Clustering Provenance

Facilitating provenance exploration through data abstraction

Linus Karsai
University of Sydney
Australia 2006
lkar7536@uni.sydney.edu.au

Alan Fekete
University of Sydney
Australia 2006
alan.fekete@sydney.edu.au

Judy Kay
University of Sydney
Australia 2006
judy.kay@sydney.edu.au

Paolo Missier
Newcastle University
NE1 7RU, United Kingdom
paolo.missier@ncl.ac.uk

ABSTRACT

As digital objects become increasingly important in people's lives, people may need to understand the *provenance*, or lineage and history, of an important digital object, to understand how it was produced. This is particularly important for objects created from large, multi-source collections of personal data. As the metadata describing provenance, Provenance Data, is commonly represented as a labelled directed acyclic graph, the challenge is to create effective interfaces onto such graphs so that people can understand the provenance of key digital objects. This unsolved problem is especially challenging for the case of novice and intermittent users and complex provenance graphs. We tackle this by creating an interface based on a *clustering* approach. This was designed to enable users to view provenance graphs, and to simplify complex graphs by combining several nodes. Our core contribution is the design of a prototype interface that supports clustering and its analytic evaluation in terms of desirable properties of visualisation interfaces.

Keywords

Provenance; Visualisation; Large-scale graphs

1. INTRODUCTION

The concept of provenance (from the Latin *provenio*, "to come forth"), has been around for a long time [3]. It has been used in many other fields outside of computer science, primarily that of art and antiques, where the provenance of a piece is used as a guide to authenticity and quality. It has also been used in the field of accounting in relation to auditing as well as in databases, linking tuples in a query output to the reasons they exist.

In this paper, provenance is a form of metadata representing the lineage of a dataset or digital object. This is similar to the information stored in a version control system but goes beyond versions and authorship to capture a wider range of lineage data. Provenance makes it possible to identify and trace what other pieces of information and activities have led to a digital object being in its current state.

Large amounts of system-level, *observational* provenance can be captured automatically and inexpensively, as in PASS [10, 2]. User-level provenance, however, depends more on the specific applications like Burrito [8], an application for experimental scientists.

Burrito, and similar early interfaces, stored provenance information in a variety of flat file formats. Since 2013, the PROV specification [4] provides a standard generic model for provenance, with well-defined domain-specific extension points and a number of serialisations (RDF, XML, JSON, PROV-N).

PROV statements express "*descriptions of the entities and activities involved in producing and delivering or otherwise influencing a given object*" and they are underpinned by three main concepts illustrated in Fig. 1: (i) Entities: Physical, digital, conceptual objects, (ii) Activities: Elements that cause an entity to come into existence, and (iii) Agents: Someone or something that can be assigned responsibility for an activity taking place.

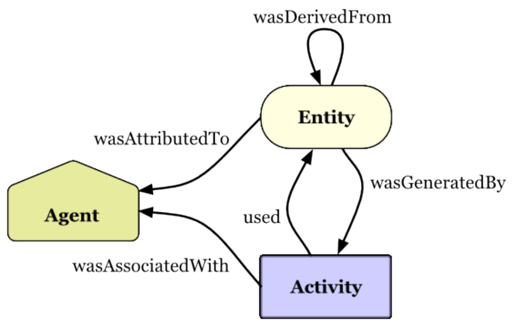

Figure 1: Key Concepts and relationships from the PROV standard displayed in a labelled acyclic graph.

Provenance data has the potential for an individual to understand and manage the sharing and use of their own data. Enabling users to understand information such as granularity or aggregation level could be important for citizen sci-

HILDA'16, June 26 2016, San Francisco, CA, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4207-0/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2939502.2939508

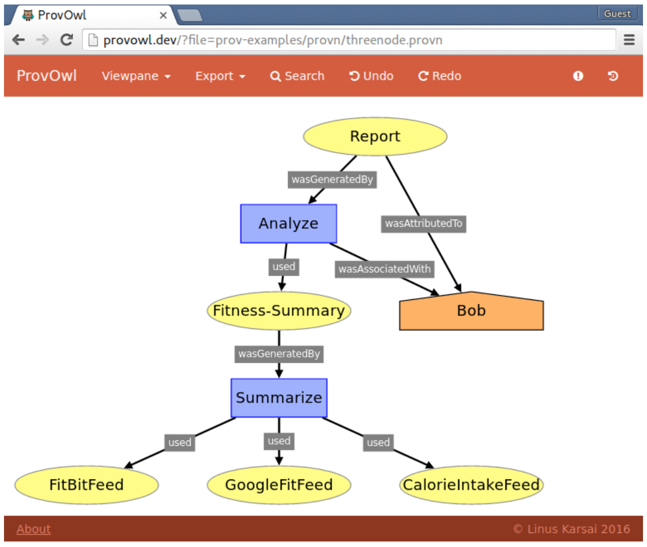

Figure 2: Provenance Data, as viewed in our *ProvOwl* prototype. This labelled acyclic graph shows Bob's role in creating a report based on his fitness data.

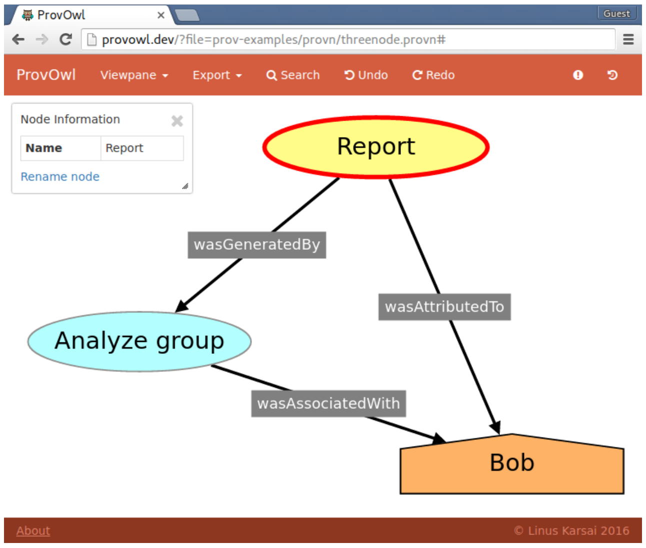

Figure 3: The graph of Fig. 2, after the user selected the `Analyse` node, with all its children, and clustered them, as a new node named `Analyse group`. The user has selected the `Report` node, indicated by the red outline, and its details appear at the top left. Contextual actions are show in blue text in the details panel: *Rename node* allows the user to change the name of the node. If multiple nodes where selected there would be an option to group those nodes.

ence contributions. This is because it is only the individual user who should decide how their data is used and with this control they may be more likely to release it. It also has potential roles in personal informatics [9] where a user manages their own streams of personal data.

Here is an example to illustrate this point. Alice has a report that describes her current fitness level as well as outlining possible improvements. Viewing the provenance of this report shows which sources have been used (e.g. FitBit[1] data to track steps, Withings[2] scale to track weight) and what processes analysed them. In this case it could show that Alice's fitness report was generated using step data from her FitBit data that had been passed through a summarisation process that summarises 15 minute interval data into daily steps. Expanding on this it also allows her to track information forward in case of errors that need correction. Alice remembers that she lent her FitBit to a friend to try for a week, causing errors in her data. Provenance allows tracing of the erroneous information forward to see what other processes and entities it effected and will need to be corrected.

Provenance data, especially of the observational, low-level kind, quickly grows in size. Whilst the examples in this paper only show graphs with a small number of nodes, a provenance file can readily grow to millions of nodes. Because provenance stores historical data without summarisation, its grows monotonically over time. The speed of this increase is directly related to the granularity that the captured provenance. So, for example, capturing user actions typically makes the increase slower than is the case for operating system actions. In both cases, however, handling these large graphs causes both technical and usability issues.

From a technical standpoint it becomes resource intensive to scan a provenance file and display a visualisation of it, more so if the application reading the file runs analysis or summarisation of the data.

Even more critically, from a usability perspective, it quickly becomes difficult for a user to interpret and explore such a large amount of information. The question then becomes how to ensure users can readily understand, explore and find the information they want from a graph. We build on research in both provenance [13, 5] and large scale graphs [12, 1] to tackle this problem by simplifying graphs. We describe our approach as *clustering*, meaning that we combine multiple nodes into a single node. This means that there are fewer nodes on the screen, with new combination concepts.

Notable amongst prior attempts at adaptive visualization of provenance graphs is the MapOrbiter [10], which is based on *semantic zoom* and applies primarily to low-level system provenance, like PASS [10]. One limitation of the approach is that the *semantic* aspect assumes a priori knowledge of the type of processes that appear in the provenance trace, and of their hierarchical relationships.

Figures 2 and 3 illustrate our approach for an entity `Report`, written by `Bob`, using information he `analysed` based on a `Fitness-Summary` that was generated by a `Summarize` script that joins information from his own fitness feeds, {`Fitbit`, `GoogleFit`, `CalorieTracker`}. Figure 3 shows the same provenance graph as Fig. 2 after all the nodes related to fitness analysis and summarisation have been clustered into one node `Analyse group`. Note that these graphs both still convey the same core concept: *Bob has written a report*

[1]FitBit is a type of fitness tracker, best known for tracking steps. https://www.fitbit.com/au
[2]Withings manufacture a digital scale that uses WiFi or bluetooth to log your weight. http://www.withings.com/us/en/products/smart-body-analyzer

regarding fitness data he has analysed himself. Even this small example illustrates clustering can simplify a graph, potentially making it more useful if the level of abstraction is right for Bob's needs.

2. CHALLENGES

This clustering seems promising to reduce the size, and so, the complexity, of the visible provenance graph in an interface. Many challenges remain about the ways to make the clustering interface work well for users, so they can do clustering, and then to explore the graph. This section discusses challenges we have encountered.

2.1 Specification of user-defined clusters

A clustering action takes a set of nodes and combines them. Our interface follows the established interface method to select multiple items, hold down *ctrl* then click each node, once multiple nodes are selected a contextual hyperlink will appear in the details panel (like the blue hyperlinks in Fig. 3) allowing the user to group them. This works well for combining a small number of nodes.

For combining larger numbers of nodes we have implemented a search feature that can be used to group multiple nodes at once. Searching via the search panel will select the nodes matching the search phrase which in turn allows grouping of these nodes. If, for example, you had 20 nodes named 'Fitbit-1-Jan-2016', 'Fitbit-2-Jan-2016'... 'Fitbit-20-Jan-2016'. You want to group each of these nodes together, instead of clicking on each node individually you can instead search 'Fitbit-[1-20]-Jan-2016' which will select all the FitBit nodes and allow them to be grouped. Currently this regex search is done on each attribute of each node. In the future it would be useful to specify and limit the regex on particular properties, this way you could search for nodes with a certain title and/or a certain property[3].

This could be further extended beyond features of the node, to include the relationship between a node and others: for example, "cluster all nodes that match the name 'Fitbit-*' and are also not used in a particular report." Furthermore during initial usability tests users requested the ability to cluster all the children of a certain node.

It may be useful to support parameterised clustering, where one command creates multiple clusters (for example, "cluster all the Fitbit nodes from each month, separately, into a month-granularity cluster'). It will be challenging to design such a language, so that it is both powerful and easy to learn and use.

Some automation may help, with where the platform suggesting clusterings, based on workloads of provenance queries recorded (for example, one might cluster nodes which are frequently accessed together).

2.2 Useful naming

Once nodes have been clustered, it is difficult to automatically generate a name that the user will find meaningful for the new cluster [12, 1]. Automating this process requires domain knowledge and it may also need deep models of the user and their needs. This would be a substantial undertaking. For example, it may demand recognition that *vim*, *emacs* and *nano* are all text editors?

[3]Depending on the way the provenance is captured different properties may be attributed to nodes. The prov standard allows an arbitrary number of key value properties to be added to a node.

Similar problems have been solved in other settings. If you create a new folder on an iPhone it will automatically be named with a label appropriate to the applications inside it (a folder of photography apps may be named 'photography'). However in this setting applications are given a category by their creator when submitted to the app store, whether it be photography, games or utilities. This metadata allows for easy naming of clustered items.

In an early version of our prototype, a new cluster-node was given a short random alpha-numeric name. However, this made the graph incomprehensible, with users needing to manually update the name immediately so that they could understand the graph. Our current, still simple approach uses the name of the node in the cluster with the shortest distance from the root with the text "group" appended to the end.

2.3 Avoiding false dependencies

Clustering is a simple type of graph rewriting, which creates an abstraction of the graph and in turn simplifies details of the original graph. This can produce false dependencies: newly implied lines of lineage, created by clustering, that falsely suggest one entity had influence upon another. This violates the main assumption that provenance records the factual history of data derivation. Conversely the opposite of this, removing a dependency, is not as large an issue because it is understood that the ability to observe data transformations is limited and therefore incomplete.

As well as false dependencies circular dependencies may also occur, along with other violations of the constraints defined in the PROV-CONSTRAINT W3C document [6]. For example, if the clustering set in the example of Fig. 2 only included nodes $\{\texttt{fitness-summary}, \texttt{CalorieIntakeFeed}\}$, then a simple replacement of these nodes with a node x would result in a circular dependency, namely
$\langle x\ \texttt{wasGeneratedBy summarize}\rangle$ and $\langle \texttt{summarize used}\ x\rangle$, you can see this in Fig. 4.

The undesirable consequences of false dependencies can include unnecessary checks and recomputation when revising dependent entities to reflect corrected or changed source data (like the example from the introduction where Alice is trying to fix data affected by erroneous FitBit data). Similarly, if provenance is used to enforce limits on where and how data is exposed, a false dependency could lead to some computations being stopped as they would be considered (incorrectly) to violate the limits. Conceptually it violates the intuition that a transitive relationship cannot be circular.

A theoretical formulation of provenance abstraction by *grouping* (clustering) has been proposed in [11] to describe this and other problems that occur with clustering, along with simple algorithms for grouping arbitrary sets of nodes. Essentially, that work showed that to avoid false dependencies, as well as circular data dependencies, one must first compute a *closure* operation that extends the user-selected nodes with all other nodes that sit on any path amongst these initial clustering nodes. Combining this prior work with our user-oriented provenance navigation model can lead to a provably correct clustering mechanism.

There is also the issue of mislabelled relationships. The labels used on relationships are specific to relationships be-

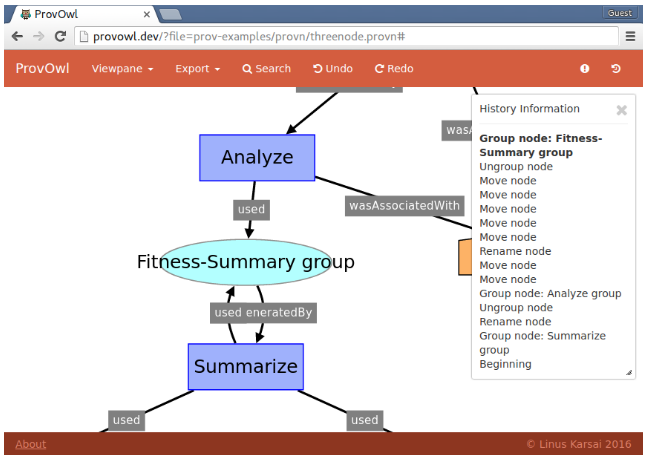

Figure 4: When grouping on {`fitness-summary`, `CalorieIntakeFeed`} a circular dependency is caused between `Fitness-Summary group` and `Summarize`. The panel on the right can be toggled by the history button above and shows a list of user actions with the current state indicated in bold. Users can use the *Undo* and *Redo* buttons to move between actions.

tween different concepts. An Entity *was generated by* an Activity or an Activity *used* an Entity. However clustering nodes create a fourth concept that uses an unlimited set of labels for its relationships. This can be confusing to users and can also reveal information about nodes in a cluster. However it would also be too restrictive to require every clustering to only group items that are from a single kind of concept. Finding suitable rules or interfaces to help the user cluster without creating mislabelled relationships is a challenge we hope to tackle in the future.

3. INITIAL INTERFACE

Our interface reads provenance data stored in the PROV format [4] and renders a directed acyclic graph that users can explore by zooming, panning, rearranging nodes and clustering nodes manually.

We implemented this as a web application. We use the Cytoscape.js[4] because it has good support for graph theory. Files are loaded completely client-side, to reduce bandwidth. If faster analysis were required, it may require use of a combination of server-side and client-side processing.

3.1 Features

Below we explain the different features of the prototype application we have built. These are loosely based on the seven visualisation tasks outlined by Ben Schneiderman [14]. The mechanism for clustering nodes is described in section 3.1.3.

3.1.1 Movement and Rearranging

On first opening a provenance graph, the viewport is positioned to fit the entire graph on screen. This gives users an overview of the provenance. Users can then move the viewport around by clicking and dragging. Zooming is accomplished by using the scroll wheel.

By default the graph's overall layout is determined using the JavaScript library dagre[5], with other layout options, such as circle and breadth-first, available using the "Reset Layout" menu (visible on the top bar, second from the left in the figures). Users can also re-arrange nodes as they wish by clicking and dragging on a node.

3.1.2 Details-on-demand

Selecting a node shows the details panel, as in Fig. 3. This displays other information about the node and contextual functions such as renaming nodes (the blue link at the bottom of the panel) or clustering nodes.

3.1.3 Clustering

Users can select multiple nodes at once, by clicking on each whilst holding down *ctrl*. Once multiple nodes have been selected, the information panel will contain a link to group the nodes. Selecting this moves the nodes together, replacing them with a single composite node, represented by a light blue oval, with default name based on the name of the node closest to the root plus the word "group".

It's also possible to search and select multiple nodes. Opening the search panel from the header allows the user to enter a string they would like to select nodes on. Once selected, nodes can be grouped the same way as above — using the group link in the information panel.

3.1.4 History

Having the ability to undo and redo actions is critical to ensure that users can confidently and safely explore the information, without fear of causing permanent damage [14]. Our interface tracks the movement and clustering of nodes. Then undo and redo buttons allow users to step through through these actions. A history pane can be toggled, by clicking the top right history icon (this pane is visible on the right of Fig. 4), to show what current step of history a user is at.

3.1.5 Sharing

The "Export" menu item (in the top bar of the figures) saves an image of the current graph with all the user modifications. This can either save the entire graph or be limited to the current viewport, if the user wanted to focus on a certain section.

3.2 Planned Features

The prototype source code is available on GitHub[6]. The interface is also available online where you can test it with a sample graph[7]. We describe features under development to improve usability.

We propose to improve on the regular expression language to select nodes, as mentioned in the section on challenges. This will enable users to select nodes in cases such as: (i) Select all nodes *not* influencing the "Summarize" node (ii) Select all children of "Fitness-Summary". For large graphs,

[4]Cytoscape.js: Graph theory library for analysis and visualisation http://js.cytoscape.org/

[5]Dagre: supports lay out of directed graphs client-side. The main skeleton of algorithm comes from "A Technique for Drawing Directed Graphs" [7]

[6]https://github.com/karsai5/ProvOwl

[7]http://provowl.com/?file=/prov-examples/provn/threenode.provn

this could allow for faster user-directed simplification of a graph.

As an extension of this, the language should describe parameterised clustering, so users ask for multiple similar clusters to be formed. For example "Create clusters from nodes the same depth from the root" or even using data inside the nodes "Create clusters from nodes that have the same creation date".

This could also impact the way nodes are automatically named. If the user created clusters from nodes that all have the same creation date, the system may infer that the name for the new node should include the creation date.

We also wish to extend the PROV standard to include descriptions of cluster nodes. This would allow a user to cluster nodes manually, export the PROV file with cluster descriptions and then share with someone else. This *extraction*, being able to export your current state of exploration, would allow further exploration from the current state later on or even sharing with other users for further analysis.

4. REFERENCES

[1] J. Abello, F. Van Ham, and N. Krishnan. ASK-GraphView: A large scale graph visualization system. In *IEEE Transactions on Visualization and Computer Graphics*, volume 12, 669–676, 2006.

[2] N. Balakrishnan, T. Bytheway, R. Sohan, and A. Hopper. OPUS: A Lightweight System for Observational Provenance in User Space. In *USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 8, 2013.

[3] D. Bearman and R. Lytle. The Power of the Principle of Provenance. *Archivaria*, 21(February 1982):14–27, 1985.

[4] K. Belhajjame, H. Deus, D. Garijo, G. Klyne, P. Missier, S. Soliand-Reyes, and S. Zednik. PROV Model Primer. In *W3C Working Group Note*, 2013.

[5] M. A. Borkin, C. S. Yeh, M. Boyd, P. MacKo, K. Z. Gajos, M. Seltzer, and H. Pfister. Evaluation of filesystem provenance visualization tools. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2476–2485, 2013.

[6] J. Cheney, P. Missier, and L. Moreau. Constraints of the Provenance Data Model. Technical report, 2012.

[7] E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo. A Technique for Drawing Directed Graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.

[8] P. Guo and M. Seltzer. BURRITO : Wrapping Your Lab Notebook in Computational Infrastructure. In *USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 4, 2012.

[9] I. Li, Y. Medynskiy, J. Froehlich, and J. E. Larsen. Personal informatics in practice: improving quality of life through data. *CHI Extended Abstracts on Human Factors in Computing Systems*, 2799–2802, 2012.

[10] P. Macko, M. Chiarini, and M. Seltzer. Collecting Provenance via the Xen Hypervisor. In *USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2011.

[11] P. Missier, J. Bryans, C. Gamble, V. Curcin, and R. Danger. Provabs: Model, policy, and tooling for abstracting PROV graphs. In *International Provenance & Annotation Workshop (IPAW)*, 2014.

[12] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer-Human Interaction*, 3(2):162–188, 1996.

[13] M. Seltzer and P. Macko. Provenance Map Orbiter: Interactive Exploration of Large Provenance Graphs. In *USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2011.

[14] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *IEEE Symposium on Visual Languages*, 336–343, 1996.