, changing the search parameters updates both the search results and the visualization. The reverse is also possible: selecting data in the visualization can also update the search results.

Similarly, user profile tools that show to users their past flights and bookings can be linked to update the visualization to show delay statistics of the user’s past flights, as well as to update the search results with flights the user has taken. In short, any application that tracks backward provenance can issue interactions that update the presentation in any application that supports forward provenance, as long as the two ends coordinate on the same base relations.

Provenance of Interactions. So far, we have described how provenance can be used to express the results of interactions. For example, Figure 3(b) shows that the bottom bar chart is updated by re-running $V_1$ over the backward provenance of the highlighted bars in the top bar chart. In many cases, interactions simply change the inputs to the application logic (e.g., $V_1, V_2$) rather than the logic itself. In these cases, interactions are a form of input data, whose provenance and versions can be tracked.

Note that the interaction events $b$ and $c$ are data, thus we might track the provenance of the visualization interactions in e.g., a relation of events (Figure 6c shows a relation containing $b, c$). This relation lets us decouple visualization update logic from user interactions, and manage them explicitly. For instance, Figure 6c shows
Tracking and recovering Application Design Search. Thus, there is potential to develop this iterative form of refinement can help the developer find the desired interaction. This is akin to but specific to fine-grained provenance due to the expressiveness of SQL, and approaches typically focus on a semantically meaningful subset of the language for which provenance to the base and examples of desired query results, the goal is to return queries that generate the example results (or a superset). This formulation can be attractive because SQL queries are known to be hard to compose. However, the general problem is very challenging due to the expressiveness of SQL, and approaches typically focus on a semantically meaningful subset of the language for which identifying the queries by output examples can be efficient.

Earlier, we described how a wide range of visualization interactions can be decomposed into combinations of provenance queries. Thus, there is potential to develop interaction-by-example, where the user directly selects and manipulates parts of a static visualization (e.g., drag marks to new locations) to specify an example of a desired interaction. This is akin to but specific to fine-grained data visualization lineage rather than coarse-grained workflow provenance. A synthesis engine can then generate the appropriate provenance statements to support the interaction. The simplicity of provenance queries—namely coarse-grained and fine-grained backward and forward queries, along with refresh—suggests that this may be both tractable and semantically meaningful.

Deconstruction and Restyling. Harper et al. [20] present a technique to extract data from marks in D3 visualizations and re-style the data using new visual encodings. For instance, a bar chart might be restyled into a scatterplot that is colored differently. Their technique relied on D3 because it automatically annotates each mark with the record used to generate the mark. However, D3 does not track annotations across data processing workflows, thus restyling is limited to design. In contrast, tracking provenance can let users restyle the data processing, for example by plotting MAX rather than COUNT statistics, or modifying the semantics of linked interactions.

4 DISCUSSION

Provenance is a fundamental type of information with wide applications across domains. In this paper, we showed that provenance can serve as the logical underpinning of well-established, as well as novel, interactive visualization functionalities. Overall, the purpose, and corresponding takeaways, of this paper is three-fold:

First, is to convey the value of leveraging provenance capabilities and semantics to declaratively express and design visualization applications. Current visualization developers build custom data structures and make optimization choices that are coupled with interactions that can be efficiently supported; changing the visualization interactions often means rearchitecting the entire visualization application. Expressing interactive visualizations in terms of provenance introduces physical data independence, and can help developers rapidly iterate upon visualization designs.

Second, is to highlight the need for fast coarse- and fine-grained provenance engines. Traditionally, data processing engines that support fine-grained provenance expect to incur non-trivial amounts of overhead in order to quickly answer provenance queries, yet interactive visualizations are only useful if the application responds within interactive latencies. Recent work [44, 45] showed evidence that fine-grained provenance can both be materialized at interactive speeds, and be used as index data structures to accelerate visualization queries. We believe that the connections between query optimization and provenance is a rich area of research that is worthy of further pursuit.

Finally, interactive visualizations are a prominent type of data-driven interactive applications. We believe many of the connections and benefits described in this paper can translate to the general class of optimizing and expressing interactive applications.


