Literature Review of Analytic Provenance

Phong H. Nguyen*

Middlesex University

Tuesday 22nd May, 2012

Abstract

Analytic provenance is the area of research that focuses on understanding a user's reasoning process through the study of their interactions with a visualisation. Studying user interactions allows not only understanding the user's reasoning process, but enhancing the user's analysis process and reusing the user's successful analysis strategies and methods as well. In this survey paper, we categorise the literature in analytic provenance by the methods of using the captured user interactions to facilitate the analytical reasoning. The recorded information can help to recall the reasoning process, to recover the reasoning process, or to construct the reasoning process. Moreover, other research examines semantic interactions and reuse of user interactions.

1 Introduction

"Visual analytics is the science of analytical reasoning facilitated by interactive visual interfaces" [42, p.4]. The key role of visual analytics is to support analysts to derive insights from the massive amounts of data. However, not only is the extracted knowledge important, but the analysis process that led to that knowledge and the rationale underlying that analysis are also of great significance. To discover these two important aspects, a potential solution is to examine the user interactions with the analysis tool. And "the area of research that focuses on understanding a user's reasoning process through the study of their interactions with a visualisation is called Analytic Provenance" [31, p.33].

In 1996, Shneiderman already noticed the importance of capturing user interactions in information visualisation by considering *history* is one of the seven tasks in his *Task by Data Type Taxonomy* [37]. According to Shneiderman, information visualisation systems need to support users to review previous actions and correct mistakes because the information exploration process is typically long and complex. Since then, there is more research on *exploration history* in visualisation and related fields. In *information visualisation*, graphical histories are used to enhance the data exploration by allowing users to review and revise their past actions [7, 36]. In *scientific visualisation*, a model and framework to capture and reuse the information within the visualisation process is introduced by Jankun-Kelly, Ma and Gertz [19]. Moreover, VisTrails is an open-source scientific workflow and provenance management system that provides support for simulations, data exploration and visualisation [3]. In *visual data mining*, a framework to integrate the history functionality into visual data mining software to support users in complex visual mining scenarios is described by Kreuseler, Nocke and Schuman [23]. In *visual analytics*, the study of user interactions needs to go beyond the chronicle of actions to be able to understand the analyst's analytical reasoning

*e-mail: p.nguyen@mdx.ac.uk

process. Gotz and Zhou [14] characterise users' visual analytic activity for capturing insight provenance instead of capturing low-level user interactions. Other notable research works on understanding and supporting a user's reasoning process through study the user interactions, or *analytic provenance*, are performed by Shrinivasan and van Wijk [38–40] and by Chang and his colleagues [8, 20, 26].

Since 1996, there is an increase of research on history and analytic provenance; however, until May 2011, the first workshop dedicated for analytic provenance was held in *CHI 2011* called *Analytic Provenance: Process + Interaction + Insight* [31]. The goal of that workshop is *to develop a research agenda for how to better study analytic provenance and utilise the results in assisting users in solving real world problems*. After that successful workshop, a panel in *VisWeek 2011* [18] was held to draw more attention from the research community and *call to action* for further research on analytic provenance.

Even though the original purpose of studying user interactions in analytic provenance is to understand the user's reasoning process, other benefits can also be gained from this study. So far, there has not been any review of analytic provenance research. The only exception is a design guideline for building a graphical history for visualisation of Heer et al. [16]. User interactions capturing is important; however, it is only the first step in supporting analytic provenance. In this survey paper, we categorise the literature in analytic provenance based on how the captured user interactions are used. Before discussing about analytic provenance, an overview of research on other provenance fields is described in Section 2. The first and most common use of captured user interactions is to recall the reasoning process during the analysis session (Section 3). Second, user interaction logs are analysed after the data exploration session to recover the analyst's reasoning process (Section 4). Third, captured user interactions can help users to manually construct the reasoning process (Section 5). Fourth, Section 6 introduces a method that only focuses on semantic interactions. Fifth, the last category is a variety of applications of reusing the captured user interactions (Section 7). In a technical view, Section 8 discusses about the theory and practice in implementing the user interactions capturing. Finally, the paper ends with the conclusion in Section 9.

2 Background

According to Oxford dictionary, *provenance* is defined as "the place of origin or earliest known history of something" [1]. Provenance plays an important role in many aspects of our daily lives. For example, in food industry, before buying a bottle of fruit juice, people are happy to know what its ingredients are, what its origin is, how fruits are collected, stored and processed, and how the product is maintained and delivered. Therefore, the provenance of a product much affects the purchasing decision of customers. Another example is in the context of art, the provenance information of a painting such as authorship, material, drawing time and the story behind the painting greatly decides the value of that painting.

In computer systems, according to Luc Moreau [29], "the provenance of a piece of data is the process that led to that piece of data". Most of research on provenance has been done in the *database*, *scientific workflow* and *semantic web* communities. In scientific workflow, e-science, the experiment needs to be performed through many complicated steps and rerun many times with parameters tuning. Therefore, the experiment process needs to be recorded so that the entire or partial process can be easily re-executed with modification and/or compared with other processes. As a result, provenance allows the reproducibility of scientific results. In database technology, the research is focused on the data itself; for example, improving the performance and security in storing and querying the provenance information

[1] http://oxforddictionaries.com/definition/provenance?q=provenance

of the data. Especially, in curated databases where the data is manually created and edited by various authors, it is essential to track different versions of data items and to be able to answer questions such as who changes the data, in which parts and what the rationale behind the changes is. Provenance research on database and scientific workflow is matured and can be reviewed in many good surveys as [6, 41]. Recently, World Wide Web Consortium has supported research on provenance in semantic web [29]. When people come across a webpage, how can they trust a piece of information there? Detailed research questions and road map for developing provenance on the web can be found in the final report of the W3C Incubator Group [2].

3 Recalling the Reasoning Process

The most common use of capturing user interactions with visual analytics tools is to recall the reasoning process. When a user experiences with the system, every his or her meaningful interaction is automatically captured. The definition of semantically meaningful interactions varies among applications and contexts. For example, scrolling the mouse may not be important in a data analysis software but is essential in a geographic visualisation because the zoom level and focus area are changed. Additionally, moving the mouse is normally not crucial; however, when a glyph in a visualisation is hovered and a tooltip is displayed to show more details which may lead to some critical finding, this action is of importance.

After capturing relevant user interactions, these pieces of information can be used in three different methods to help to recall the reasoning process. First, past actions are arranged chronologically to show the systematic actions of the user. Second, the evolution of the analysis process is illustrated in terms of another factor instead of time. Third, the overview of the analysis process is provided statistically.

3.1 The Chronicle of Actions

History View *How to arrange space for displaying the history?*

Typically, history is displayed separately besides the main data exploration. An additional view can be used to display only important visualisations like *bookmark view* in VisTrails [3] or *synopsis view* in Outpost [22]. Besides the aforementioned global history view and important history view, Outpost has a local history view for presenting history items related to the selected objects in the data view.

History can also be embedded into the exploration workspace [9, 27]. After each meaningful interaction, the whole visualisation is generated, added into the analysis space, and linked with its parent in the same manner as adding a history item into a separate history view. This integrated approach reduces users' effort when tracking and using history. However, since all past states are simultaneously displayed in the main analysis view, the method is only suitable for *small-scale visualisations*; for instance, scientific visualisation (Image Graphs [27]) or single chart visualisation (GraphTrail [9]). Figure 1 shows an overview of those two examples.

History Item Representation *How to represent a single state or action?*

The simplest method to describe states or actions is to use *text*. Because of limited space for displaying history items, texts needs to be shortened [2]. Moreover, to increase the readability, icons can be augmented to represent different types of actions [13] as in Figure 2.

[2] http://www.w3.org/2005/Incubator/prov/XGR-prov-20101214/

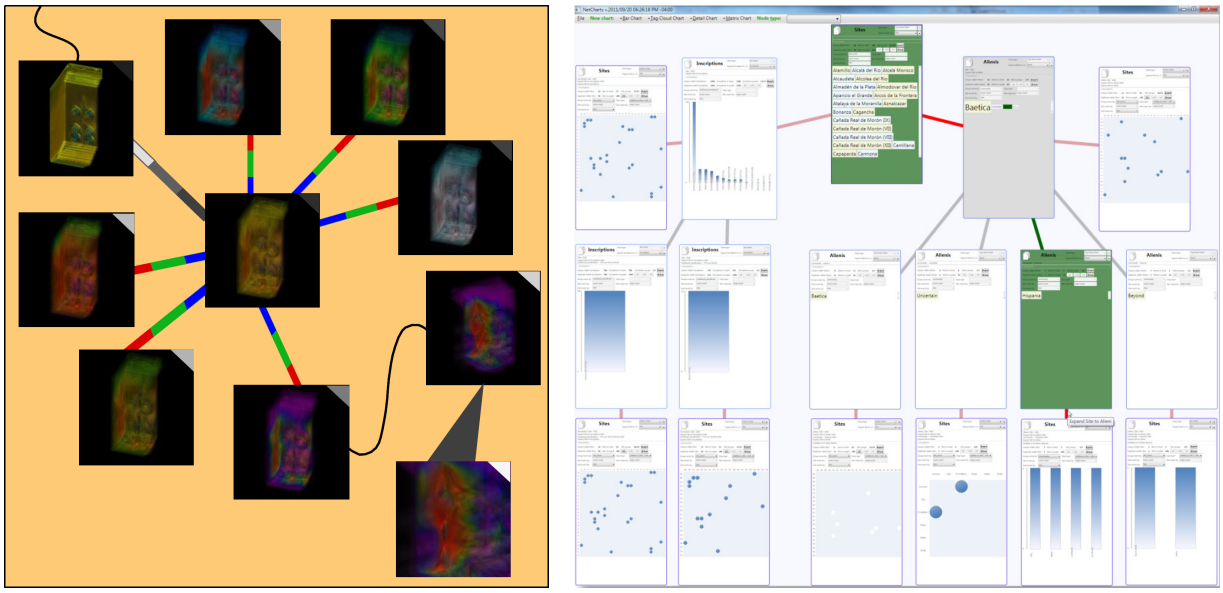


Image Graphs [27] GraphTrail [9]

Figure 1: Examples of integrated history views

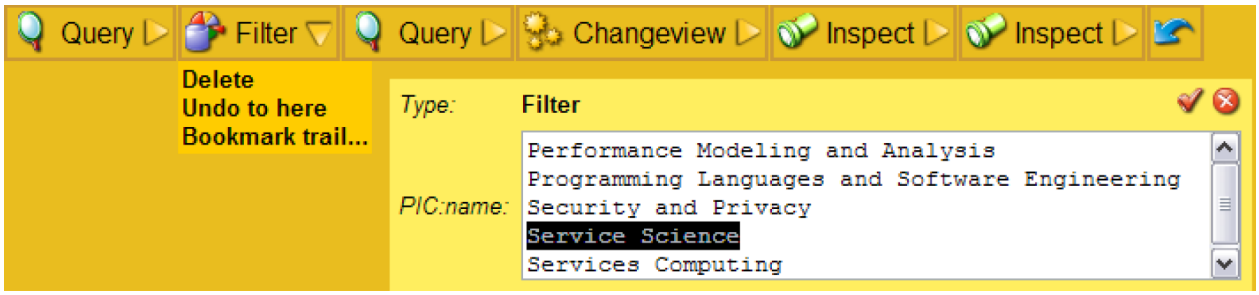


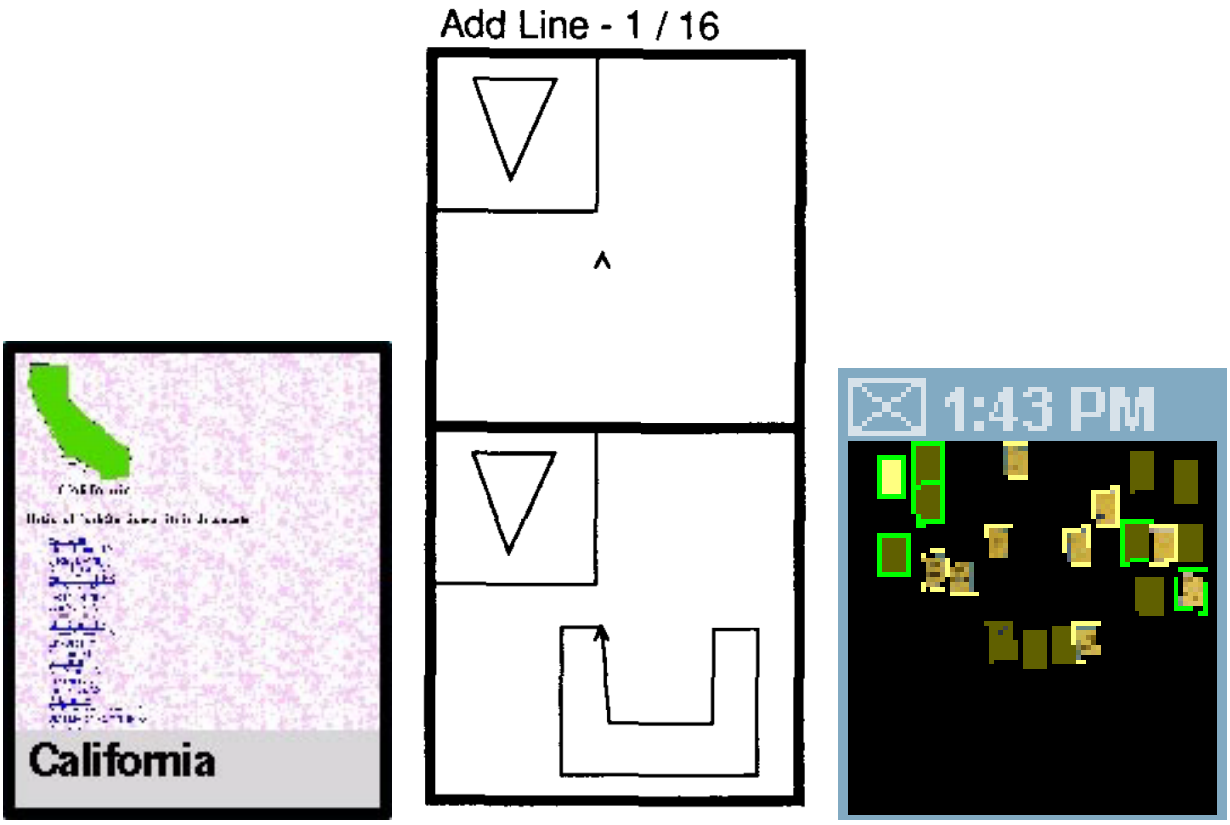
Figure 2: An example of using icons to represent different types of actions including Query, Filter, Inspect, Change view. The actions are layout linearly with comic strip metaphor [13].

Graphical representations of history items are more common because images help to recall past visualisations. Miniatures of visualisations are frequently used [2, 17, 22, 28]. Because the space for displaying a history item is substantially smaller than the space for the main analysis view, using an exact miniature screen capture causes difficulty in observing the details. In Chimera [24], a graphical editor, Kurlander and Feiner restrict the displayed objects and use a number of heuristics to find the region reflecting the changes and to easily recognise where that region is in the entire view. Similarly, Klemmer et al. [22] highlight the changes in thumbnails. Figure 3 show those variations of thumbnails usage.

History Layout *How to arrange the history items?*

The first and most basic method is to use a *vertical list* to display past user interactions as a history drop-down menu in typical computer applications or web browsers. The history list can be displayed in a separate window with supports for manipulating past actions as in Figure 4.

The second method is to use *comic strip metaphor* to represent a linear sequence of history items [13]. Kurlander et al. [24] use two parallel comic strips to depict two views, before and after the actions, to easily show the difference of those. Comic strips are also used in inline branching histories where only nodes of the active branch are displayed while other nodes are collapsed [22] as in Figure 5.



Whole screen miniature [17] Before/After thumbnails [24] Highlighting difference [22]

Figure 3: Examples of using thumbnails in presenting history item.

The third method is to use a *node-link diagram* to represent the nature of history tree. Besides the ordinary layout [3], branches of a tree can be arranged horizontally to save space [2, 17, 21, 38]. A tree can actually be a graph, which allows merging two nodes [9, 27]. An example of history graph is shown in Figure 1 and examples of trees are shown in Figure 6. When using full trees, additional techniques need to be employed to solve the space consuming problem such as only displaying nodes of the active branch [22], collapsing non-annotated (less important) nodes, providing multiple levels of detail nodes [2]. When the history grows, zoom-able and pan-able interface is also useful for managing large number of nodes [17]. Moreover, the importance of all past states are not equivalent, distortion techniques [25] can be used to emphasise on more relevant states such as the latest one. Perspective view in Meng et al.'s graphical editor [28] is an example of using distortion techniques to display the history tree as in Figure 7.

Two history items are often connected by an *edge* and the edge is visually augmented to represent the transition action. In GraphTrail [9], edges are colour coded to illustrate different reasons of new graph creation such as filtering, pivoting or cloning. Moreover, hovering on an edge pops out the textual information of the analysis trail leading from the beginning to the hovered edge and highlights the entire trail. Image Graphs [27] also uses six different shapes of edges to depict six different types of parameters change between two graphs.

History Navigation *How to navigate to the desired past state?*

The first method is to reverse the last state sequentially until the desired status is found. This undo/redo approach is common in most computer applications; whereas, in web browsers, it is known as back/forward mechanism.

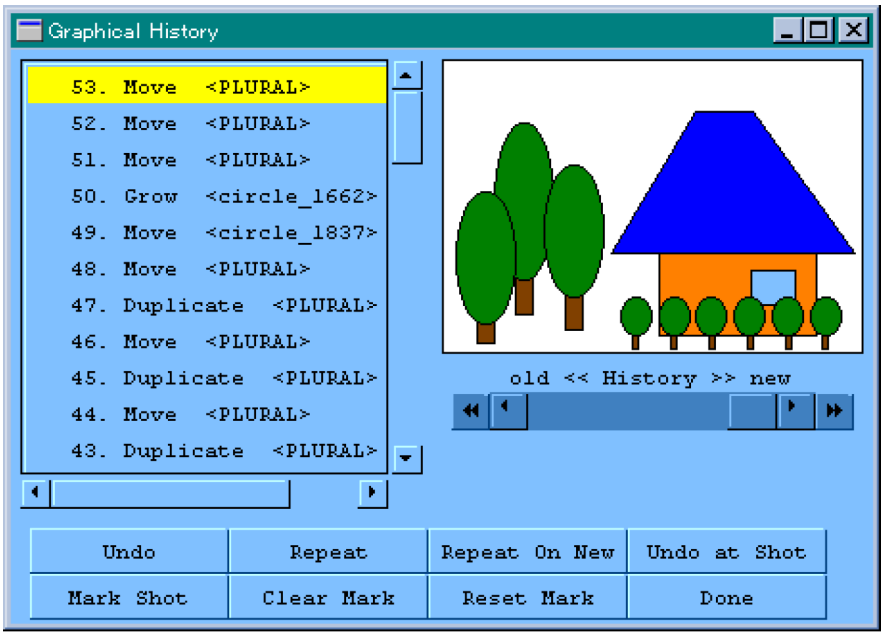


Figure 4: An example of using text-based list to represent past actions. [28].

Figure 5: An example of using comic strip to illustrate an inline branching tree [22].

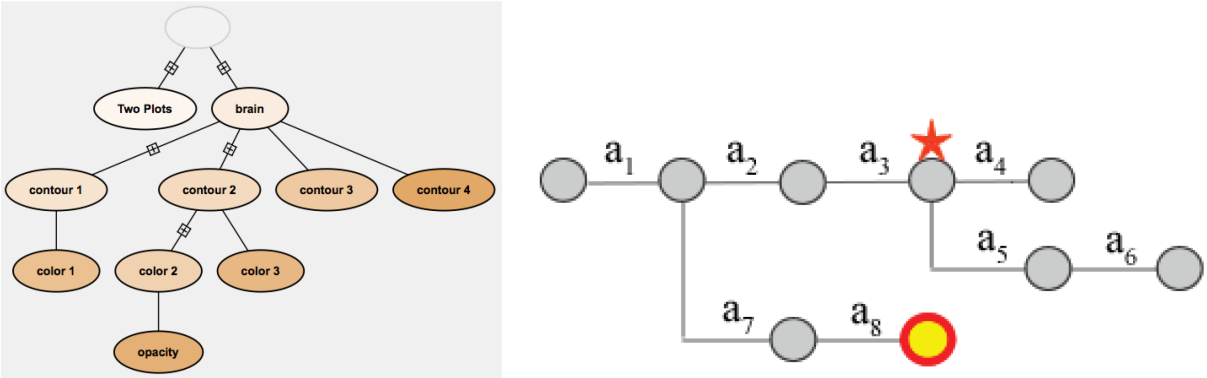
The second method is to manually scan through a catalogue of all past states and select the desired state. The catalogue of past states can be a text-based list, which is common in many web browsers, or a graphical representation, which quickly recalls the state's content [2,3,7,17,22,24,28].

The third method is to provide *search and filter* functionalities to support finding the desired state more efficiently when the number of past states is large. During the analysis process, users can take notes, tag keywords on visualisations [3,13,22] or bookmark interesting visualisations for later reference [3,13,22,27]. All those metadata are subject to search [39,43]. Moreover, VisTrails supports *query-by-example*, which allows finding related visualisations [35].

Past states can be filtered to display every the fixed number of actions or seconds systematically [22]. Another method to filter is based on metadata such as authors [22,39] or range of time [39]. A graphical editor tool developed by Meng et al. [28] provides context-based filtering. When an object in the editor is selected, the history displays only actions related to that object. This mechanism quickly helps the user to understand what has happened with the objects of interest.

3.2 Dependency Graph

Dependency graph is implemented in CzSaw [21], a text analytics tool, to show the flow of data during the analysis process. Figure 8 shows the dependency graph of the analysis session of a scenario from the novel *The Day of the Jackal*. First, the analyst searches for person *Montpellier* and the result is assigned to *entPerMont*. Then, all reports related to Montepellier are fetched to *repMont*. All persons mentioned in those reports are listed



Ordinary tree [43] Horizontal tree [38]

Figure 6: Examples of using trees in history layout.

for further investigation (*entPerRelMont*). The analyst notices on *Duggan* and continues investigate examine that person. That is the process which the analyst investigates the case; however, in the data perspective, after noticing Duggan, the analyst restarts the steps as he or she did with Montepellier. Therefore, in terms of data, Montepellier and Duggan are two independent branches in the data analysis tree.

3.3 Analysis Process Statistics

The overview of the analysis process can be revealed by showing some statistical values of application usages. In Aruvi system, [39], the key-aspects overview uses tag cloud metaphor to represent the most important items in four different aspects including most used visualisation tools, most investigated data aspects, most selected objects and most view objects as in Figure 9.

4 Recovering the Reasoning Process

When a user experiences a visual analytics tool, his/her plans and methods to analyse data could partially be reflected through his/her interactions with the application. Therefore, by analysing the system's interaction logs, the user's reasoning process could be revealed. To verify this hypothesis, Dou et al. [8] conduct a quantitative study to measure how much of a user's reasoning process can be recovered from only the captured user interactions.

Ten professional analysts are recruited to use WireVis [5], a multiple linked views visualisation of categorical, time varying wire transaction data, to analyse a realistic data set. A transaction data item consists of the sender and receiver's names, the transferred amount, the transaction time, and some particular keywords relating to the transaction. The four linked views of WireVis illustrate the relationships among accounts, keywords and time; thus support analysts detecting suspicious transactions. The analysis sessions are captured by a camera and the participants are asked to think aloud to present the reasons behind their actions. The participants are also interviewed to describe the high-level strategies they plan to detect fraud, the suspicious transactions they found and the methods leading them from the general plan to the detailed findings. Those strategies, methods and findings are recorded as the ground truth to measure the recovered information from the interaction logs.

While the participants experience with WireVis, their interactions are captured and the interaction logs are passed to the two interaction analysis tools [20] when the sessions finish. First, the Operation Analysis tool shows the user interactions with WireVis over time during

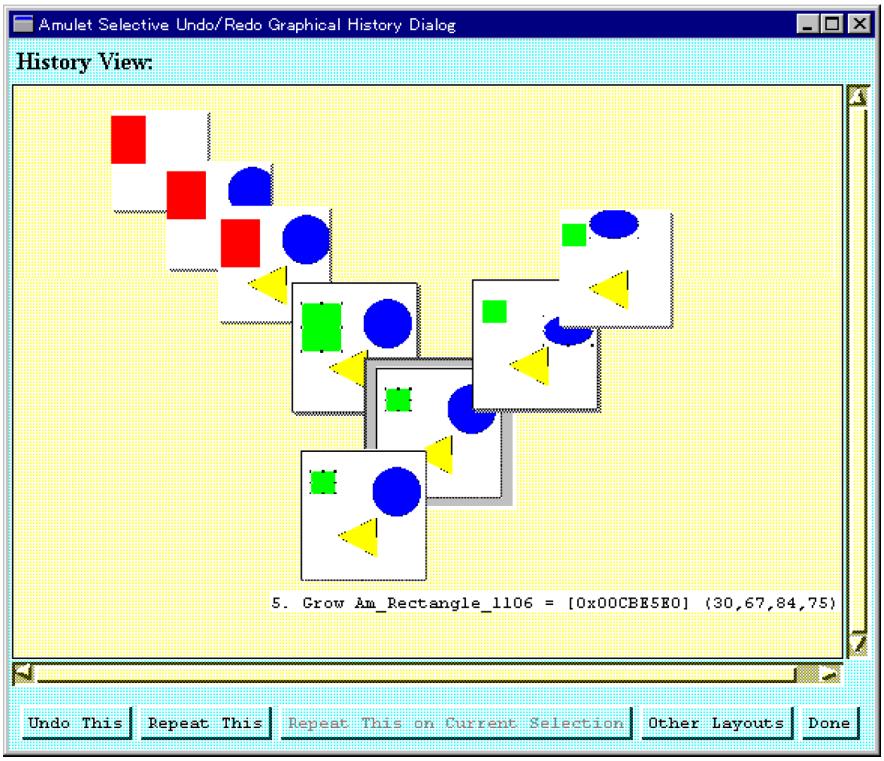


Figure 7: An example of using perspective view in history layout: further objects are smaller [28].

the analysis session. The tool can reveal the analyst's usage pattern; for example, in the first five minutes, the analyst uses the Heatmap view and focuses on some high frequency keywords, then he switches to the Strings and Beads view to observe the transaction pattern of some interested accounts found in the Heatmap view. Moreover, the tool also shows how "deep" the analyst interacts with the system (analyse the overview pattern or directly examine detail transactions) and how "wide" the analyst analyses the data (investigate the entire time span of the data or just focus on some suspicious transactions). Second, the Strategy Analysis tool shows the overall picture of the analysis process regardless of time instead of the systematic actions as in the Operation Analysis tool. The tool helps to discover the high-level strategy that the analyst employs to detect suspicious activities; for instance, the analyst focuses on some particular accounts or some keywords that typically should not exist in the same transaction.

Four people already familiar with WireVis use the two interaction analysis tools to discover the strategies, methods and findings from the interaction logs of those ten analysts. The outcomes are compared with the aforementioned ground truth and graded. The result shows that 79 per cent of the findings, 60 per cent of the methods and 60 per cent of the strategies are extracted from manually analysing only the interaction logs.

This post-analysis interaction logs approach is domain-specific because proprietary tools need to design to effectively discover some well-known strategies in detecting suspicious transactions. Even though reasoning processes are discovered, the interaction analyses are taken place after the data analyses and thus the recovering cannot support analysts much.

5 Constructing the Reasoning Process

During the exploration analysis, the user may encounter some interesting patterns or outliers, their causal relationships, and associate them with his or her thought. The user may also build a hypothesis and use those found artefacts to confirm or contradict this possibility.

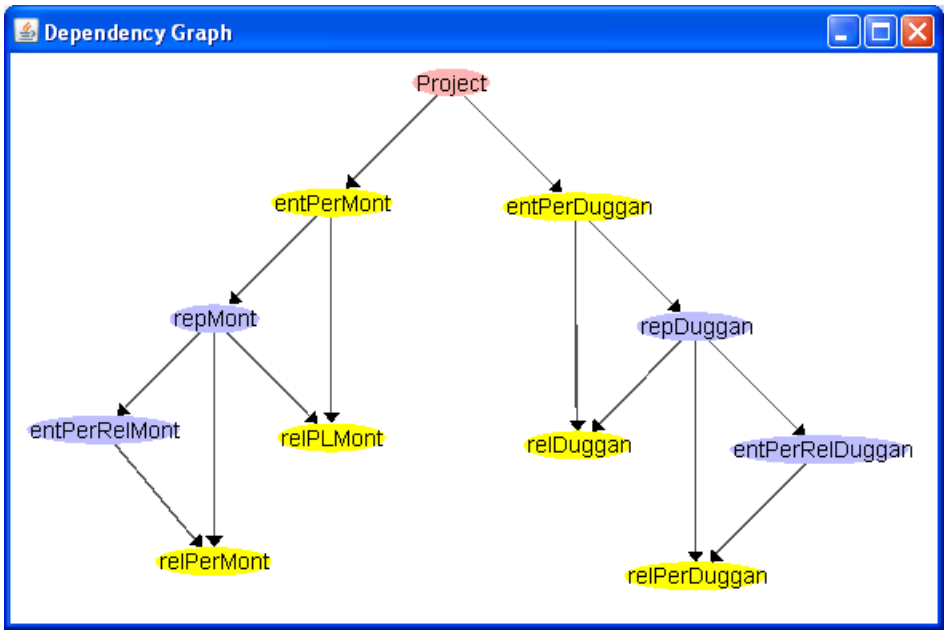


Figure 8: An example of Data-Flow Graph: Dependency Graph in [21]

Commonly, the user has to keep this reasoning process in his or her mind or document the idea in a paper. However, these two methods suffer from many disadvantages. First, when the analysis process is long, the user's cognitive load increases, and thus important findings may be lost. Second, taking notes in an external application requires many user' efforts and can cause the analysis less effective. Third, there exists no recall of visual data representation in those two environments. Finally, there is missing a management of a complex reasoning process, which is able to search or filter the artefacts, to combine the findings and to support verifying the hypotheses. One solution to those problems is to manually construct the reasoning process inside the exploration environment by using discovered knowledge and linking the knowledge with visualisations.

The knowledge view in Aruvi system [38], equipped by a graphical editor, allows users freely to create notes inside rectangles or ellipses, to connect related notes with arrows, or to link a note with a captured visualisation. Figure 10 shows an example of the reasoning process constructed in the knowledge view.

In Scalable Reasoning System [32], users can construct a powerful, well-supported reasoning diagram with different reasoning artefacts including evidence, causal relationship, assumption and hypothesis. First, users drag and drop an interesting visualisation to the reasoning space to convert it to a node in the reasoning diagram. The node shows small-scale visualisation and can be tagged as an evidence artefact. A node can be converted to an edge, which connects the two nodes as the casual relationship between them. Users can also create an annotation node to explain their thoughts and tag the node as an assumption artefact. Assumptions can be converted to hypotheses by associating evidence and casual artefacts. Moreover, by setting the confidence and uncertainty of the evidence as well as the degree of importance of the evidence, Scalable Reasoning System uses Dempster-Shafer [34] belief network to calculate the possibility of the hypothesis, and consequently help to confirm or contradict the hypothesis. Figure 11 shows the reasoning workspace of the system.

So far, all the approaches to construct the reasoning process during the exploration analysis are manual. Users need to select important visualisations, put these artefacts into the reasoning space and construct their casual relationships. Heuristics could be used to automatically or semi-automatically choose relevant visualisations to construct the reasoning process. A simple heuristic that may work is to pick all bookmarked or annotated visualisa-

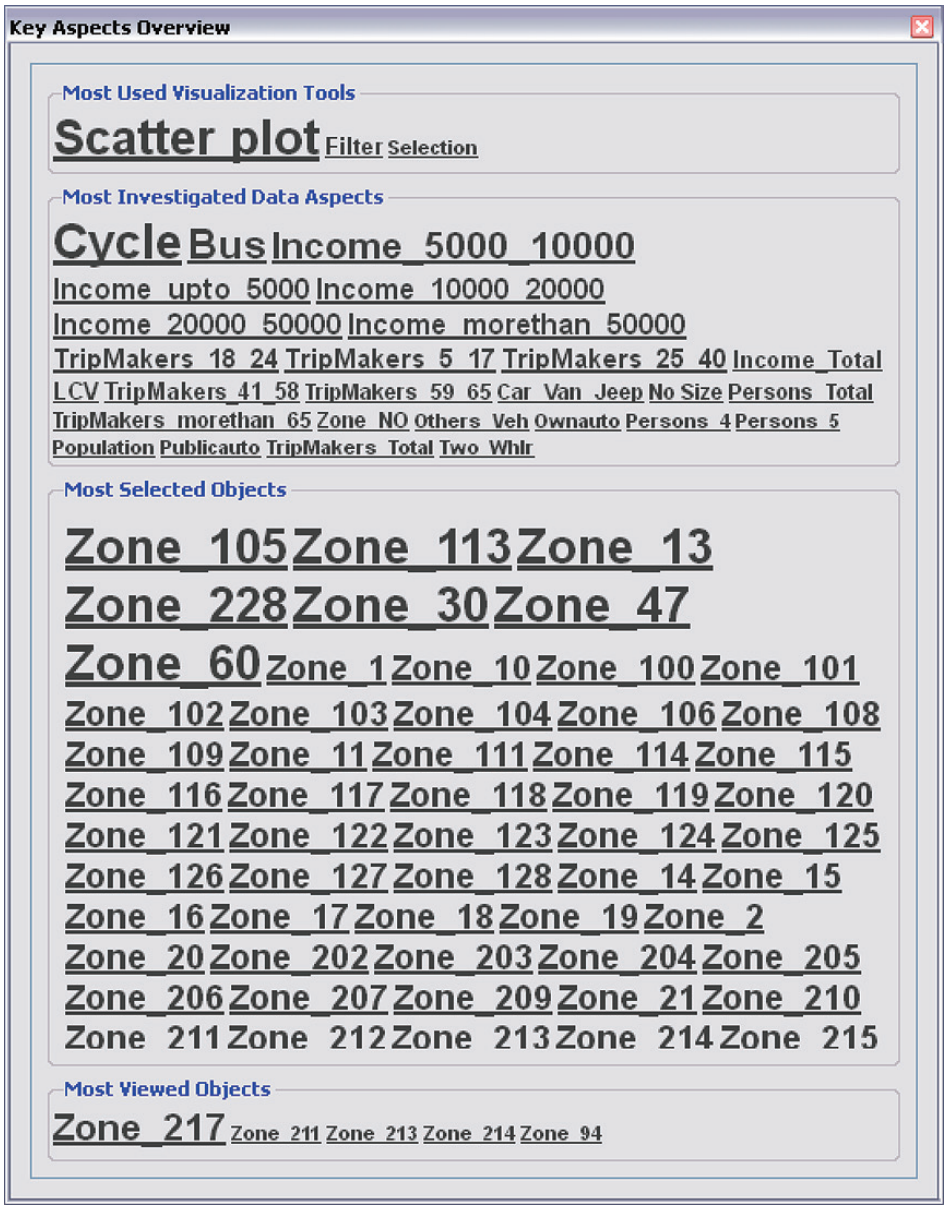


Figure 9: An example of analysis process overview: Key-aspects Overview in [39]

tions.

6 Focusing on Semantic Interactions

Instead of capturing all user interactions, another approach is to focus on some types of actions that have higher importance in some particular applications.

Data selection operation is specially managed in Aruvi system [40]. When a subset of data is selected, Shrinivasan and van Wijk argue that the user is defining a semantic zone containing these selected data items. Therefore, the data selection actions need to be captured and managed over time. A table representation called Select & Slice table is used to analyse those semantic zones. The horizontal axis of the table represents semantic zones and the vertical axis depicts the data subsets of the semantic zones. Data subsets can be subdivided into smaller chunks for analysing purpose. A cell of the table holds selected items corresponding to a sub-dataset of interest. Cells can display the number of items, the average value or some simple graphical representations to illustrate their sub-selections. To assist the analysis of semantic zones, a variety of capabilities is provided to combine zones

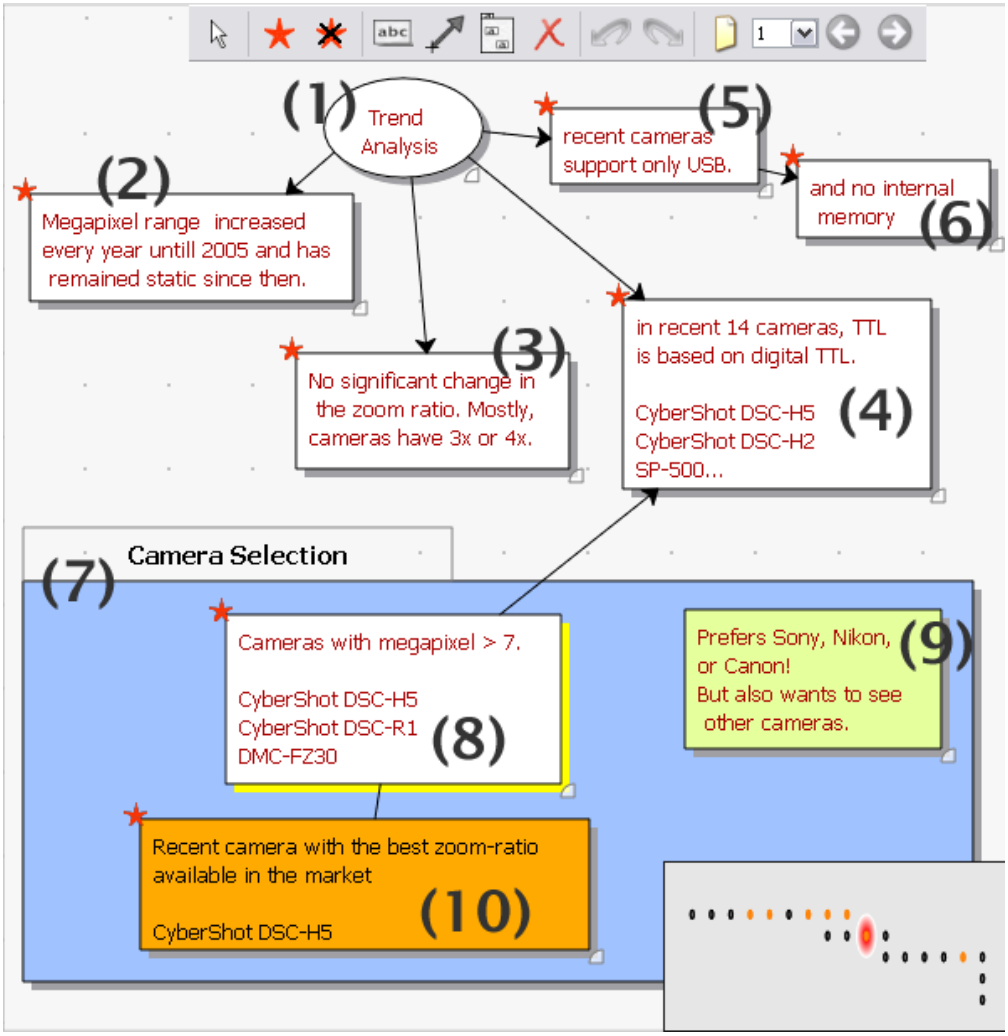


Figure 10: An example of manually constructing the reasoning process. The user analyses data camera dataset, during the analysis process, he or she builds the reasoning diagram using the graphical editor and linking notes with visualisations (notes with stars) [38].

using set operations, to view item distributions of one cell among other cells and to search the semantic zones by keywords. Figure 12 shows Select & Slice table and the distribution analysis of selected items.

In text visual analytics, typically, data is processed using some data mining algorithms or statistical models and then the results are visualised. When users want to customise the visualisation, they need to directly interact with the complicated algorithms and models by changing some parameters. Those parameters are not intuitive and very often, the expert analysts do not have much knowledge in text data mining or statistics and have difficulty in tweaking those settings. A novel mechanism implemented in ForceSPIRE by Endert et al. [11] allows users to modify visualisations by using semantic interactions. ForceSPIRE uses force-directed graph layout [12] for visualising documents. A node represents a keyword extracted from documents and an edge depicts the relationship between two nodes. In force-directed model, a node is associated with a mass and an edge is associated with a spring; and within text analytics context, mass corresponds to the number of entities in the document and spring corresponds to the number of shared entities between two documents and the importance values of those entities. Endert et al. suggest a list of semantic interactions and their associated analytic reasoning. For example, when a user highlights a word, he or she implies that this word is important. Therefore, ForceSPIRE increase the importance of that entity or create a new entity if it does not exist. The importance values of other

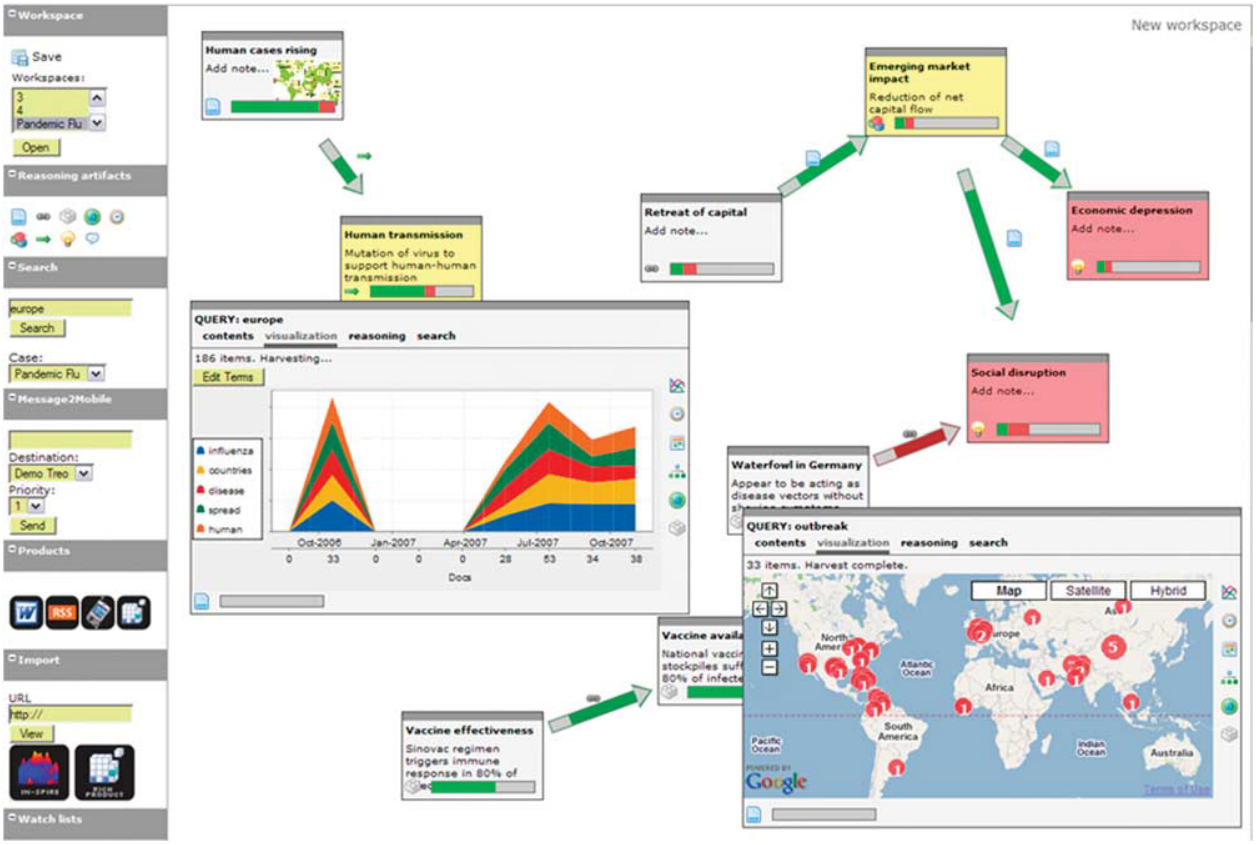


Figure 11: Another example of manually constructing the reasoning process. Reasoning Workspace in [32] supports constructing evidence, casual relationship, assumption and hypothesis (distinguish from different icons in the figure). The small bar at the bottom of each node represents the possibility of confidence, uncertainty and neutral of the artefact.

entities are subtracted equally to compensate for the increase. As a result, the graph layout changes, more important entities, set by semantic interactions such as text highlighting, term searching or document annotation rather than an "importance-entity-slider', are moved close to each other and probably an interesting pattern can be found.

7 Reusing User Interactions

7.1 Tutorial by Demonstration

Grabler et al. [15] present a demonstration-based system for automatically generating a compact systematic visual tutorial of photo manipulation based on a demonstration. When the tutor demonstrates the manipulation, the system records both interface state (what the tutor did in the interface; for example moving Saturation slider from value 0 to value 5) and application state (the resulting changes; for example, saturation parameter is changed from 0 to 5). The screenshot is also taken for any important changes (for example, in dragging slider value, only initial and final states are captured). The tooltip is also recorded for the meaning of the result of the action. Besides recording those states, an image labeller is used to detect features of interacting regions. It helps to generate meaningful description; for instance, instead of "selecting the ellipse region with coordination x, y", a more meaningful narrative is "selecting the right iris, the right pupil". The system is able to group multiple actions into one operation (tuning value by changing slider), group multiple operations into one-step. The system is also be able to generate macro to accomplish demonstrated task with different input images thanks to application state recording and features detection. Figure

Figure 12: Select & Slice table in [40]. (a) Analyse distribution of Japanese cars, (b) one-fourth of *cars having good acceleration* are from Japan and most of them have 3-6 cylinders (50/51) and (c) all of those cars weigh less than 3000 pounds.

13 shows an example of a tutorial of changing the eye colour in the paper.

7.2 History Modification

When reviewing the history, users can insert missing actions, remove undesired actions and reuse past actions [24, 28, 30]. Amulet system [30] can support undo parts of commands. For example, first, ten graphical objects are selected and deleted from a graphical editor. Then, the user recognises that one object is accidentally deleted. The mistaken deleted object can be recovered by modifying the delete command to deselect that object. Another graphical editor [28] supports content-based undo. For instance, using that graphical editor to draw a *human face*, it is possible to select a *nose* to revert to a past state while other details keep unchanged.

The past actions can also be modified directly by changing the command statements [21], the command parameters [13] and the changing effects can be propagated along the history trail [27].

Considering the visualisation as the result of processing a number of understandable parameters, it is feasible to compare the visualisation by measuring those values. In scientific visualisation, the visualisation exploration process can be compared and merged [3], a new visualisation can be generated from previous visualisations by using various *set* operations including *intersection*, *union* and *difference*. In GraphGrail [9], a network exploration tool, Dunne et al. map the active dataset of the visualisation with a *SQL statement*; thus, this mapping allows merging two visualisations by performing a SQL *union* statement.

7.3 History Exporting

Because the history of the analysis process is important for understanding the analysis results, it is essential to support exporting and sharing the history. Klemmer et al. [22] provides a print version of annotated thumbnails as a report. VisTrails [43] supports including a visualisation process in a paper through Latex format.

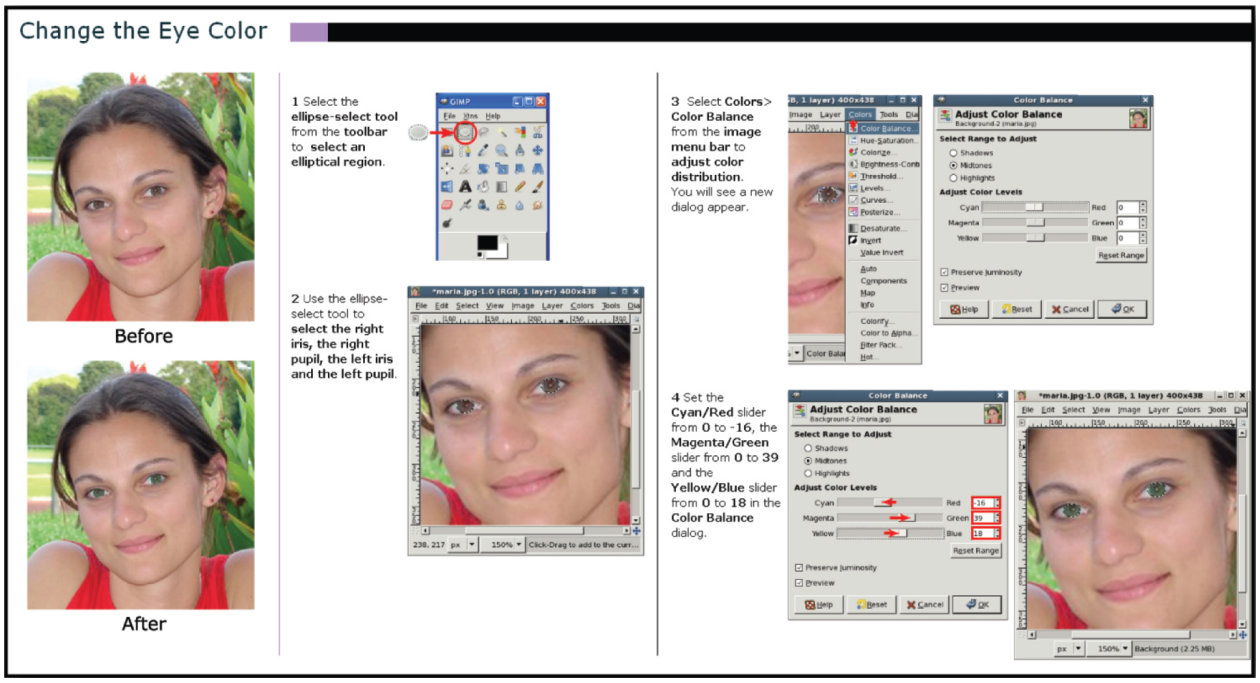


Figure 13: An example of generating a tutorial based on demonstration: Changing eye colour tutorial [15].

An explicit view of bookmarked visualisations provides the overview of the analysis process [3, 22]. An animation constructed from those key visualisations also helps to understand the analysis process visually [27].

For analysis processes, it is essential to maintain the current analysis process and restore to use later. Therefore, history information is also need to be able to serialised [2, 3, 21].

8 Capturing User Interactions - Theory and Practice

In this section, we review the internal history representation and implementation in computer applications.

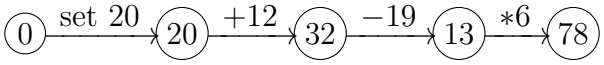
8.1 History Model

In this section, we use the term *state* to refer to all internal data and external interface variable values constructing the application state, i.e., all the values that reflect the application status. All the terms *action*, *operation* and *command* are used interchangeably and refer to an interaction that a user experiences with the system. The current application state is defined as the result of a sequence of actions performed since the application started. All executed commands are stored in an internal data structure which typically is a list called *history list*. For example, assume that the initial application state is S_0, if the user performs five actions in the following order A, B, C, D and E, then the history list will be

$$A \quad B \quad C \quad D \quad E.$$

Another method to interpret the history is to use *graph metaphor*. In a graph, *nodes* represent states and *edges* depict actions that transform one state into another state. Considering a simple calculator application with basic arithmetic operations, a node (state) will be the calculation result and an edge (action) will be the arithmetic operation. For a sequence of commands such as, 20 adds 12, then subtracts 19, and then multiplies by 6, the

graph representation of these operations is as follows:



History models can be divided into two groups. In *linear undo models*, only the last action can be undone; whereas, in *non-linear undo models*, arbitrary action can be undone.

8.1.1 Linear undo

Single-step undo Single-step is the simplest undo model and is commonly used in primitive editors such as *vi* and *notepad*. In this model, only the latest command is able to be undone. However, the model can support redo function that is typically considered as undo of the last undo, or flip-undo. It implies that the state of the system can flip back and forth between the current state and the last state.

For example, assume that the history list contains

$$A \quad B \quad C \quad D \quad E.$$

Performing undo once reverses command E and the history list becomes

$$A \quad B \quad C \quad D.$$

However, preceding command D cannot be reversed. Performing another undo re-executes command E and the history list returns to the original state

$$A \quad B \quad C \quad D \quad E.$$

The crucial limit of the single-step undo model is that only the last action can be undone.

Linear undo In the linear undo model, the last command in the history list is to be undone. When that last command is undone, it is removed from the history list and appended to the *redo list*. It is able to continue undoing preceding commands until the history list is empty, i.e., all the undone commands are passed to the redo list. Practically, the number of allowed undoing commands can be restricted to save the memory. The model can also support redo function. When a redo is performed, the last command in the redo list is removed and added back to the history list.

When a new command is issued, it is possible to keep the redo list or to empty the redo list. In a *restricted version of linear undo*, it is not allowed to redo an action from the state other than the original state that the action has happened; thus erasing the redo list when a new action is performed is required. For example, in a graphical editor, the user first draws a circle C and then sets the colour of circle C to blue. Next, the user undoes the setting colour command and deletes circle C. The problem arises when the user issues a redo command. If the redo list is not empty, the command to colour circle C should execute. However, because circle C does not exist anymore, that redo command will lead to an error.

One approach to save the redo list is to create a new branch to store the new command and subsequently, a *history tree* is formed. When performing a redo action, if there is more than one branch in the tree, another parameter needs to be provided to choose which node to execute.

In practice, the linear undo model can be implemented using a single doubly linked list and a pointer aims at the last done command which is also subject to be undone. The pointer works as a sentinel between the history list and the redo list. When an undo command is performed, the sentinel shifts back a step, i.e., exclude that node from the history list and include it into the redo list. The redo command behaves in the exactly opposite manner.

For instance, assume that the history doubly linked list is

$$A \quad B \quad C \quad \underset{\uparrow}{D},$$

whose sentinel (the red arrow) is pointing at the last command, D. When a new command E is performed, it is appended to D and the sentinel moves to the recently added command E

$$A \quad B \quad C \quad D \quad \underset{\uparrow}{E}.$$

Undoing a command once shifts the pointer to D and performing another undo causes the pointer to move to C as follows:

$$A \quad B \quad \underset{\uparrow}{C} \quad D \quad E.$$

Performing a new command F in non-restricted linear undo version appends F to C while maintains D and E. If a redo command is issued, the pointer will advance from F to D and the doubly linked list will become

$$A \quad B \quad C \quad F \quad \underset{\uparrow}{D} \quad E.$$

The essential limit of the linear undo model is that in some case, it is impossible to revert to the state that has happened before. As in the example above, once executing F, it is unable to revert to the original state, $ABCDE$. However, a new command (F) can be inserted in-between past commands in the history list.

History undo In the history undo model, the undo command is a primitive command, not a meta-command as in the linear undo model. The last command in the history list is also to be undone. When the last command is undone, the inverse command of that command is appended to the end of the history list. There is no explicit redo command in this model; redo is achieved by undo of undo command.

The same example as in the linear undo model is used for comparing these two models. Assume that the history list is

$$A \quad B \quad C \quad D \quad E.$$

Performing an undo command adds $\overline{E}$, the inverse command of E, to the end of the history list as follows:

$$A \quad B \quad C \quad D \quad E \quad \overline{E}.$$

Similarly, undoing another command causes the history list to become

$$A \quad B \quad C \quad D \quad E \quad \overline{E} \quad \overline{D}.$$

When a new command F is performed, it is appended to the end of the history list

$$A \quad B \quad C \quad D \quad E \quad \overline{E} \quad \overline{D} \quad F.$$

The active command list $ABCF$ and if the user wants to insert F between C and D as in the linear undo model, the user needs to perform D manually. Nevertheless, performing undo command three times allows returning to the original state, $ABCDE$

$$A \quad B \quad C \quad D \quad E \quad \overline{E} \quad \overline{D} \quad F \quad \overline{F} \quad \overline{\overline{D}} \quad \overline{\overline{E}}.$$

Opposite of the linear undo model, the history undo model allows returning to any happened state; however, new command cannot be inserted into past commands.

8.1.2 Non-linear undo

Non-linear undo models overcome the *change of heart* problem of the linear undo model and the *command insertion* problem of the history undo model. Moreover, non-linear undo models are able to undo and redo any action in the past, i.e., rearrange the entire history.

US&R Similar to the history tree model, the US&R model [44] provides explicit undo and redo commands and stores past commands in a tree. Moreover, another meta-command, *skip*, is introduced to skipping commands along the history branch when performing redo commands. Skip command allows choosing the desired command to redo; therefore, it is possible to restore to any past state.

The US&R model also suffers from the ambiguous problem when performing a redo command in case of multiple branches. Vitter [44] solves this issue by using additional interface to ask users to select the desire action to redo.

An example of undoing an arbitrarily past action will be used to demonstrate the strength of the US&R model. Assume that the history list is

$$A \quad B \quad C \quad D \quad E.$$

The user recognises that command C is unnecessary and wants to remove it. First, the user performs three undo commands and the history list becomes

$$A \quad B.$$

Then, the user issues a skip command to ignore command C which is surrounded by a blue box as follows:

$$A \quad B \quad \boxed{C}$$

Finally, the user redoes twice to execute two commands D and E to complete

$$A \quad B \quad \boxed{C} \quad D \quad E.$$

Triadic model Similar to the linear undo model, the Triadic model [45] maintains a history list and a redo list. When an undo command is issued, the last command in the history list is removed and appended to the end of the redo list; and that command goes back to the history list when a redo command is executed. Besides, the Triadic model provides a *rotate* command to spin the redo list one item. As a result, the last command becomes the first command in the redo list and the command before the last command turns to the last one. Eventually, it is able to select any command to redo and allow arranging past commands in the history list arbitrarily.

Once more, the example in the US&R model is used to illustrate the Triadic model. Assume that the history list is

$$A \quad B \quad C \quad D \quad E,$$

and the redo list is empty. Performing three undo actions makes the history list become

$$A \quad B,$$

and the redo list become

$$E \quad D \quad C.$$

Executing a rotate command changes the redo list to

$$C \quad E \quad D.$$

Finally, performing two redo commands to complete

$$A \quad B \quad D \quad E.$$

Selective undo The US&R model and Triadic model allow undoing arbitrary command, or *selective undo*; however, it is not always reasonable to do that. For example, in a graphical editor, it is not feasible to undo the object creation command but keep all other commands referring to that object. Therefore, it is necessary to check if it is possible to undo an arbitrary command or not. [4] and [33] are two selective undo models that perform logical checking before actually reverse a command.

There are two variations of selective undo. *Script-defined selective undo* reverses an action with the same result as rerunning all the actions but skipping the undone action while *direct selective undo* reverses the effect of the action itself. Choosing an appropriate variation is the trade-off between propagating the modification along the history path and changing the actual effect of the action. For example, in a graphical editor, the user draws a human face and replicates the result ten times. Then, he or she recognises that there is a redundant stroke on the face and undoes the action creating that stroke. If the script-defined variation is used, all ten faces are updated while if the direct variation is used, only the original face is corrected. Considering another example, a simple calculator application, the user enters *10* and performs many arithmetic operations; for instance, adding 20, then multiplying 30, then dividing by 4, then subtracting by 8. Then, the user wants to reverse the calculation result to the original number, *10*. If the direct selective undo is used, the user just needs to reverse the first calculation. Whereas, with the script-defined selective undo, the user has to undo the entire calculation steps to achieve that result.

Local and global undo The history can have one global history and many local histories for each *component*. In collaborative applications, it is practical that the user can undo their own action instead of global action irrespective of authorship. In multi-view visualisations, it is also useful if actions can be undone in terms of each visualisation view. Another application is in graphical editors where typically, the user wants to undo the changes of each object separately. More details about local and global undo models can be referred in [10] and [33].

8.2 History Implementation

The current state of the visualisation is the result of a sequence of actions performed since the beginning of the analysis. To recover a past state, either the state or all actions led to that state can be saved. Typically, each action is wrapped by a command object which implements *do* and *undo* methods to execute and reverse the action respectively [30]. When a command is issued, the corresponding command object is added into the history list to keep track. The implementation needs to ensure that it is possible to replay any past action in the history list by invoking its *do* method.

According to Archer et al. [1], there are four strategies to implement undo actions.

The complete rerun strategy Suppose that the initial state (S_0)is maintained, one approach to recover the state is to first, reset the current state to S_0; and then, completely rerun the entire sequence of actions excluding the latest one by invoking their *do* methods. The main cost of this method is the time to rerun all those commands from the beginning.

Full checkpoint strategy During the interaction process, the system periodically (after a fixed number of actions or a fixed amount of time) saves all the application statuses that are sufficient to recover the state from those statuses. When the desired state needs to be undone, first the closest checkpoint to that action is used to recover the backup state, and then rerun all actions from the restored state to the desired state. The extreme version of this approach is to save the application state after each interaction; thus, there is no need to

run any command after recovering the state. The main cost of this method is the memory to store the application states.

Inverse command strategy In this method, each command is associated with an inverse command (*undo* method), which reverses the effect of that command. Reverting to the desired state is simply backward executing inverse commands from the current state to that state. This strategy is not either time consuming as the complete rerun strategy or memory consuming as the full checkpoint strategy. However, some inverse commands are difficult or even impossible without storing more data. For example, undoing the image down-sampling command requires original image data.

Partial checkpoint strategy When a command is executed, typically it affects only some of the application statuses. Therefore, to be able to restore the state, only changed data needs to be stored. Similar to the inverse command strategy, the desired state is recovered by backward reversing commands from the current state to that state. However, partial checkpoint strategy reverses a command by loading the data to restore the command effect instead of executing the inverse command.

Choosing an undo strategy to implement depends on the nature of the application. Complete rerun strategy is not appropriate for computationally intensive applications while full checkpoint strategy is not suitable for high data producing applications. Moreover, if memory ratio of a full checkpoint to a partial checkpoint is high, partial checkpoint strategy will be much more efficient than full checkpoint strategy.

9 Conclusion

In this paper, we introduce a categorisation of the literature in analytic provenance based on the purpose of using captured user interactions with a visualisation. We hope the review can provide an overview as well as the state of the art of research on analytic provenance; thus, researchers will have more opportunities to overcome current limitation and to accelerate the research on all aspects of analytic provenance including capturing, storing and reusing user interactions.

References

[1] J. E. Archer, R. Conway, and F. B. Schneider. User Recovery and Reversal in Interactive Systems. *ACM Transactions on Programming Languages and Systems*, 6(1):1–19, Jan. 1984.

[2] E. Z. Ayers and J. T. Stasko. Using Graphic History in Browsing the World Wide Web. Technical report, 1995.

[3] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo. VisTrails: Enabling Interactive Multiple-View Visualizations. In *IEEE Symposium on Information Visualization*, pages 135–142. IEEE, 2005.

[4] T. Berlage. A Selective Undo Mechanism for Graphical User Interfaces Based On Command Objects. *ACM Transactions on Computer-Human Interaction*, 1(3):269–294, 1994.

[5] R. Chang, M. Ghoniem, R. Kosara, W. Ribarsky, J. Yang, E. Suma, C. Ziemkiewicz, D. Kern, and A. Sudjianto. WireVis: Visualization of Categorical, Time-Varying Data From Financial Transactions. In *IEEE Symposium on Visual Analytics Science and Technology*, pages 155–162. IEEE, Oct. 2007.

[6] J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases*, 1(4):379–474, Apr. 2007.

[7] M. Derthick and S. F. Roth. Enhancing data exploration with a branching history of user operations. *Knowledge-Based Systems*, 14(1-2):65–74, 2001.

[8] W. Dou, D. H. Jeong, F. Stukes, W. Ribarsky, H. R. Lipford, and R. Chang. Recovering Reasoning Processes from User Interactions. *IEEE Computer Graphics and Applications*, 29(3):52–61, May 2009.

[9] C. Dunne, N. H. Riche, B. Lee, R. A. Metoyer, and G. G. Robertson. GraphTrail: Analyzing Large Multivariate and Heterogeneous Networks while Supporting Exploration History. In *ACM Conference on Human Factors in Computing Systems*, pages 1663–1672, 2012.

[10] W. K. Edwards, T. Igarashi, A. LaMarca, and E. D. Mynatt. A temporal model for multi-level undo and redo. In *ACM Symposium on User Interface Software and Technology*, pages 31–40, New York, New York, USA, Nov. 2000. ACM Press.

[11] A. Endert, P. Fiaux, and C. North. Semantic Interaction for Visual Text Analytics. In *ACM Conference on Human Factors in Computing Systems*, pages 473–482, 2012.

[12] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, Nov. 1991.

[13] D. Gotz, Z. When, J. Lu, P. Kissa, N. Cao, W. H. Qian, S. X. Liu, and M. X. Zhou. HARVEST: An Intelligent Visual Analytic Tool for the Masses. In *International Workshop on Intelligent Visual Interfaces for Text Analysis*, pages 1–4, New York, New York, USA, Feb. 2010. ACM Press.

[14] D. Gotz and M. X. Zhou. Characterizing users visual analytic activity for insight provenance. *Information Visualization*, 8(1):42–55, Jan. 2009.

[15] F. Grabler, M. Agrawala, W. Li, M. Dontcheva, and T. Igarashi. Generating photo manipulation tutorials by demonstration. *ACM Transactions on Graphics*, 28(3):1, July 2009.

[16] 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*, 14(6):1189–96, 2008.

[17] R. R. Hightower, L. T. Ring, J. I. Helfman, B. B. Bederson, and J. D. Hollan. Graphical Multiscale Web Histories: A Study of PadPrints. In *ACM Symposium on User Interface Software and Technology*, pages 121–122, New York, New York, USA, Nov. 1998. ACM Press.

[18] T. J. Jankun-Kelly, R. Chang, C. T. Silva, C. Weaver, S. Parker, and L. McNamara. Process + Interaction + Insight : The Need for Analytic Provenance. Technical report, VisWeek 2011 Panel, 2011.

[19] T. J. Jankun-Kelly, K.-L. Ma, and M. Gertz. A model and framework for visualization exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):357–369, 2007.

[20] D. H. Jeong, W. Dou, F. Tukes, W. Ribarsky, H. R. Lipford, and R. Chang. Evaluating the relationship between user interaction and financial visual analysis. In *IEEE Symposium on Visual Analytics Science and Technology*, pages 83–90, 2008.

[21] N. Kadivar, V. Chen, D. Dunsmuir, E. Lee, C. Qian, J. Dill, C. Shaw, and R. Woodbury. Capturing and supporting the analysis process. In *IEEE Symposium on Visual Analytics Science And Technology*, pages 131–138. IEEE, 2009.

[22] S. R. Klemmer, M. Thomsen, E. Phelps-Goodman, R. Lee, and J. A. Landay. Where do web sites come from? Capturing and Interacting with Design History. In *ACM Conference on Human Factors in Computing Systems*, pages 1–8, New York, New York, USA, Apr. 2002. ACM Press.

[23] M. Kreuseler, T. Nocke, and H. Schumann. A History Mechanism for Visual Data Mining. In *IEEE Symposium on Information Visualization*, pages 49–56. IEEE, 2004.

[24] D. Kurlander and S. Feiner. Editable graphical histories. In *IEEE Workshop on Visual Languages*, pages 127–134. IEEE Comput. Soc. Press, 1988.

[25] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1(2):126–160, June 1994.

[26] H. R. Lipford, F. Stukes, W. Dou, M. E. Hawkins, and R. Chang. Helping users recall their reasoning process. In *IEEE Symposium on Visual Analytics Science And Technology*, pages 187–194. IEEE, Oct. 2010.

[27] K.-L. Ma. Image graphs - a novel approach to visual data exploration. In *IEEE Conference on Visualization*, pages 81–88, Oct. 1999.

[28] C. Meng, M. Yasue, A. Imamiya, and M. Xiaoyang. Visualizing histories for selective undo and redo. In *Asian Pacific Computer and Human Interaction*, pages 459–464. IEEE Comput. Soc, 1998.

[29] L. Moreau. The Foundations for Provenance on the Web. *Foundations and Trends in Web Science*, 2(2-3):99–241, Nov. 2010.

[30] B. A. Myers and D. S. Kosbie. Reusable Hierarchical Command Objects. In *ACM Conference on Human Factors in Computing Systems*, pages 260–267, 1996.

[31] C. North, R. Chang, A. Endert, W. Dou, R. May, B. Pike, and G. Fink. Analytic provenance: process+interaction+insight. In *ACM Transactions on Computer-Human Interaction*, pages 33–36. ACM, May 2011.

[32] W. Pike, J. Bruce, B. Baddeley, D. Best, L. Franklin, R. May, D. Rice, R. Riensche, and K. Younkin. The Scalable Reasoning System: Lightweight visualization for distributed analytics. *Information Visualization*, 8(1):71–84, 2009.

[33] A. Prakash and M. J. Knister. A framework for undoing actions in collaborative systems. *ACM Transactions on Computer-Human Interaction*, 1(4):295–330, Dec. 1994.

[34] A. Sanfilippo, B. Baddeley, C. Posse, and P. Whitney. A Layered Dempster-Shafer Approach to Scenario Construction and Analysis. In *IEEE Intelligence and Security Informatics*, pages 95–102. IEEE, May 2007.

[35] C. Scheidegger, H. Vo, D. Koop, J. Freire, and C. Silva. Querying and creating visualizations by analogy. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1560–1567, 2007.

[36] F. M. Shipman and H. Hsieh. Navigable History: A Readers View of Writers Time. *New Review of Hypermedia and Multimedia*, 6(1):147–167, 2000.

[37] B. Shneiderman. The Eyes Have It : A Task by Data Type Taxonomy for Information Visualizations. In *IEEE Symposium on Visual Languages*, pages 336–343, 1996.

[38] Y. B. Shrinivasan and J. Van Wijk. Supporting the analytical reasoning process in information visualization. In *ACM Conference on Human Factors in Computing Systems*, pages 1237–1246, New York, New York, USA, Apr. 2008. ACM Press.

[39] Y. B. Shrinivasan and J. Van Wijk. Supporting Exploration Awareness in Information Visualization. *IEEE Computer Graphics and Applications*, 29(5):34–43, Sept. 2009.

[40] Y. B. Shrinivasan and J. Van Wijk. Supporting Exploratory Analysis with the Select & Slice Table. *Computer Graphics Forum*, 29(3):803–812, Aug. 2010.

[41] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *ACM SIGMOD Record*, 34(3):31–36, Sept. 2005.

[42] J. J. Thomas and K. A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Center, 2005.

[43] University of Utah. VisTrails Documentation Release 2.0.0. Technical report, 2012.

[44] J. Vitter. US&R: A New Framework for Redoing. *IEEE Software*, 1(4):39–52, Oct. 1984.

[45] Y. Yang. Undo support models. *International Journal of Man-Machine Studies*, 28(5):457–481, May 1988.