Trrack: A Library for Provenance-Tracking in Web-Based Visualizations

Zach Cutler* University of Utah
Kiran Gadhave† University of Utah
Alexander Lex‡ University of Utah

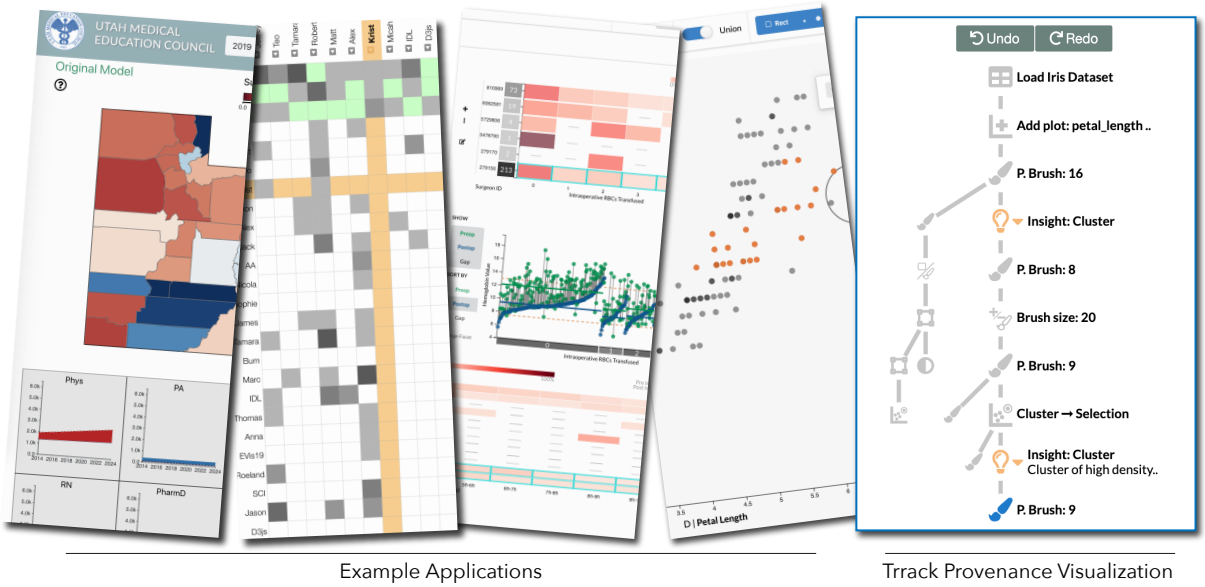

Figure 1: Four example applications using Trrack, our provenance-tracking library and TrrackVis, the associated provenance visualization library for different purposes, ranging from action recovery to logging for user studies. TrrackVis, shown on the right, utilizes custom icons, annotations, and grouping of nodes.

Abstract

Provenance-tracking is widely acknowledged as an important feature of visualization systems. By tracking provenance data, visualization designers can provide a wide variety of functionality, ranging from action recovery (undo/redo), reproducibility, collaboration and sharing, to logging in support of quantitative and longitudinal evaluation. However, no widely used library that can provide that functionality is current available. As a consequence, visualization designers either develop ad hoc solutions that are rarely comprehensive, or do not track provenance at all. In this paper, we introduce a web-based software library — Trrack — that is designed for easy integration in existing or future visualization systems. Trrack supports a wide range of use cases, from simple action recovery, to capturing intent and reasoning, and can be used to share states with collaborators and store provenance on a server. Trrack also includes an optional provenance visualization component that supports annotation of states and aggregation of events.

Index Terms: Human-centered computing—Visualization—Visualization systems and tools

1 Introduction

At least since Shneiderman argued that we cannot expect users to get it right immediately, and visualization systems need the ability to correct a sequence of actions and replay it, the value of undo/redo and replay [15] (or action recovery) has been undisputed. Even though professional desktop applications commonly support action recovery, many web-applications, and especially academic visualization prototypes, do not. For example, the authors of both Lyra [14] and Data Illustrator [9], two popular web-based data visualization authoring tools, mentioned the lack of undo in their original designs as a cause for concern for users, which resulted in less exploration. Yet, action recovery is only one of the purposes of collecting provenance data. Provenance data has many other uses, ranging from recalling an analysis process, to reproducing it, to collaborating, and to logging for evaluation or meta-analysis [13].

Designers and developers of web-based visualization tools and prototypes frequently develop custom software to capture, store, and utilize provenance information. Doing so, however, can be tedious and is often not in immediate service of the goals of a visualization prototype. Home-grown solutions are also unlikely to leverage the full potential of provenance, such as history visualization, or easy state sharing among remote collaborators.

To remedy this problem, we developed the Trrack library (the name derives from **r**eproducible **track**ing), our primary contribution, which provides provenance-tracking for the purpose of action recovery, reproducibility, collaboration, and logging. Trrack can widely benefit existing and future systems, and support data collection in quantitative and longitudinal evaluations. Trrack is designed to support a wide variety of types of provenance, including provenance of interactions, insights, and rational. Trrack is accompanied by TrrackVis, which provides optional provenance UI elements, such as simple undo/redo buttons, or a sophisticated provenance visualization that can be customized and supports annotation and aggregation of states. Finally, Trrack enables sharing states through a URL, as well as provenance data management on a server. Trrack comes with well-documented examples and uses tests to ensure code quality. Trrack is available under a permissive open-source license at `https://github.com/visdesignlab/trrack`. To show the utility of Trrack, we have integrated it into four visualization tools: a technique developed in support of a research paper [4], a visualization system for blood product management, a visualization system

*e-mail: zcutler@sci.utah.edu
†e-mail: kiran@sci.utah.edu
‡e-mail: alex@sci.utah.edu

for modeling workforce needs in the medical sector, and two network visualization tools used in a large study [10].

Our contribution is in the spirit of an **applications note** —— a short paper with the goal of informing the visualization research community of a robust, well-tested, and easy-to-integrate technology of broad interest. We believe this applications note is consistent with the renewed calls for applications and systems-oriented research that is necessary to tackle the increasing complexity of datasets and analysis challenges.

2 Design Goals

We designed the library based on our experience with developing a provenance graph for a storytelling application [5]. The overarching design goals are (a) versatility of use — the library should support all different purposes of provenance-tracking, and (b) ease of use — developers should be able to track and visualize provenance with minimal effort. Here we list our specific design goals.

Allow Developer Agency. Application developers should have the flexibility to decide which actions to track. For example, what is sensible to track might be vastly different between a production visual analytics system and a prototype used in a user-study.

Support Action Recovery. Undo/redo are important in all user interfaces, so that mistakes can be quickly recovered from. In addition to undo/redo, we want to make it possible to quickly browse to any prior state, so analysts might be willing to investigate paths they otherwise would not have if they can easily recover.

Support Reproducibility. Reproducibility of analysis processes is critical in data analysis. However, unlike analysis done in computational notebooks, interactive visualizations are difficult to make reproducible. We strive to capture not only interactions, but also annotations so that user intent and reasoning can be captured as well.

Support Collaboration. Analysts rarely work in a vacuum: they need to either share and communicate their results or collaborate with other analysts on the same project. A provenance-tracking library needs to support both. To give developers flexibility, we envision two ways of collaborating: a light-weight approach of sharing the state of an application by copying the browser URL, and a more sophisticated approach to share the full analysis provenance.

Support Meta-Analysis. We want to design our library to also support collecting information about usage, either for analysis of long-term use in the field, or for controlled studies. The requirements for action-recovery/reproducibility are different from those for meta-analysis; usually the latter requires more fine-grained logs, which can be a hindrance for the former. Our goal is to design our system such that both can be supported simultaneously. Meta-analysis also requires that the provenance data be exported in a format that is amenable to further processing. Finally, unlike traditional logging, we want to be able to jump into any recorded session, so that the context of, e.g., a performance problem, can be investigated.

Support Annotation, Highlighting, Bookmarking. As already alluded to when discussing reproducibility, users need to be able to bookmark states, so that they can be quickly found and retrieved, and annotate states, so that users or systems can provide context, including insights or the rationale for a specific action. More generally, a developer might want to store a variety of meta-information with individual states, which should be supported by a provenance-tracking library.

Provide UI Elements and Provenance Visualization. In addition to provenance-tracking, we also intend to provide user interface elements that can be used to navigate provenance data, if desired by the developer. A provenance visualization should be able to manage the whole feature set of the library, including branching states, annotations, and bookmarks. Also, as provenance data can quickly grow, it needs to be able to manage large interaction graphs.

Technical Goals A long visual analysis session can lead to a very large provenance graph if every interaction is tracked or the tracking continues for long periods of time. A library, therefore, needs to be designed with efficient storage in mind. A provenance library also should support the quick recovery of a particular state.

3 Related Work

Research on provenance capture and visualization has a long history. Here we present only a small subset of related papers that are particularly relevant to our work. We refer to Xu et al.'s recent survey [20] and Ragan et al.'s analysis for a more complete picture [13].

Provenance can be captured either through explicit workflow modeling systems [2], or by capturing the analysis process in an interactive system [11]. We are interested in the latter approach, since it can be easily integrated in arbitrary systems. This tracking-based approach requires a system to stores each state, a sequence of actions, or a combination thereof [6], with various implications for the abilities of such systems. Examples include the graphical histories by Heer et al. [6] and CzSaw [7]. Multiple systems present provenance data as node-link diagrams [8,16,19], including some of our own work [5,18]. Like Trrack, SIMProv [3] captures provenance for web-based visualizations. In contrast to Trrack, SIMProv is less modular and uses a hybrid model that primarily stores actions, which can make switching between states slower. We discuss these and other differences in Section 8.

As far as logging for evaluation is concerned, tools such as EvalBench [1] or GraphUnit [12] are examples of frameworks that provide logging capability, but neither captures a complete provenance graph that can then be re-played.

Even though undo/redo is not widely used in web applications, some systems and nonacademic software libraries provide related functionality. Many document editors, such as the Google suite of productivity tools, have some version of undo/redo. How these tools, however, are implementing action recovery is unclear, and most are likely not using publicly available libraries to do so. A few front-end libraries offer basic undo/redo capabilities. Redux (`https://redux.js.org/`) is a popular library used with React that allows users to store past and future states for undo/redo. NgRx (`https://ngrx.io/`) is a similar library for use with Angular. These libraries allow for basic undo/redo but not for complete action recovery for every previous state. Branching off from the original path will result in future states being permanently lost, and most implementations will limit how many previous states get stored. None of these libraries visualize provenance.

4 Trrack Design

Here we describe the design decisions that went into the development of the Trrack library, as motivated by our design goals. We also provide a brief architectural overview and describe how Trrack interacts with a visualization application (see Figure 2).

Trrack uses a provenance graph approach where each recorded action results in a new node in the graph. Nodes can be attached at any point in the graph, following the branching model of provenance. To utilize Trrack, developers must define a state that fully describes their application. Developers may then add observer functions to individual keys in the state. These observers will be triggered if that particular part of the state changes, either by the addition of a new state or by changing the current node in Trrack, e.g., through undo. We recommend updating only the front-end application within these observers. The only other interaction that developers must implement is to create and apply an action when a user interacts with the visualization (see Figure 2). This action is what will then create a new node with a modified state in the graph. We provide a repository with examples of different complexity to illustrate this approach at `https://github.com/visdesignlab/trrack-examples`.

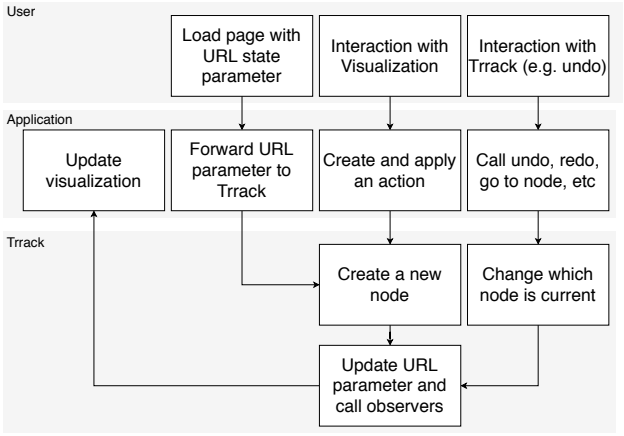

Figure 2: The relationships among the user of a visualization application, the application itself, and the Trrack library for storing provenance.

Trrack Architecture The common storage types for action recovery systems are state-based and action-based [6]. **State-based** systems store the user-defined state of the application at every node, allowing for instant jumps to any node in the history. The downside of this approach is inefficient use of memory or disk space. **Action-based** systems store the action required to get from one node to the next, which can lead to slow performance when jumping between states since actions must be applied sequentially to maintain proper recovery. The advantage of this approach is that it minimizes the storage/memory overhead. Heer et al. [6] also discuss **hybrid approaches**, where action-based systems periodically store a state to increase performance when loading a previous state.

In Trrack, we use a different approach: **differential states**. To ensure quick loading of arbitrary nodes of the provenance graph, we store states, and require developers to define a state to be used in Trrack. However, to address the size problems common with such an approach, we do not store the state at every node. Instead, we store a difference between the current node's state and the last node that stored the entire state. We track how many keys in a state object have changed, and if a heuristic threshold is surpassed, we store a full state. This approach ensures that the stored differences do not grow larger than the original state and minimizes the size of differences of ensuing nodes. Developers may also specify that a particular action should always store the entire state, if desired. This mechanism is completely abstracted from developers using the library. A developer only has to request a particular state, e.g., through the undo function, to receive a corresponding state, as illustrated in Figure 2. In addition to the state information, we also store meta-information about the actions, which is useful for visualization and to maintain a user's mental map.

Ease of Integration Trrack is a JavaScript/TypeScript library developed with the goal of making the integration of the library as seamless as possible. There are two ways to use Trrack: it can take over the state management in the application, which is most useful when designing a new web-based visualization. Alternatively, Trrack can interact with any existing state management solution to track changes to state without affecting the UI. Trrack is designed to be framework agnostic. It can work with vanilla JavaScript, UI frameworks such as React, and state management libraries such as Redux.

Sharing State The ability to share the state of an application is not commonly provided by visualization applications. Trrack allows for easy sharing by sending the current URL to a collaborator. The current state of an application is encoded and added to the URL as an URL parameter. Whenever the state of an application changes, the URL is updated. When a page with a matching URL parameter is loaded, the Trrack library parses it and returns the desired state to the application.

Persistence By default, the provenance graph maintained by Trrack is stored only in memory. To ensure reproducibility and also collaboration beyond just sharing states, we need to make the provenance graph persistent. Trrack has functionality for exporting provenance graphs, so that they can be managed the way a developer desires. We also provide a default implementation based on Google Firebase. Users can connect a Firebase database to Trrack during setup, and have every node in the graph stored in Firebase automatically every time a change is made.

Reproducibility and Capturing Intent An important aspect of reproducibility is understanding the reasoning behind steps taken by the user. For this purpose, each node in the provenance graph stores multiple types of metadata. Every node has an annotation property and a "type", a user-defined string meant to identify the purpose of the state change at that node. Additionally, a generic object may be defined by the developer and stored with the node. This metadata can be used to capture and later interpret actions taken at each node, which can range from simple bookmarks, to written annotations, to complex cases involving algorithmically predicted intents [4].

Logging for User Studies Due to Trrack's ability to reproduce entire sessions and store generic metadata, we believe it is especially useful for user studies. Typically, analysis of individual actions in a user study would require manual annotation of every user, a time intensive process. By storing labels, event types, time stamps, and generic metadata on every action, most meta-analyses of a study using Trrack can be captured automatically. This functionality can save time, reduce human error, and allow for more diverse analysis. Additionally, our differential states architecture allows for convenient analysis of stored graphs, since all of the relevant information is contained in the provenance data, independent of the tracked application. Trrack provides dedicated export functionality tailored to the needs of post hoc data analysis. Specifically, it can export details about each action that are not stored directly on the graph and not required for action recovery. Finally, Trrack makes it straightforward to jump into a specific analysis session of a user, allowing analysts to understand their context. [Here] is an example of a link to a specific state for one of our application examples.

Logging vs. Action Recovery Some actions developers want to track (e.g., for the purpose of logging) may be too frequent or inconsequential to include in the undo/redo chain. A hover action that highlights an item when the mouse rests on it is an example of this. Users would not expect to undo/redo a hover, but when analyzing logs, seeing when hover was used can be important. For these cases, Trrack allows developers to label actions as ephemeral. By default, the undo/redo functions in the library will skip over ephemeral nodes, and ephemeral nodes are also treated specially in the provenance visualization.

5 TRRACKVIS

TrrackVis is a separate library complementing Trrack to visualize the provenance graph, as shown in Figure 2. The purpose of this library is to allow for easy navigation in the provenance graph, and to provide a way to create and view annotations and metadata. The graph is shown as a node link tree. Clicking on any node will change the application's state to the one associated with that node. TrrackVis is designed to be highly customizable, allowing the user to choose to integrate features, such as tooltips or annotations. Custom icons can be added to nodes to match user-defined event types. To address the scalability of the visualization, consecutive nodes can be defined as a group, which can collapse the constituting elements. For example, groups can be used to identify specific sections of the analysis process as related and organize them as such for additional annotation. The "Insight" nodes in Figure 1 contain

nested actions that are associated with a particular insight captured in this application. By default, we also use groups to collapse nodes that are labeled as ephemeral.

6 Implementation, Testing, and Documentation

We developed Trrack over the course of 18 months, constantly refining and adapting the library to a variety of changing needs. The library has roots in an integrated application we designed for visual storytelling [5], and in a prototype library based on the concepts from that paper [17].

To encourage adoption of the library, we created a series of basic examples that demonstrate the core features of the library. The repository contains additional documentation and other examples. Trrack also includes a comprehensive suite of unit tests to ensure previous versions of the library do not break with future additions.

7 Usage Examples

We have used the Trrack library in multiple projects, spanning prototypes developed for a technical visualization paper, applications used by medical professionals, and user studies. Figure 1 shows some of the examples. Here we provide details on two of them.

We used Trrack to capture data on participants in a crowd-sourced study [10] for evaluating multivariate network visualization techniques. The study used two levels of provenance, a study-level provenance to track the progression of the study and a task-level provenance to track the interactions in a particular task. A combination of these two provenance graphs allowed us to collect data about interaction patterns in both conditions while participants completed the tasks. To explore the logged data, we developed a custom visualization, which also allowed us to jump into individual analysis sessions. The study stimulus is available at `https://vdl.sci.utah.edu/mvnv-study/`.

We also leveraged Trrack in our prototype system designed for predicting analysis intents when brushing in scatterplots [4] to capture user interactions such as selections, brushes, and adding/removing plots. The system uses this provenance trail to calculate a set of predictions for possible intents (clusters, outliers, etc.) and ranks them. We used Trrack's ability to store metadata for capturing these predictions along with relevant action nodes. Users can select one of the predictions to mark it as their intent, and provide annotations, both of which are then also stored in the provenance graph. Additionally, we captured data when we ran a crowd-sourced evaluation of this project with Trrack. The evaluation involved 128 participants and used the Firebase integration. The screenshot for TrrackVis shown in Figure 1 is taken from this project. A demo is available at `http://vdl.sci.utah.edu/predicting-intent/`.

8 Discussion and Limitations

The library most related to Track is SIMProv [3]; we discuss the main differences next, followed by a brief discussion of limitations.

Performance: Compared to SIMProv, Trrack's differential states storage model provides benefits over SIMProv's hybrid model. An action-based model can be slow when jumping between nodes that are far from each other, and due to the use of differential states, the storage requirements of Trrack are mitigated.

Easy-to-Use Exports: For user studies and log-file analysis, being able to easily analyze exported provenance data is critical. Because SIMProv uses an action-based model, the exported data refers to context-specific information, such as function names, which does not exist in the export. This context-specificity makes it difficult to analyze the data outside the original application. In contrast, Trrack's exported data is not application specific and is human readable. Every node has a state that can be used for analysis without knowing anything about the architecture of the application.

Ease of Integration: Trrack abstracts away as much of the storage model as possible. Developers simply register observer events on each property of the state. These observers then handle forward and backward navigation. In contrast, to ensure efficient navigation, SIMProv requires users to define checkpoint rules, forward changes, inverse changes, state changes, and inverse state changes, thereby increasing the complexity of the application.

Data Provenance: Although Trrack is well suited to track interaction data, it is not designed to store the provenance of datasets. For example, interactively running a normalizing procedure would create a new dataset. This dataset could either be stored directly, as an attribute of a node, or written to a separate file, and that file could be referenced as the output. Both solutions have disadvantages: the former mixes actions and data and creates a large provenance graph; the latter makes the association between data and an application state brittle. We plan on investigating the integration between data and action provenance in the future.

URL-based sharing: Sharing state through URLs may become a problem if the state is large and hence exceeds URL size limits. However, we have successfully tested this approach with over 150 keys describing a state. We believe most applications will be able to store a state much smaller than this, as long as data is stored separately.

9 Future Work

In the future, we plan to strengthen the collaborative aspects of our method. We currently enable asynchronous collaboration, but we do not track contributions by separate users. We also would like to allow synchronous collaboration, so multiple users may view and interact with the same visualization session, similar to the functionality available in Google Docs. As an immediate next step, we want to add story-editing and story-viewing capabilities to our library, similar to the approach demonstrated by Gratzl et al. [5].

With our hybrid strategy for storing states and diffs, the primary question to answer is how frequently states should be stored. We hope to do more investigation into ideal times to store states. We also hope to add functionality that optimizes the size of the graph when it is being exported by iterating over the graph and re-storing nodes as appropriate.

10 Conclusion

In this paper we introduced Trrack, a library to track provenance in web-based visualizations. Our goal is to provide an option to easily track provenance in existing or future visualization systems. The companion library TrrackVis allows for visualization of and interaction with the provenance graph. Tracking provenance with Trrack allows action recovery, collaboration, reproducibility, annotation of analysis steps, and post hoc meta analysis of the interaction sequences; and provides persistent storage with Firebase, or other custom storage solutions.

Trrack and TrrackVis are open source and published in the npm registry. We provide extensive usage examples and documentation and hope that our library will contribute to increased provenance-tracking in more visualization tools. Also, although Trrack was designed primarily with visualization tools in mind, it can also be used used with general purpose web applications.

Acknowledgments

We want to thank Sai Varun Addanki for help with the implementation, as well as Samuel Gratzl, Marc Streit, Nils Gehlenborg, and Holger Stitz for making the precursor library available. We also want to thank past and present members of the VDL team who integrated the library in projects and provided valuable feedback. We gratefully acknowledge funding by the National Science Foundation (IIS 1751238).

REFERENCES

[1] W. Aigner, S. Hoffmann, and A. Rind. EvalBench: A Software Library for Visualization Evaluation. *Computer Graphics Forum*, 32(3pt1):41–50, 2013. doi: 10.1111/cgf.12091

[2] L. Bavoil, S. P. Callahan, C. Scheidegger, H. T. Vo, P. Crossno, C. T. Silva, and J. Freire. VisTrails: Enabling Interactive Multiple-View Visualizations. In *Proceedings of the IEEE Conference on Visualization (VIS '05)*, pp. 135–142, 2005. doi: 10.1109/VISUAL.2005.1532788

[3] A. Camisetty, C. Chandurkar, M. Sun, and D. Koop. Enhancing Web-based Analytics Applications through Provenance. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):131–141, 2019. doi: 10.1109/TVCG.2018.2865039

[4] K. Gadhave, J. Görtler, O. Deussen, M. Meyer, J. Phillips, and A. Lex. Capturing user intent when brushing in scatterplots. *Preprint*, 2020. doi: 10.31219/osf.io/mq2rk

[5] S. Gratzl, A. Lex, N. Gehlenborg, N. Cosgrove, and M. Streit. From Visual Exploration to Storytelling and Back Again. *Computer Graphics Forum*, 35(3):491–500, 2016. doi: 10.1111/cgf.12925

[6] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '08)*, 14(6):1189–1196, 2008. doi: 10.1109/TVCG.2008.137

[7] N. Kadivar, V. Chen, D. Dunsmuir, E. Lee, C. Qian, J. Dill, C. Shaw, and R. Woodbury. Capturing and supporting the analysis process. In *2009 IEEE Symposium on Visual Analytics Science and Technology*, pp. 131–138, Oct. 2009. doi: 10.1109/VAST.2009.5333020

[8] M. Kreuseler, T. Nocke, and H. Schumann. A History Mechanism for Visual Data Mining. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '04)*, pp. 49–56, 2004. doi: 10.1109/INFVIS.2004.2

[9] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko. Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pp. 1–13, Apr. 2018. doi: 10.1145/3173574.3173697

[10] C. Nobre, D. Wootton, L. Harrison, and A. Lex. Evaluating Multivariate Network Visualization Techniques Using a Validated Design and Crowdsourcing Approach. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, pp. 1–12, Apr. 2020. doi: 10.1145/3313831.3376381

[11] C. North, R. Chang, A. Endert, W. Dou, R. May, B. Pike, and G. Fink. Analytic Provenance: Process+Interaction+Insight. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pp. 33–36, 2011. doi: 10.1145/1979742.1979570

[12] M. Okoe and R. Jianu. GraphUnit: Evaluating Interactive Graph Visualizations Using Crowdsourcing. *Computer Graphics Forum*, 34(3):451–460, 2015. doi: 10.1111/cgf.12657

[13] E. Ragan, A. Endert, J. Sanyal, and J. Chen. Characterizing Provenance in Visualization and Data Analysis: An Organizational Framework of Provenance Types and Purposes. *IEEE Transactions on Visualization and Computer Graphics (VAST '15)*, 22(1):31–40, 2016. doi: 10.1109/TVCG.2015.2467551

[14] A. Satyanarayan and J. Heer. Lyra: An Interactive Visualization Design Environment. *Computer Graphics Forum*, 33(3):351–360, 2014. doi: 10.1111/cgf.12391

[15] B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of the IEEE Symposium on Visual Languages (VL '96)*, pp. 336–343, 1996. doi: 10.1109/VL.1996.545307

[16] Y. B. Shrinivasan and J. J. van Wijk. Supporting the analytical reasoning process in information visualization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pp. 1237–1246, 2008. doi: 10.1145/1357054.1357247

[17] H. Stitz, T. Klaver, S. Verhoeven, M. van Meersbergen, P. Pawar, and M. Streit. Provenance core. A javascript library to create and manipulate a provenance graph, 2019.

[18] M. Streit, H.-J. Schulz, A. Lex, D. Schmalstieg, and H. Schumann. Model-Driven Design for the Visual Analysis of Heterogeneous Data. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):998–1010, 2012. doi: 10.1109/TVCG.2011.108

[19] S. van den Elzen and J. J. van Wijk. Small Multiples, Large Singles: A New Approach for Visual Data Exploration. *Computer Graphics Forum (EuroVis '13)*, 32(3pt2):191–200, 2013. doi: 10.1111/cgf.12106

[20] K. Xu, A. Ottley, C. Walchshofer, M. Streit, R. Chang, and J. Wenskovitch. Survey on the Analysis of User Interactions and Visualization Provenance. *Computer Graphics Forum*, 2020. doi: 10.1111/cgf.14035