Noname manuscript No.
(will be inserted by the editor)

VisAct : A Visualization Design System based on Semantic Actions

Hongjin Wu · Danqing Shi · Nan Chen · Yang Shi
Zhuochen Jin · Nan Cao

Received: date / Accepted: date

Abstract Current visualization design toolkits help users to explore data and create visualizations. However, most of these systems do not record the executed actions during the visualization construction process with semantic context. In this paper, we present VisAct, a visualization design system based on semantic actions that helps average users to construct visualizations step by step. Our system contributes a set of action-based visualization components and a high-level grammar for semantic actions. VisAct also guides the visualization construction process and provides an action tracker for history management and data collection. We demonstrate the usability of VisAct by example visualizations and a plugin application. Finally, we conduct a user study to evaluate the efficiency and effectiveness of our system.

Keywords Visualization · Semantic actions · History

1 Introduction

A full range of visualization toolkits and systems have been built to support people in the exploration of data and the construction of meaningful visualizations. Typically, these toolkits can be classified into two main categories: programming frameworks and commercial-grade visualization systems. Programming frameworks such as D3 [4], Vega [22], and Vega-Lite [21] allow users to create and edit visualizations through writing programs. These techniques are widely used by professional data visualization designers but are not user-friendly to average users. Commercial-grade visualization systems (e.g., Tableau [25], PowerBI [16], chariticulator [19], ManyEyes [26], and Lyra [20]) are developed to allow users to create statistical graphics without programming skills. These systems predefine rich visualization modules for users such as the line chart module and the bar chart module, etc. However, most of these systems do not record the executed actions during the visualization construction process in a semantic way (e.g., mapping, filtering, brushing). Instead, they collect individual visualization states which are described as basic operations (i.e., undo and redo). When manipulating these two operations, users can check the resulting changes in the visualization without knowing specific operations they have retrieved/performed. Besides, these systems record the history logs for analyzing user activity as event-based actions (e.g., mouse drops and drags). It is difficult for users to understand the semantic context of these actions and thus infer their analytical intention.

To facilitate a better understand and management in users visualization design process, some of the aforementioned systems provide history tool for users. The history tool allows users to display and manipulate (e.g., undo, redo, and revisit) their action history, thus help people better explore data and construct meaningful visualizations [11]. The existing forms of recording actions in history tool can be separated into two categories: manual capture of history actions (e.g., taking notes by users) and automatic generation of history actions (e.g., automatically generating users interaction log). However, the manual approach might cause mistakes and omissions of user actions, and the automatic method that records event-based actions (e.g., mouse drags, drops, and clicks, hover, etc.) might lack semantic meaning (e.g., the analytic and logical intention of a mouse click and drag).

To address the above issues, we present VisAct, a visualization design system based on semantic actions. Our approach helps users interpret analytical logic much easier through the display of semantic actions, and facilitates a quicker and more accurate construction process through convenient history management. During the design process of visualization, several users event-based actions, such as drag and click, are aggregated into one semantic action which represents a meaningful unit of user action (e.g., filter, sort, zoom, etc.). The semantic action is represented by

Hongjin Wu, Danqing Shi, Nan Chen, Yang Shi(✉), Zhuochen Jin, Nan Cao
Intelligent bigdata visualization lab, Tongji University, Shanghai, China
E-mail: yangshi.idvx@tongji.edu.cn

our proposed high-level visualization grammar, which will be introduced in Section 3.3. These semantic actions can be used to constitute a sequence. Given the sequence of semantic actions, visualization components can be triggered, and then the instant result of visualization design is displayed. Our system also provides an action tracker which tracks all users semantic actions and then transports these actions to the provided history panel. The history panel receives and displays these actions, and allows users to manage (e.g., undo, redo, reset, and revisit) their history. Moreover, our system provides interactive guidance that leads average users to construct visualizations step by step. The steps of the interactive guidance include setting the information of visualization, uploading data, selecting the visualization component, choosing data attributes for mapping, and exporting the final work. Also, VisAct enables users to publish and share their visualizations via the public gallery.

The key contributions of our paper can be summarized as follows:

- We design a visualization system VisAct which contains interactive guidance of visualization construction process and an action tracker to record all user actions with semantic meaning.
- We propose a high-level grammar for semantic actions and a set of action-based visualization components in the system. These components can be constructed by the proposed grammar.
- We conduct an user study to evaluate the effectiveness of VisAct, compared with a traditional visualization tool. The result of our user study reveals that VisAct exhibits better guidance for average users, make the visualization design process more understandability for them, thus maintains good usability.

The rest of the paper is structured as follows: The discussion of related work is demonstrated in Section 2. We introduce our design process in Section 3, including system overview, interface and interaction, and action-based visualization design. In Section 4, the implementation of our system is presented. The visualization constructions on multidimensional data and graph data approaching by VisAct and office add-in of our system are illustrated in Section 5. In Section 6, we offer the evaluation of our system VisAct by conducting an user study and then discuss our results. Finally, Section 7 summarizes our contributions, discuss the flexibility and expressiveness of our system, and clarifies the future work.

2 Related Work

Long-standing research on informative visualization toolkits and systems provides useful references for our research. We introduce relevant previous work by classifying discussion of visualization Grammars and Programming Frameworks, and interactive Systems for Visualization Construction.

2.1 Visualization Grammars and Programming Frameworks

Visualization grammars and programming frameworks provide an essential way to construct information visualization. Most of these grammars are declarative languages [10] and can be separated into two categories: Low-level grammars and high-level grammars.

Low-level grammars, such as ProtoVis [3] and Vega [22], are powerful tools to create sophisticated and highly-customized visualization designs. They help visualization designers to specify visualizations by a direct visual mapping without computational details. D3 [4] is the most popular framework for web development, which requires designers to operate DOM elements with JavaScript code. However, these grammars are only popular among users who have enough visualization knowledge and programming skills.

High-level grammars ignore intricate programming details and focus on rapid visualization constructions. ggplot2 package [28] for R language proposes a high-level grammar that builds up the visualization from multi-layers of data. Vega-Lite [21] provides another alternative grammar of interactive data visualizations based on Vega. ECharts [14] provides a more flexible grammar to find the balance between the simplicity of usage and the complexity of visualizations. These grammars are useful to explore the visualization design space.

We implement these two type of grammars for our system. More precisely, we develop visualization components using low-level grammar D3. For system usage, we provide a high-level grammar based on semantic actions [9], which is more understandable for end users compared to event-based actions.

2.2 Interactive Systems for Visualization Construction

Over the past decades, research works took effort to the construction of interactive toolkits or systems for information visualization. These tools and systems help people explore data and construct meaningful visualizations. The existing works include but not limited to Many Eyes [26], Lyra [20], iVisDesigner [18], Data Illustrator [15], Charticulator [19], Spotfire [2], Sense.us [12], Tableau [25], and Improvise [27]. These systems simplify the construction process of visualizations by predefining several visualization modules for users. Using this approach, users without

the experiment in textual programming can also utilize these toolkits and systems to create visualizations. Most of these systems model the histories as changes through a graph of visualization states and allow users to undo/redo the operations during the design process.

Visualization tools and systems also help users to find the analytical logic and rationale of their visualization process. There are mainly two approaches in this area: the manual approach to record and the automatic method record history actions. The manual approach allows users to capture their historical activities manually, including taking notes by users [8]. This method promotes the high-level rationale recording of history activities thus is very useful for users. However, mistakes and omissions of specific history action appear during the recording process. A more practical and automatic method [13, 23] are needed to avoid this drawback. For example, VisTrails [23] automatically records the analytical actions of user visualization without manual operations. However, these methods are limited to recording low-level activities like mouse drags and clicks. These low-level activities cannot be easily interpreted by users, e.g., inferring the purpose behind a mouse drag can be challenging.

Compared to existing research on manual visualizations systems and visual analytic systems equipped with history tools, our system provides a more flexible and understandable history tool for users. More precisely, we proposed a high-level grammar for semantic actions (e.g., mapping data attributes to specific visual channels). We record user semantic actions to represent the visualization design process and then display them in the history tracker. Users are able to review and interpret their visualization process, resulting in an easier way to revisit/reset/undo/redo a certain intermediate action in the visualization process to meet a new goal.

3 The Design of VisAct

In the section, we introduce the pipeline and design details of VisAct. First, we give our system framework, which includes workflow layer, visualization components layer, and action tracker layer. Then we discuss the user interface and interaction in VisAct. Finally, we proposed a high-level semantic grammar for action-based visualization design.

3.1 System Framework

VisAct is designed to empower average users to explore data and construct visualizations interactively. We identified three design requirements (R1-R3) for our target.

R1 **Clear Guidance**. Most everyday users without visualization experience have little idea about how to construct a visualization from zero. The system should properly guide users to design data visualizations step by step.

R2 **Understandable History**. It is important to let users know what they have already done as well as what to do next. Most graphics tools keep history in the low-level actions (e.g., mouse clicks, drags, drops, etc.) which analytic and logical intention is vague for users to review and revisit. The system should design a more meaningful form of action history for average users.

R3 **Flexible Operation**. The flexibility in the visualization system means users can explore visualization design space easily. It also means that users can switch to any previous states of the visualization process immediately. The system should allow users to take atomic action and provide an interactive history interface for the management of history actions.

The system framework is designed to meet these three requirements and is illustrated in Fig. 1. The system includes three layers: (1) *Workflow Layer*, (2) *Visualization Components Layer*, and (3) *Action Tracker Layer*. Actions designed in Section 3.3 are used to transfer information among these three layers.

Workflow Layer (Fig. 1(a)) focuses on the pipeline of visualization construction. It contains five vital interactive modules to help users to design visualizations step by step (**R1**). *Data* module is the first module to handle the dataset

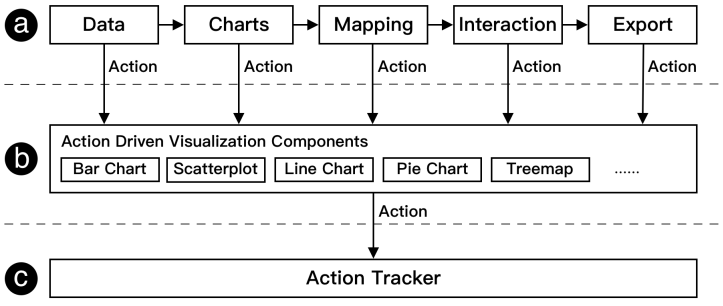


Fig. 1 The system framework of VisAct including (a) workflow layer, (b) visualization components layer, and (c) action tracker layer.

uploaded by users and support all data-related operations (e.g., sort, filter, etc.). *Charts* module is the interface to choose visualization type (e.g., line chart, bar chart, and so on.). *Mapping* module enables users to map data attributes to the visual channel freely. Data attributes and visual channels are based on *Data* module and *Charts* module, respectively. *Interaction* module allows users to interact with the visualization to modify the visualization result in detail. For instance, mouse drag can alternate the layout of DICON visualization, and mouse click can make a highlight in scatterplot visualization. *Export* module is the final step to help users to download, or publish their work to our public gallery. *Visualization Components Layer* (Fig. 1(b)) provides the reusable charts that can be triggered by actions from *Workflow Layer*. To support the semantic actions which represent meaningful user activities (**R2**) and atomic design step (**R3**), we design a set of action-based visualization components and a high-level visualization design language. We follow the semantic actions characterized by Gotz and Zhou [9]. *Action Tracker Layer* (Fig. 1(c)) records the actions reported by *Visualization Components Layer*. This layer maintains users' action history. It provides the history tool and uploads the actions to the server for future research purpose. The history tool supports the interactive guidance of visualization construction process. It facilitates analysis by enabling undo, redo, and revisit previous states (**R3**).

3.2 Interface and Interaction

We discuss the user interface and user interactions in VisAct. The overall user interface of VisAct (see Fig. 2) can be split into four essential parts. The left part is the **Wizard Panel** (Fig. 2(a)), showing the workflow for visualization construction. The middle part is the **Chart Editing Panel** (Fig. 2(b)), displaying the resulting visualization of user actions and allowing users to edit. The right part is the **Bookmark Panel** (Fig. 2(c)), illustrating the visualization state bookmarked by the user. User actions were recorded by the action tracker and shown in the **Action History Panel** (Fig. 2(d)). Besides, VisAct also includes personal projects page and a public gallery for registered users to manage and publish their visualizations.

3.2.1 Wizard Panel

Wizard panel contains the basic guidance for visualization construction. The navigation buttons on the bottom of the wizard panel guide users to design visualizations step by step. The tab buttons on the top allow users to jump to any step with a simple click.

Information tab is the first part of the wizard panel. Users can input the necessary information about their visualization design, such as name and description, etc. Users can search for this information, and then the corresponding visualizations relevant to the keyword will be displayed in the users' personal projects page. Projects are set to be private by default and can be transformed into a public project by clicking the switch button.

Data tab is the place to upload and process data. Users can click the upload button to submit their data in CSV format. The system loads the data and then transforms it into an internal representation. Data table displays the whole dataset, including field name, types of data attributes, etc. The basic data actions like sorting or filtering can be performed in the interactive table header.

Charts tab lists all the chart type accessible in our system. Users can select an appropriate visualization chart for their data. The chosen chart decides the specific setting in the mapping tab.

Mapping tab allows users to design visual encoding using drag-and-drop interaction. In Fig. 2(a), the data attributes and the visual slots are listed in the panel. It supports a search box for the data attributes in case of high-dimensional datasets. Some visual channels with a red dot means necessary encodings for the chosen chart. For example, in Fig. 2(a), *X* and *Y* have to be filled by data attributes, but *Color* and *Size* can be empty.

Export tab allows users to export their final visualization design as an image. It can also interact with other applications to insert the visualization into Microsoft PowerPoint and other software.

3.2.2 Chart Editing Panel

Chart editing panel draws the current specified chart. It supports chart-based operations according to the user chart choice. For most of the charts given by VisAct, users can zoom and pan the chart to change the view, move graphical elements in the visualization, and inspect the visual elements for data detail in this panel.

3.2.3 Bookmark Panel

Bookmark panel is the list of visualization states that saved by users during the exploration process for future investigation. Bookmark panel supports snapshots, text notes, and bookmarks timestamps that help users recall these meaningful views.

Fig. 2 The user interface of VisAct consists of four essential parts, which includes a wizard panel, a chart editing panel, a bookmark panel, and an action history panel. Wizard panel (a) contains the basic guidance for visualization construction. Chart editing panel (b) draws the current specified visualization state. Bookmark panel (c) is the list of visualization states saved by users. Action history panel (d) displays a sequence of all the semantic actions during the visualization construction process.

3.2.4 Action History Panel

Action history panel displays a sequence of all the semantic actions of a user. As shown in Fig. 2 (e), the action history displays five latest actions. The scatterplot is chosen by the user in the charts tab. Then, the user triggers three *Mapping* actions to map data attributes to the visual elements including *X*, *Y*, and *Color*. Finally, the user zooms the scatterplot into a specific range in the chart editing panel. The details of each action can be reviewed by clicking the corresponding action. The action detail view will pop up with a confirm button for a revisit. Users can click the confirm button to get back to this specific visualization state. The *Revisit* action will change the display in the wizard panel and the chart editing panel.

3.2.5 Public Gallery

One of the crucial goals of the system is to expose the visualization technology to the broadest possible audience. To achieve this goal, VisAct is built as a web-based visualization system which allows users to publish and share their visualizations in a public gallery. When creating visualizations, people can select the private mode or the public mode of their works. The public visualizations shared by registered users are collected and listed on the public gallery, which is the homepage of VisAct. Later users can view listed visualizations on the site and view the details of the interesting work by a simple click. During the exploration, analysts can download the source dataset of a public visualization, or edit the work online.

3.3 Action-Based Visualization Design

Action-based visualization design means the visualization construction process can be defined as a set of actions. These actions should represent strong semantic meaning to help users to understand. In our system, we define the semantic action based on the characterization of users visual analytic activity [9]. Each semantic action is the combination of a series of event-based actions (e.g., mouse clicks, drags, drops, etc.). They are also domain-independent that can be applied to a broad range of applications and tasks.

An *action* identies an atomic and semantic step performed by visualization designer. The representation of the action should include both semantic properties and specific parameters. Formally, the action definition can be represented with a three-tuple:

$$action := (category, type, parameters) \tag{1}$$

Category	Type	Parameters	Description
data exploration	upload	data URL	load dataset.
data exploration	change	data URL	change dataset.
data exploration	sort	data field; order	re-arrange the visual representation in ascending (descending) order.
data exploration	filter	data field; filter condition	reduce the dataset according to filter condition.
visual exploration	change chart	chart type	change chart type (e.g., bar chart, line chart, etc.).
visual exploration	mapping	operation;data field;visual channel	map data attributes to visual elements.
visual exploration	zoom	zoom scale	scale a visualization in a new range.
visual exploration	pan	pan offset	scroll a visualization to a new location.
visual exploration	reconfiguration	layout detail	change the layout of current visualization.
visual exploration	brush	brush range	highlight a range of visual elements.
visual exploration	inspect	element ID	check the detail about a visual element.
visual exploration	restore	bookmark state	restore to a saved bookmark state.
meta	undo	—	remove the latest performed action.
meta	redo	—	the inverse of undo.
meta	reset	—	return to the original state.
meta	revisit	action ID	return to a previous state.
insight	bookmark	text note	save current visualization state into bookmark list.
insight	save	—	save current visualization state.
insight	export	—	save current visualization state as image file format.

Table 1 The set of semantic actions identified in our visualization design grammar.

The *category* defines the top-level category of the semantic action. We identify four categories of actions: *data exploration* actions, *visual exploration* actions, *insight* actions, and *meta* actions. The *type* identifies a unique semantic action. The *parameters* specifies the values required by the implementation actions. Table 1 lists all the semantic actions defined in our system. In the next four paragraphs, we introduce these actions under four categories. *Data exploration* are data related actions including the *Upload*, *Change*, *Sort*, and *Filter* actions. The *Upload* action means uploading datasets. The *Change* action replaces the current dataset with a different one. There are other data exploration actions, including traditional data operation behaviors like *Sort* and *Filter*.

Visual exploration actions are performed to change the visualization presentation of the dataset and can be represented as the *Change Chart* action, the *Mapping* action, and so on. The *Change Chart* action is the key request by the user to change the chosen visualization component to another one in the charts tab to present information. For example, users can change from the scatterplot to the scatterplot matrix to present multidimensional data. The *Mapping* action is the most critical action in our system, which maps the data attributes to the visual channels, such as position (*x*,*y*), *color*, *size* and so on. Other actions including *Zoom*, *Pan*, *Brush*, *Inspect* and *Reconfiguration* action change the visualization presentation based on the user interaction. For example, users may zoom in to a specific area in the scatterplot to display the information that they intend to show. *Restore* action is performed by users to revisit a saved bookmark of previous visualization state.

Meta actions are used to operate on the user's action history. *Undo* and *Redo* are the function of backing to the prior state of the visualization or withdrawing *Undo* operation. These are very common operations in history tools. *Revisit* action is performed to return to the visualization state of a certain semantic action in the history panel. *Reset* action brings the visualization back to the original state. These *Meta* actions are essential to help designers to compare and switch between current and earlier visualization design.

Insight actions are usually performed by users when they discover a meaningful result during the exploration. *Bookmark* action save the current visualization state and leave a note for future reviewing and restoring. *Save* and *Export* action are used in visualization system to store and export the visualization state.

We propose a high-level grammar for visualization design based on the definition of the semantic action. Fig. 3 presents an example of the action-based visualization design. The grammar is in JSON format (Fig. 3), which is easy to store, execute, and understand. Formally, our visualization design grammar is represented as an action sequence:

$$visualization := [action_1, action_2, ...] \tag{2}$$

The visualization specification of users' history is designed to record action which category is data exploration or visual exploration in the order of execution one after another. The action sequence in the specification will be reordered according to meta actions. For example, the actions executed before reset action will be removed in the specification. Besides, the insight actions (e.g., bookmark, save, export) are not contributable to the construction of the visualization process, thus are not included in the specification as well.

4 Implementation

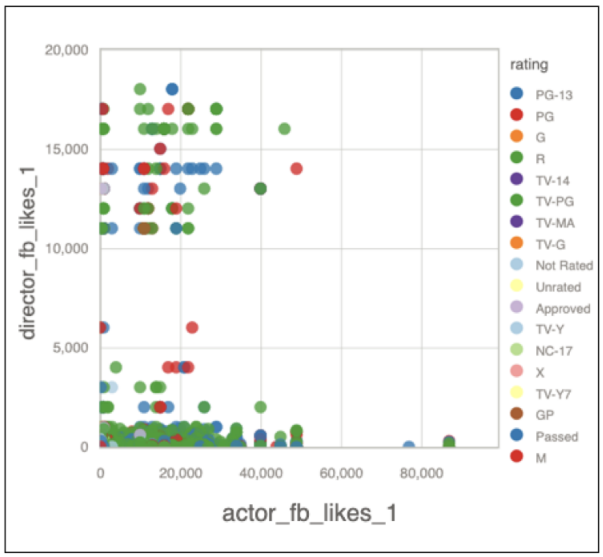
The implementation of the whole system includes three parts: front-end application, back-end server, and visualization components. The front-end application of the system is a web-based application, which is developed using React and other open-source libraries. The application is a thin client which accesses the rich functionality using API provided by the back-end server.

The back-end server is responsible for the management of the user's account, datasets, and visualizations. It includes a server application written in Node.js, a file server for uploaded data, a relational database for user metadata, and a NoSQL database for action logs. The visualization components are a set of charts implemented in JavaScript using programming framework D3. All the charts provide the same action API, which follows the high-level grammar based on semantic actions we proposed in Section 3.3. We offer several basic statistical charts in the system. To make our system more extensible, we implement a chart extension architecture for the chart developers to add their own charts to the VisAct. The workflow is shown in Fig. 4. We provide a web page for developers to upload their own chart extensions (Fig. 4(a)). One chart extension includes three parts: a configuration file, a client-side extension file, and a server-side extension file (Fig. 4(b)). The configuration file is a JSON file which describes the basic information about the chart, such as the unique id, name, description, and the maximum data count to support. After the chart extension is uploaded, the configuration will be stored in the server database (Fig. 4(c)). The client-side extension is the JavaScript file that supports the interface and interaction for the chart. It has to follow the API defined by our proposed high-level grammar based on semantic actions. The file is bundled by webpack [1] and is stored on the file server. The server-side extension is another JavaScript file providing the data processing function and layout function that needs to be executed on the server. As illustrated in (Fig. 4(d)), the client JavaScript file will be loaded by front-end application using AMD (Asynchronous module definition) [30] specification. The server JavaScript file will be loaded by back-end application with *eval()* function.

(1)
(2)
(3)
(4)
(5)

```
[
 {
  "category": "Data Exploration",
  "type": "Upload",
  "parameters": {"url": "movie.csv"}
 },
 {
  "category": "Visual Exploration",
  "type": "Change Chart",
  "parameters": { "type": "scatterplot" }
  },
 {
  "category": "Visual Exploration",
  "type": "Mapping",
  "parameters": {
     "operation": "Add",
     "slot": "X",
     "dimension": "actor_fb_likes_1"
  },
 {
  "category": "Visual Exploration",
  "type": "Mapping",
  "parameters": {
     "operation": "Add",
     "slot": "Y",
     "dimension": "director_fb_likes_1"
  },
 },
 {
  "category": "Visual Exploration",
  "type": "Mapping",
  "parameters": {
     "operation": "Add",
     "slot": "Color",
     "dimension": "rating"
  },
 },
 {
  "category": "Data Exploration",
  "type": "Zoom",
  "parameters": {
     "X": {"start": 0, "end": 100000 },
     "Y": {"start": 0, "end": 20000 }
  }
 }
]
```

(a)



(b)

Fig. 3 (a) Action-based specification (b) The corresponding visualization of movie data. It takes six semantic actions in visualization specification: uploading movie dataset, changing to the scatterplot, adding data mapping for *X*, adding data mapping for *Y*, adding data mapping for *Color*, and zooming into a specific range.

5 Applications

In this section, we show a gallery of visualization examples created by the participants in our user study. These examples use a variety of datasets including the scatterplot matrix for visualizing multidimensional data, the treemap for visualizing hierarchical data, force-directed graphs for visualizing graph data, and a novel visualization tool for visualizing multidimensional data. Our system also is applied as an add-in application for office software such as Microsoft Powerpoint [17].

5.1 Visualization Design Examples

The existing chart types in our system including line chart, bar chart, scatterplot, histogram, pie chart, area chart, scatterplot matrix, treemap, boxplot, DICON [5], and force directed graph. Here, we show four types of visualization including scatterplot matrix, treemap, Force-directed Graph, and DICON.

5.1.1 Scatterplot Matrix: Multidimensional Data

We first illustrate the flexibility and expressiveness of our system by visualizing multidimensional dataset using scatterplot matrix. We choose wine quality dataset [6] provided by UCL Machine Learning Repository for the illustration. The red wine quality data includes 4898 instances, and each instance has 12 attributes, including fixed acidity, volatile acidity, residual sugar, and so on. Residual sugar, fixed acidity, chlorides, and alcohol are selected as vertical variables. These data attributes are also interpreted as horizontal variables due to the set of our system; We show the scatterplot matrix of red wine quality dataset in Fig. 5(a).

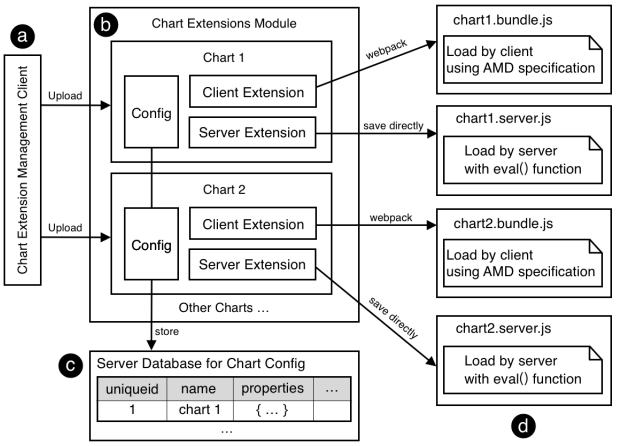


Fig. 4 The workflow of chart extension architecture. Management web page is the entry for developers to upload their charts (a). Chart extension module processes all the uploaded chart files including config file, client extension, and server extension (b). The config information is stored in the server database (c). The client extension and server extension are stored in the file server and will be loaded by client and server application respectively.

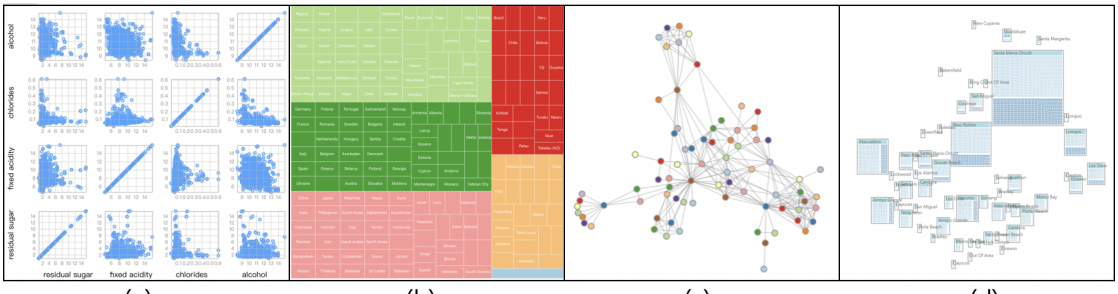


Fig. 5 Visualization design examples created by VisAct : (a) Scatterplot matrix of multidimensional data; (b) Treemap of hierarchical data; (c) Force-directed graph of graph data; (d) DICON visualization for multidimensional data.

5.1.2 Treemap: Hierarchical Data

Hierarchical data refers to the dataset that data items in it are hierarchically related to each other. Our system supports users to visualize it using a famous method (treemap) for displaying tree-structured data as some nested rectangles. User can upload their hierarchical dataset as a CSV format, and then choose appropriate data attributes into the hierarchical shelf. We use the information of Sovereign states and dependencies by population provided Wikipedia [31]. We group listed countries in the site by continents, and then collect the information in a table with 207 rows and 3 columns. The three column names are continent, country, and population. Dragging the continent attribute into the hierarchical shelf, and then dragging the country attribute into another hierarchical shelf. These actions mean the area will be separated into sub-rectangles according to the continent, and each sub-rectangle will be separated into smaller rectangles according to the country. The color of each sub-rectangle is encoded by the information of the continent. The visualization of this dataset is shown in Fig. 5(b).

5.1.3 Force-directed Graph: Graph Data

A graph refers to the data set which depicts nodes and co-occurrence of each node and is widely used in real-world scenes, such as character relationships, citation network, etc. Force-directed graph is a traditional visualization method for visualizing connections between nodes in the network. In practice, the graph dataset we used is the dataset [7] depicts the character co-occurrence in Victor Hugos Les Miserables. The required dataset consists of two CSV files. One contains links of the network, including the source and target vertices plus the value of the edge between each pair vertice. Another CSV file contains nodes of the network, including the name of each note, the group that the node belongs to, and the node size correlated with the frequency of use of the tag. We map the id attribute into the node shelf, and drag the source attribute and target attribute as the source shelf and the target shelf, respectively. The visualization design for the graph data is shown in Fig. 5(c).

5.1.4 DICON: Visualization of Multidimensional Data

Cluster analysis is an essential approach for analyzing for multidimensional datasets, and it is applied to many areas. DICON [5] is a novel visualization tool for visualizing multidimensional clusters that promotes the interpretation and manipulation of clusters. Our system has implemented the DICON chart for users to analyze clusters. The visual encoding shelf of DICON consists of the feature shelf and the group shelf. Users can upload their data, choose the DICON component, drag numerical data attributes into the feature shelf, and drag categorical data attributes into the group shelf. We choose house dataset [29] downloaded from the Wiki server of Cal Poly Computer Science Department Labs. This dataset is a table with 782 rows and 8 columns, illustrating the information of houses in San Luis Obispo County and its neighborhood. The information includes a unique ID, price (in dollars), location, the number of bedrooms and bathrooms, size (in square feet), and type of sale (including Short Sale, Foreclosure and Regular). We choose price into the feature shelf and location into the group shelf. The resulting DICON visualization of house dataset is presented in Fig. 5(d).

5.2 Office Add-in

When using Microsoft PowerPoint to make slides and deliver a presentation to the public audience, people need to create powerful data visualizations. The most common way is creating visualizations by visualization tools, saving them in image file formats, and then adding them into Microsoft PowerPoint. This method of work scrambles the process of users' day-to-day work arrangements, especially when they need to create lots of visualizations and reuse them in work. VisAct provided an alternative way to complete this process by implemented itself as a plugin to Microsoft PowerPoint, to make the workflow of editing slides and creating visualizations into a whole. By not disturbing the process of adding textual content into PowerPoint, VisAct add-in helps users to think deeply of their creating. VisAct can also apply to other office software (e.g., Microsoft Word).

To use VisAct in Microsoft PowerPoint, users need to search and append VisAct in the add-in panel of Microsoft PowerPoint. The interface of VisAct add-in in PowerPoint is shown in Fig. 6. The traditional user interface of slides is still the main part of PowerPoint (Fig. 6(a)). After clicking the top button in toolbar (Fig. 6(b)), VisAct add-in will appear on the right side, which contains the chart editing panel in the top (Fig. 6(c)) and the wizard panel in the bottom (Fig. 6(d)). The bookmark panel and the action history panel is hidden, and they will be shown by clicking the top buttons (Fig. 6(e)). To optimize users' browsing experience in the narrow right sidebar, we apply the responsive design to [32] VisAct. Users can log in their accounts to sync their visualization projects. The process of creating visualizations in VisAct add-in is similar to the workflow of the web version. The visualization designed in add-in can be export into current PowerPoint document as an image format.

6 Evaluation: VisAct vs. Polestar

We conducted a within-subject user study to evaluate the efficiency and effectiveness of our system compared with a traditional visualization system, Polestar. We hypothesize that our system performs better on these two aspects than the baseline system.

6.1 Study Design

Participants are asked to interact with two visualization systems: VisAct and the baseline system. For each system, we prepared a dataset with practical meaning for participants to complete three tasks. In order to counterbalance potential learning effects during the study, the order of the three tasks was intermixed across participants.

Visualization Tools. Users were asked to use two visualization systems: VisAct and Polestar, a tableau-style visualization system built on top of Vega-Lite [21]. Polestar is a manual visualization tool for visual analysis that allows users to create visualizations on their uploaded dataset. It is more concise and unlikely to bring external factors to our user study when compared to other systems.

Datasets We offered two datasets for users to explore. One is a dataset of movies which contains 3202 rows and 16 columns. The columns include title, director, genre, creative type, IMDB votes, and rating scores in IMDB and Rotten Tomatoes, etc. Another is a dataset of car, which is automotive statistics for a variety of car models between 1970 and 1982. It is composed by name, year, acceleration, displacement, horsepower, miles per gallon, etc. The table as 406 records and 8 variables. We choose these two datasets because they are statistics from the real world, thus users can gain latent information with practical meaning through visualizations.

Participants. We recruited 17 participants (8 women, 9 men), ranging in age from 17 to 35 years, with an average age of 24. All of them are average users who have no programming skills and professional knowledge of visualization. No subjects have used Polestar and VisAct to analyze data before. We identify the participants as P1-P17, respectively. The study session lasted approximately 1.5 hours. After the completion of tasks, we offered a present for participants to express our appreciation for their time and efforts.

Study Protocol. The study session begins with a 10-minute brief introduction of VisAct and Polestar. After the tutorial, participants were asked to complete three tasks using each of the two visualization tools. The description of the tasks is listed as follows. **T1)** Create a bar chart of the dataset and find which value of attribute A corresponds to the highest value of attribute B. **T2)** Create a scatterplot of the dataset and find the range of attribute A that attribute B mainly concentrates on. **T3)** Create a line chart of the dataset and find whether there is a trend of increasing attribute A over attribute B. The order of the tasks in Polestar and VisAct are set to be different to counterbalance potential learning effects. Participants can end the study session until they have finished tasks and answered all the questions.

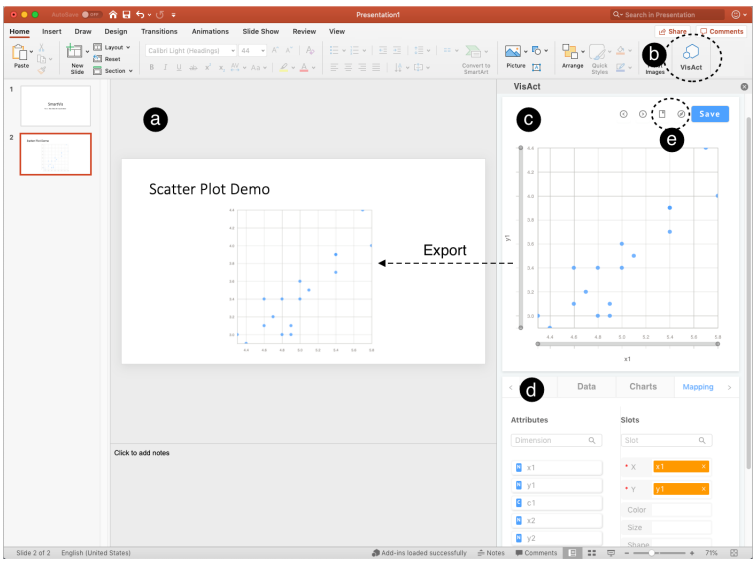


Fig. 6 VisAct is added in the left side in Microsoft PowerPoint as a plugin. Users can create visualizations in PowerPoint. The procedure of the usage is similar to the web version of VisAct.

We held the study sessions in a laboratory setting. VisAct and Polestar were both visited in Google Chrome on iMac desktop computers (3.2 GHz Processor, 8 GB RAM) with a 27-inch retina display. After completing the study sessions, participants filled out a questionnaire and took an interview for feedback. After the completion of tasks, users were asked to fill out a questionnaire with 5-point Likert-scale (-2 = Strongly Disagree, -1, 0, 1, 2 = Strongly Agree) questions. Then, we gave a short interview about the user interface usability, interaction and experience of two systems during completing tasks.

Collected Data. We collect four kinds of data in the user study: (1) Final visualization design results for each task were stored after the study session; (2) Interaction log files of every user's visualization process in Polestar and VisAct; (3) The questionnaires completed by all participants were collected for result analysis; (4) Audio of the user interview was recorded and finally was recorded into text format.

6.2 Analysis & Results

We present the study results in terms of the task completion steps, correct rate, ratings of the questionnaire, and users' qualitative feedback.

Task completion steps and correct rate. A paired-samples t-test was conducted to examine the result. The result indicates that the completion steps in VisAct (M = 6.94, SD =0.16) are significantly fewer than in Polestar (M=10.94, SD=0.31; $p<.01$). The correct rate in VisAct (M=1.0, SD=0.1) is significantly higher than in Polestar (M=0.82, SD=0; $p<.01$).

Questionnaire. After the completion of tasks, participants are asked to reflect on their experiences with VisAct and Polestar. The average results of 5-point likert-scale questionnaire are: After the completion of tasks, participants are asked to reflect on their experiences with VisAct and Polestar. The average results of 5-point likert-scale questionnaire are: "The system is {easy to use (VisAct: M=1.76, SD=0.51; Polestar: M=1.43, SD=0.62), easy to understand (VisAct: M=1.82, SD=0.04; Polestar: M=0.11, SD=0.28), expressive (VisAct: M=1.57, SD=0.23; Polestar: M=1, SD=0.45), convenient for collaboration (VisAct: M=1.42, SD=0.51; Polestar: M=-2, SD=0)}" and "the system has {a smooth interaction (VisAct: M=1.71, SD=0.51; Polestar: M=1.21, SD=0.3), a clear guidance of visualization process (VisAct: M=1.47, SD=0.51; Polestar: M=-1, SD=0.62), various visualization components (VisAct: M=1.24, SD=0.51; Polestar: M=1.12, SD=0.03)}", and "The whole design process is more fluid and flexible (VisAct: M=1.76, SD=0.51; Polestar: M=0.12, SD=0.62)". The result of the designed questionnaire reveals very positive feedback on our system by users.

Participant feedback. In the post-interview, participants elaborated their experience during their study sessions. Our system gained positive feedback that our system makes the visualization process much easier for average users, and the history management is flexible and understandable. Participants feedback reinforces the previous results. They appreciated the support for the clear guidance and history view in VisAct.

P3 said, "*When I was using VisAct, the beginning guidance leads me to upload data, choose chart type, and map data attribute. Each chart type has a thumbnail which makes me find the right type in a short time. But in Polestar, I can't figure out what to do next and consume about 8 minutes to try those buttons to see if my operations were right.*" His comments point out the advantage of the guidance in our system. P7 mentioned the history tracker in VisAct, he said: "*I found that all my history actions are recorded below the site, and I can easily switch to any state of my previous steps. This functionality helps me understand what I have done, and save lots of time due to the needless of restart*". Moreover, some participants proposed meaningful suggestions that point out future work. P8 said, "*VisAct is not adaptable in some browsers due to the inappropriate adaptation ratio*". Another participant P17 suggested that it would be better for her if Attribute values are sorted alphabetically. P15 pointed out, "*I think if VisAct can recommend some visualizations automatically, I would use it more frequently in the future.*"

7 Discussion and Future Waork

VisAct is an interactive visualization system designed to help average users to analyze their data, and the learning curve of our system is extremely low. The novelty of VisAct is achieved by the following two main characteristics: It guides users to create visualizations step by step through the wizard panel; It offers a high-level history tracker which records all the semantic actions created by users and support users in reviewing, resetting, revisiting these actions. These features of our system aim to offer an easy-to-use visualization system which facilitates better history management. In the evaluation session, we use two matrics (task completion steps and correct rate) to examine the efficiency and effectiveness of our system in the comparison of Polestar. We find that it takes fewer steps by users to complete tasks in VisAct and the correct rate is significantly higher than in Polestar.

The existing chart types in our system include line chart, bar chart, scatterplot, histogram, pie chart, area chart, scatterplot matrix, treemap, boxplot, DICON [5], and force directed graph. The chart type in our system is extensible as developers can create new chat types according to our development standards and integrate them into our system as visualization components. To design a new chart type such as storyline visualization [24], a developer first defines the attributes of the chart (e.g., size and margin of the chart, visualization channel, and chart properties), and the actions of the chart (e.g., data exploration actions, visual exploration actions, meta actions, and insight actions). Finally, the developer can upload the new visualization component to our system through the administrator account.

Acknowledgements Yang Shi is the corresponding author. This research was sponsored in part by the Fundamental Research Funds for the Central Universities in China and the National Natural Science Foundation of China under grant No.61802283, 61602306.

References

1. Webpack. https://webpack.js.org/, 2019.
2. C. Ahlberg. Spotfire: an information exploration environment. ACM SIGMOD Record, 25(4):25–29, 1996.
3. M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. IEEE transactions on visualization and computer graphics, 15(6):1121–1128, 2009.
4. M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. IEEE Transactions on Visualization and Computer Graphics, 17(12):2301–2309, 2011.
5. N. Cao, D. Gotz, J. Sun, and H. Qu. DICON: Interactive visual analysis of multidimensional clusters. IEEE Transactions on Visualization and Computer Graphics, 17(12):2581–2590, 2011.
6. Center for Machine Learning and Intelligent Systems, UCL. Wine quality data set. Retrived from https://archive.ics.uci.edu/ml/datasets/wine+quality. 2009.
7. Donald Knuth. dataset of co-occurrence of characters in les miserable. https://www-cs-faculty.stanford.edu/ knuth/sgb.html.
8. R. Eccles, T. Kapler, R. Harper, and W. Wright. Stories in geotime. Information Visualization, 7(1):3–17, 2008.
9. D. Gotz and M. X. Zhou. Characterizing users visual analytic activity for insight provenance. IEEE Symposium on Visual Analytics Science and Technology, pp. 123–130, 2008.
10. J. Heer and M. Bostock. Declarative language design for interactive visualization. IEEE Transactions on Visualization and Computer Graphics, 16(6):1149–1156, 2010.
11. J. Heer, J. D. Mackinlay, C. Stolte, and M. Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. IEEE Transactions on Visualization and Computer Graphics, 14, 2008.
12. J. Heer, F. B. Viegas, and M. Wattenberg. Voyagers and voyeurs: Supporting asynchronous collaborative visualization. Commun. ACM, 52(1):87– 97, 2009.
13. M. Kreuseler, T. Nocke, and H. Schumann. A history mechanism for visual data mining. In IEEE InfoVis, pp. 49–56, 2004.
14. D. Li, H. Mei, Y. Shen, S. Su, W. Zhang, J. Wang, M. Zu, and W. Chen. Echarts: A declarative framework for rapid construction of web-based visualization. Visual Informatics, 2:136–146, 2018.
15. 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. 123:1–123:13. ACM, 2018.
16. Microsoft. Power BI. https://powerbi.microsoft.com/, 2015.
17. Microsoft. Powerpoint add-ins. https://docs.microsoft.com/ en-us/office/dev/add-ins/powerpoint/powerpoint-add-ins, 2019.
18. D. Ren, T. Hllerer, and X. Yuan. ivisdesigner: Expressive interactive design of information visualizations. IEEE Transactions on Visualization and Computer Graphics, 20(12):2092–2101, 2014.
19. D. Ren, B. Lee, and M. Brehmer. Charticulator: Interactive construction of bespoke chart layouts. IEEE Transactions on Visualization and Computer Graphics, 25(1):789–799, Jan 2019.
20. A. Satyanarayan and J. Heer. Lyra: An interactive visualization design environment. Computer Graphics Forum, 33(3):351–360, 2014.
21. A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. IEEE Transactions on Visualization and Computer Graphics, 23(1):341–350, 2017.
22. A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. IEEE transactions on visualization and computer graphics, 22(1):659–668, 2016.
23. C. E. Scheidegger, H. T. Vo, P. J. Crossno, S. P. Callahan, L. Bavoil, J. Freire, and C. Silva. Vistrails : Enabling interactive multiple-view visualizations. 2005. VIS 05. IEEE., pp. 135–142, 2005.
24. Y. Shi, C. Bryan, S. Bhamidipati, Y. Zhao, Y. Zhang, and K. Ma. Meetingvis: Visual narratives to assist in recalling meeting context and content. IEEE Transactions on Visualization and Computer Graphics, 24(6):1918– 1929, 2018.
25. Tableau. Tableau software. http://www.tableau.com/, 2015.
26. F. B. Viegas, M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon. Manyeyes: a site for visualization at internet scale. IEEE Transactions on Visualization and Computer Graphics, 13(6):1121–1128, 2007.
27. C. Weaver. Building highly-coordinated visualizations in improvise. Proceedings of the IEEE Symposium on Information Visualization, pp. 159– 166, 2004.
28. H. Wickham. A layered grammar of graphics. Journal of Computational and Graphical Statistics, 19(1):3–28, 2010.
29. Wiki server of Cal Poly Computer Science Department Labs. Houses dataset. Retrived from https://wiki.csc.calpoly.edu/datasets/wiki/Houses/. 2015.
30. Wikipedia contributors. Asynchronous module definition. https://en. wikipedia.org/wiki/Asynchronous_module_definition, 2018.
31. Wikipedia contributors. List of countries and dependencies by population. https://en.wikipedia.org/wiki/List_of_countries_and_dependencies_by_population, 2019.
32. Wikipedia contributors. Responsive web design. https://en. wikipedia.org/wiki/Responsive_web_design, 2019.