Department

Demonstrational Interaction for Data Visualization

Bahador Saket and Alex Endert
Georgia Institute of Technology

Editor: Theresa-Marie Rhyne [theresamarierhyne@gmail.com]

***Abstract*—Recently, there has been an increasing trend to extend the demonstrational interaction paradigm to visualization tools. As more analytic operations can be performed by demonstration, new user tasks can be supported. In this paper, we discuss the properties of tasks where the by-demonstration paradigm can be effective and describe the main components needed to implement the demonstrational paradigm in visualization tools.**

Conveying a process or an outcome to someone by demonstration might be one of the oldest forms of communicating one's knowledge and intentions in apprenticeship-style learning contexts. People are effective in communicating their intended goals and results by gesturing, drawing visuals, and other forms of demonstration to guide someone else through the intended results.

Computing has leveraged demonstration metaphors for user interactions, dating back to the mid-1980s where programming by demonstration was introduced in software development.[7] The motivation behind programming by demonstration was simple and compelling: if users know how the output of a program should behave, they can demonstrate this intended behavior to create a program to generate that output.[6,7] The benefit is that instead of learning how to program, systems "watch and listen" to user actions and generate codes and programs that fit the demonstration. More recently, several other areas of computing have seen advances by applying the demonstrational paradigm, including data science, robotics, computer graphics, architecture, and others.

Data visualization has seen a recent trend to extend the demonstrational paradigm to visualization construction and visual data exploration. This raises interesting challenges, such as: what are the core components needed for tools implementing the demonstrational interaction paradigm? How are visualization tools implementing the demonstration interaction paradigm different from existing tools? And more fundamentally, what user tasks are well-suited for demonstration?

Digital Object Identifier 10.1109/MCG.2019.2903711
Date of current version 24 April 2019.

DEMONSTRATIONAL VISUALIZATION INTERFACES

Overview

Interaction is an essential part of data visualization. Interactivity engages users in visualization construction and data analysis processes[1] which foster cognitive activities of understanding data and sensemaking. For example, interactive visualization tools might provide a set of features to enable users to interactively construct different visualizations to present stories or ask specific questions. They do so through a variety of operations, ranging from modifying visual mappings, conditionally filtering data, to changing visual representations entirely.

A commonly used interaction paradigm in visualization tools is *manual view specification*. As the name implies, this interaction paradigm requires users to manually specify the desired visualization specifications and parameters through GUI operations typically designed as control panels. For instance, consider the process of interacting with data in a scatterplot. Users must specify data attributes to map onto axes, create any additional data mappings to encodings such as color, size, or shape, and finally set conditional filters to show only the relevant information. For situations where users' mental models contain this level of detail and specificity, their tasks are well-supported by manual view specification. However, there exist tasks that are harder to perform using the manual view specification paradigm since 1) *these tasks are ill-defined (it is nontrivial for users to break them down into a set of lower level operations on the control panel)*, or 2) *they are repetitive or highly customized (require users to go through layers of menus)*.

For these ill-defined, repetitive, or highly customized tasks, visualization is seeing an increase in tools implementing the demonstrational interaction paradigm for visual data exploration and visualization construction.[2–4,11,12] Demonstrational visualization interfaces allow users to provide partial demonstrations to the visual representation to convey higher level, repetitive, or ill-defined goals/tasks. Using demonstrations, the system first interprets users' intended changes and then applies/recommends potential change(s). Demonstrational interfaces tradeoff requiring users to specify visualization and system parameters for demonstrating the visual goals.

This tradeoff between by-demonstration and traditional control panel interfaces has multiple facets that designers must consider when determining if by-demonstration is suitable for the given task, visualization, or user group. For instance, by demonstration interfaces decrease intermediary graphical elements between users and systems. They reduce the need for users to translate their high level and sometimes ill-defined goals/tasks into a series of lower level operations typically specified via control panels. Despite these advantages, challenges for by-demonstration include how to correctly infer user's intentions from the given demonstrations, and how to make the potential space of operations discoverable to users. For example, coloring a data point in a scatterplot green could imply multiple meanings including coloring all data points green, mapping a data attribute to color encoding, the color that specific data point green, and others.

Components

Demonstrational visualization interfaces generally adhere to the process shown in Figure 1. This process includes four main components: *visual demonstrations, intent functions, transformation functions*, and *view update*. See Figure 1 for more details.

Visual Demonstrations: Each demonstration is a set of actions that a user takes to show parts of the expected results visually. The demonstrational interfaces enable users to provide a visual demonstration to convey their partial intended results [see Figure 1(a)]. Users might use one or more input modalities (e.g., sketch and touch) to provide demonstrations in systems implementing this paradigm.

Intent Functions: When a user provides a demonstration, the demonstration interface calls a set of intent functions. Intent functions are a set of rules that predict the potential meaning(s) of the given demonstration (i.e., extracting the user's intentions from the given demonstrations). For example, by dragging few points closer together in a scatterplot, one of the potential meanings that an intent function might predict is

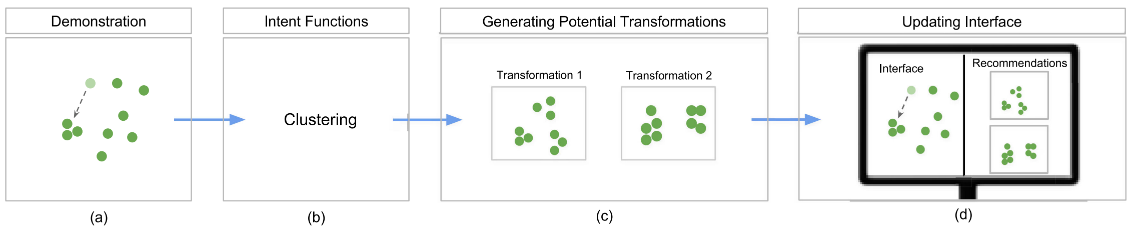

Figure 1. Conceptual diagram of demonstrational visual interfaces. Users provide demonstrations (dragging data points closer together), intent functions extract potential user intents or goals (clustering the data points), and finally, transformation functions generate possible clustering layouts to recommend.

that the user is interested in clustering those data points [see Figure 1(b)].

Transformation Functions: Transformation functions are used to compute and rank the potential changes (i.e., transformations) that can be applied to the visualization given the set of demonstrations [see Figure 1(c)]. First, a list of transformations is computed where, if applied, the final visual representation would match the given demonstration. For example, when demonstrating that two or more points should be in the same cluster, only clustering results that meet this constraint are shown. Then, these results are ranked based on transformation likelihood and fit. Transformation functions first compute all transformations which can possibly result in user's intent.

View Updates: Once possible transformations are extracted from demonstrations provided by users, the system decides how to apply/recommend possible transformations in the interface. Depending on the type of transformation, the system implementing the demonstrational paradigm might use different ways to recommend transformations. See Figure 1(d).

While these components may be implemented in different ways, the underlying paradigm remains the same. For example, tools might enable users to resize a data point by dragging a small handle on the perimeter of a glyph representing the data point or by dragging a slider revealed upon right clicking the glyph. Regardless of how these interactions are implemented to enable users to provide demonstrations, the underlying demonstrational interaction paradigm follows the same general process.

Additionally, systems may differ in their design of how to show the computed view updates. In some systems, there is only one interpretation of a given demonstration. As such, upon providing a demonstration, such systems immediately compute the expected changes and update the view. For example, sketching a scatterplot axis in SketchStory[4] always indicates the user's interest in assigning a data attribute to the axis of the visualization. Some other demonstrational visualization interfaces use heuristics to compute a list of potential interpretations of a given demonstration. In other words, these systems consider multiple potential meanings of the given demonstration and suggest them to users. For example, in VisExemplar,[3] dragging two or more data points in a scatterplot visualization closer together indicates the user's interest in either changing the axis or switching to a bar chart visualization.

WHEN ARE DEMONSTRATIONAL INTERFACES BENEFICIAL?

A practical question that is immediately relevant when considering visualization by demonstration is—*when is it beneficial, effective, or preferred over existing interaction paradigms?* From a review of tasks supported by systems in the relevant literature, as well as our own experiences designing by-demonstration systems, we discuss the tradeoffs associated with the demonstrational interaction paradigm along the following three factors.

Task Knowledge

Data visualization tasks range from low-level to high-level tasks, as discussed by Amar *et al.*[5] In general, low-level tasks require fewer parameter specifications to perform compared to high-level tasks. However, in determining whether by-demonstration or manual view specification is more effective, it is important to consider how

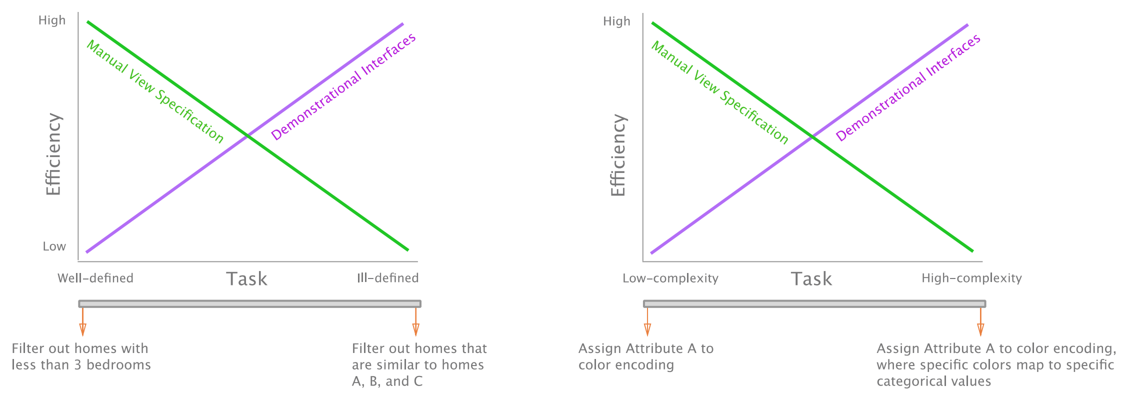

Figure 2. Task factors (e.g., task knowledge shown in the left or task complexity shown on the right) influence the potential task effectiveness and may help designers decide which interaction paradigm to use to support it.

many of these parameters that must be specified are not known to the user. In other words, if the task is *well-defined* (where all the parameters and their values are known) or ill-defined (where some of the parameters or their values are unknown) impacts the design decision about whether to support it with manual view specification or by-demonstration. See Figure 2 for more details.

For instance, consider the task of filtering data points out of a scatterplot showing homes for sale, where a user wants to filter out homes with less than 3 bedrooms. In this case, there are 2 parameters that must be specified: the operation (filter) and the criteria (value of less than 3 for the variable "bedrooms"). If both of these are known, manual view specification is an effective interaction paradigm to use. Users are capable of breaking such tasks down into a handful of operations on the control panels.

In the context of user interface design, Myers[6] also discusses the difficulties of "by-demonstration" interfaces for cases where tasks are very specific and well-defined. Myers mentions "demonstrational interfaces are harder to use in cases where the user knows exactly the relationship desired and could select it from a menu." Additionally, he discusses that demonstrating well-defined tasks may be more time-consuming than selecting among a predefined set of controls in a menu.[6]

However, even for relatively straightforward tasks such as filtering, more complex task alternatives may create situations where users do not know all the needed parameters. Taking the same example from above, what if the user instead wants to filter out homes similar to two or three she found and was not interested in? Furthermore, she does not have enough clarity at the time to define what her interests are, and thus cannot specify the exact parameters by which to filter. Instead, she could demonstrate her intent to filter out specific points (e.g., she could demonstrate to the system that she is not interested in those homes by coloring or deleting them), from which the system computes and recommends potential filtering functions and parameters.

Task Complexity

Tasks vary in complexity, based on factors including how many lower level operations they can be broken down into,[13] and how repetitive the tasks are. These factor into the design decision about which interaction paradigm best supports them (see Figure 2).

Number of lower-level operations required: The manual view specification paradigm can incur extra execution and cognitive costs especially as the number of lower level operations that a task can be broken into increases. For example, consider commonly used tasks, such as adjusting data grouping criteria (e.g., merging two bins in a histogram visualization). Currently, to perform this task in tools such as Tableau, users need to 1) select the variable and then select the Edit command from the pop-up menu. 2) In the Edit Bins dialogue, users can input new size for bins. 3) Users might also move to the next dialogue for further customization of binning. Sarvghad *et al*. leveraged the demonstration paradigm to enable users to adjust data grouping criteria.[14]

They showed that the by demonstration paradigm can significantly reduce interaction time compared to the manual view specification alternatives. Similarly, Schroeder and Keefe showed that demonstrating customized color ramps for geospatial visualizations was preferred over manual specification by artists.[15]

Repetition: One potential use of the demonstrational paradigm is for automating repetitive specification tasks and operations.[7] Imagine users are interested in assigning a specific color to all data points shown in a scatterplot or increasing the width of all bars in a bar chart. To perform such tasks in many existing tools, users have to go through several steps. For example, in Tableau, to change the size of all data points shown in a scatterplot, users first select all the data points shown on the visualization. They then need to right click and select the "format" option. A new window pops up that contains a variety of options including a slider to change the size of all data points. A demonstrational visualization interface could enable the users to perform the same task by resizing one or more data points. As a result of this demonstration, the system could then recommend the transformation of adjusting the size of all data points automatically.

Visual Analogy for Demonstration

Demonstrations are performed within visual metaphors. The ability for demonstrations to serve as visual analogies for the intended task will influence how effective they are. For example, previous work showed that dragging the tallest bar in a bar chart to the extreme left or right of an axis is an intuitive way for users to demonstrate their interest in sorting the bar chart.[8,9]

However, there exist tasks that would be more difficult to demonstrate or have more ambiguity in the system's interpretation. For instance, we recently found that users had difficulties in finding an appropriate visual analogy to demonstrate their interest in assigning a new data attribute to the axes of scatterplots.[10] Without intuitive and easy visual analogies to demonstrate an intended task or goal, the effectiveness of by-demonstration may suffer, and other interaction paradigms may be better suited (e.g., manual view specification).

Alternatively, there may be demonstrations that are too ambiguous (i.e., the system could interpret the demonstration to mean too many different tasks). For instance, if a user moves only one data point in a scatterplot, what task does that demonstrate? It may reflect a desire to shift all points, change the scale on the axis, cluster similar points, etc. In these situations, more demonstrations can incrementally help the system correctly interpret the demonstration and recommend potential transformations.

OPEN CHALLENGES

While there are examples and initial guidelines for how to design and implement by-demonstration interfaces, open challenges exist for the continued maturation of this design space. For example, the mapping between demonstrations and tasks is not clearly defined. A given demonstration could imply multiple meanings and multiple demonstrations might imply the same meaning. In practice, one of the main challenges confronting the demonstrational interfaces is how to infer the user's intent. We encourage more studies to investigate how systems should interpret user intentions from their demonstrations.

Another challenge is discoverability. How do users know the set of visualization tasks available to them? Unlike control panels that expose the functionality directly, by-demonstration interfaces rely on users knowing these tasks exist. Furthermore, users need to understand the partial visual demonstrations that trigger these tasks. Going forward, one solution may include providing signifiers or visual aids that indicate which visualization components are interactive and how users can interact with them.

CONCLUSION

We are in the nascent stages of designing and building by-demonstration interfaces for data visualization. In comparison, the design space of control panels and interface design has evolved over many years to reach the effectiveness seen today. The importance of user interaction to exploratory data analysis and visualization continues to drive innovation in this area. As the choice of interaction paradigms to support increases, the field will continue to mature and understand the tradeoffs between each of these approaches. In this paper, we discuss the exciting opportunity of by-demonstration for data visualization, and how it can benefit people in visual data exploration.

ACKNOWLEDGMENT

This project is supported by NSF IIS-1750474.

REFERENCES

1. J. Soo Yi, Y. A. Kang, J. Stasko, and J. Jacko, “Toward a deeper understanding of the role of interaction in information visualization,” *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 6, pp. 1224–1231, Nov./Dec. 2007.
2. B. A. Myers, J. Goldstein, and M. A. Goldberg, “Creating charts by demonstration,” in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 1994, pp. 106–111.
3. B. Saket, H. Kim, E. T. Brown, and A. Endert, “Visualization by demonstration: An interaction paradigm for visual data exploration,” *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 331–340, Jan. 2017.
4. B. Lee, R. H. Kazi and G. Smith, “SketchStory: Telling more engaging stories with data through freeform sketching,” *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2416–2425, Dec. 2013.
5. R. Amar, J. Eagan, and J. Stasko, “Low-level components of analytic activity in information visualization,” in *Proc. IEEE Symp. Inf. Vis.*, 2005, pp. 15–21.
6. B. A. Myers, “Demonstrational interfaces: A step beyond direct manipulation,” *Computer*, vol. 25, no. 8, pp. 61–73, 1992.
7. A. Cypher and D. C. Halbert, *Watch What I Do: Programming by Demonstration*. Cambridge, MA, USA: MIT Press, 1993.
8. R. Sadana, M. Agnihotri, and J. Stasko, “Touching Data: A discoverability-based evaluation of a visualization interface for tablet computers,” arXiv:1806.06084, 2018.
9. D. L. Maulsby, I. H. Witten, and K. A. Kittlitz, “Metamouse: Specifying graphical procedures by example,” in *Proc. 16th Annu. Conf. Comput. Graph. Interact. Techn.*, 1989, pp. 127–136.
10. B. Saket and A. Endert, “Evaluation of visualization by demonstration and manual view specification,” Computer Graphics Forum (Proc. EuroVis), 2019, to be published.
11. B. Kondo and C. Collins, “DimpVis: Exploring time-varying information visualizations by direct manipulation,” *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 2003–2012, Dec. 2014.
12. H. Kim, J. Choo, H. Park, and A. Endert, “InterAxis: Steering scatterplot axes via observation-level interaction,” *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 131–140, Jan. 2016.
13. M. Brehmer and T. Munzner, “A multi-level typology of abstract visualization tasks,” *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2376–2385, Dec. 2013.
14. A. Sarvghad, B. Saket, A. Endert, and N. Weibel, “Embedded merge & split: Visual adjustment of data grouping,” *IEEE Trans. Vis. Comput. Graph.*, vol. 25, pp. 800–809, 2018.
15. D. Schroeder and F. K. Daniel, “Visualization-by-sketching: An artist’s interface for creating multivariate time-varying data visualizations,” *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 877–885, Jan. 2016.

Bahador Saket is currently working toward the Ph.D. degree at Georgia Institute of Technology, Atlanta, GA, USA. His research focuses on the design of interaction techniques for visual data exploration. He is also interested in conducting experiments as a method to understand how visualizations can be used to support data analysis. Contact him at saket@gatech.edu.

Alex Endert is an Assistant Professor with the School of Interactive Computing, Georgia Tech, Atlanta, GA, USA. He directs the Visual Analytics Lab, where he and his students explore novel user interaction techniques for visual analytics. His lab often applies these fundamental advances to domains including text analysis, intelligence analysis, cyber security, decision-making, and others. He received the Ph.D. degree in computer science from Virginia Tech, Blacksburg, VA, USA, in 2012. Contact him at endert@gatech.edu.

Contact department editor Theresa-Marie Rhyne at theresamarierhyne@gmail.com.