SAPVis: Visualizing a SAP Network

CS547 Status Update

Vaden Masrani
vadmas@gmail.com

November 25, 2015

1 Status Update

Firstly to address some of the main points from the proposal feedback. I wasn't clear about what work had been done prior to this class and what work has been done as a part of this project. Prior to the class I had experimented with a 2D force directed layout approach using a javascript library called VisJs and this resulted in an unreadable hairball and poor performance. After a series of meetings with you I settled on a one dimensional arc-layout coupled with a hierarchical tree diagram to show parent-child subgraphs within the network. I sketched out the initial diagram on paper then took it to a friend who used Photoshop to make a sketch that I could then use to present to the class. The experimentations with 2D force directed layout and the use of Photoshop were done outside of this class; the arc-layout idea, the arrangement of the dashboard, and the implementation using D3 are all done by me and as a part of this project.

On the question of scope, you recommend to focus less on robustness and more on rapid iteration, at least at these early stages. Point taken, and I am slightly ahead of the schedule I laid out in the proposal and have managed to handle the scale concerns you raised. This has been the biggest challenge of the project so far and I felt it was important to address scale at the onset. I was able to implement a toy arc diagram using D3 (approx. 70 nodes, approx. 150 edges) with responsive pan and zoom functionality, but when I tried running on a larger dataset (approx. 2000 nodes, approx. 15000 edges) the performance suffered drastically. The issue here isnt screen space, as I am allowing the nodes to overlap initially and un-overlap as the user zooms in. This easily allowed 2000 nodes to fit on the axis. The problem is that on large dataset the zoom performance is so lousy that the system is unusable.

I tried a number of approaches to fix this performance issue, (draw the arcs as one giant path DOM element rather than 15000 small path DOM elements, rebind the data with small selections on zoom, dont store edges in memory that are outside the view port) but none of these changes improved the usability. The solution was to use HTML5s canvas element to draw the paths instead of 15000 path SVGs. Using canvas dramatically improved

the performance (while sacrificing the fancy animations that come from manipulating SVG elements). The user can also choose to hide the edges to improve the performance further.

I have spend a lot of time on the vis and currently have a system that takes a json, lays out the nodes along the x axis and draws arcs between connected nodes. It handles zoom, pan and translate, and can re-sort (with animation) either based on type, degree, name (alphabetical) or value. The user can also select which attribute to use as the nodes value and the nodes will resize according to the user selection. When the user hovers over a node, its arcs highlight and the other arcs in the network fade out in order to show which arcs are emanating from the node. The colour of a particular arc matches the colour of its parent node. There is also a tooltip which displays the name underneath the node when the hover listener is activated. Finally, the user can chose to hide all the edges to improve the responsiveness when zooming.

There is a lot more to do. In rough order, I need to: Allow the user to select a node and highlight the corresponding parent/child nodes, change the layout algorithm to cluster nodes by type, change the resorting algorithm to only resort each node with its type, add dropdown tables above each type to display the id and value as per the mockup diagram, fix a few known bugs, decide how to handle null attributes, create the filter panel to allow the user to select attribute ranges, add the hierarchical tree diagram, add CSV export functionality, restyle, and test. Learning D3 has been challenging but Im hoping Ill be able to focus on rapid iteration now that Ive passed the initial D3 learning curve.

2 Previous Work

There have been many proposed network visualization idioms. The most common and most intuitive is a node-link (NL) diagram, where each item is represented by a point in 2D space and lines are drawn to show connections between items. Data attributes can be encoded by the size, shape and colour of the nodes and edges. The NL idiom works well with small, sparse networks but suffers from readability and performance issues with large and highly connected networks. Computing the positions of the nodes can be a challenge, particularly with large networks which are prone to producing "hairballs". Force directed layout algorithms are the most common algorithms for computing node layouts, but they are notoriously brittle on large networks, and are non-deterministic which can cause confusion with the user. Alternative layouts include Chord Diagrams which lay the nodes on the circumference of the circle or arc diagrams which lay the nodes along one dimension [4]. Computing the position of nodes in 1D is inexpensive compared with the force directed layout algorithms and arc diagrams avoid the hairball problem, but they can still suffer from occlusion and edge crossing issues, especially as the size of the network grows.

A common alternative to the node-link diagram is an adjacency matrix (AM), where networks are displayed in a symmetric 2D matrix with the rows and columns each representing the nodes, and an entry at position M(i,j) representing an edge between node i and j. The entries can be filled with a boolean, an edge weight, or a colour to encode an

edge attribute. Adjacency matrices can handle large and dense matrices but their major weakness is unfamiliarity; most users need to be taught how to read adjacency matrices and the steep learning curve can be a hurdle, especially for non technical user [5].

There has been work in trying to overcome the limitations of both views by combining adjacency matrices and node-link views. For example, because tracing paths can be more challenging in adjacency matrices than in NL diagrams, MatLink [2] overlays the matrix views with an arc diagram along the row and column headers to facilitate path tracing. MatrixExplorer [1] provides two synchronized views (matrix and NL) faceted together along with manipulation tools - filtering, clustering, reordering, sorting, zoom and pan, annotation - to allow the users to explore the data. NodeTrix [3] shows multiple adjacency matrices linked together by first showing the network as a NL diagram and allowing the users to select dense regions of the network to be shown as an adjacency matrix.

Another approach in visualizing large networks is to aggregate the data in such a way as to provide the user with a meaningful abstractions of the entire network. This gives the user metadata on the network as a whole rather than displaying the exact network topology. PivotGraph [8] does this by performing a roll-up operation to combine similar nodes and display them on an grid with two categorical attributes along the axes. Instead of displaying every node and arc in the network, aggregate nodes and arcs display the relationship between categorical attributes in the network as a whole. The Honeycomb [7] system also aggregates nodes, but instead uses a predefined hierarchy to aggregate cells of an adjacency matrix in order to display social networks with millions of connections. Elzen and Wikjs DOSA system [6] aggregates user selections and displays meta-information, such as number of nodes and edges with the selection, in a high-level overview display alongside the original network. The original network is displayed as a scatterplot with the x and y axis encoding two of its attributes. Filtering and selection tools are provides to the user to allow them to select which areas of the network they wish to aggregate.

In a similar fashion to the DOSA and Matrix Explorer system, this work will offer two complementary views and a set of manipulate, filter and explore tools to the user. An arc diagram will serve as the network overview and collapsible tree network will display selected subgraphs within the network. It is important to preserve network topology in the the context understanding an SAP system so this work will not aggregate nodes like was done in the PivotGraph and Honeycomb systems. To handle the scale issues that come with a NL view, we will instead allow the user to filter based on attributes in order to find outliers and extremum (eg. nodes with low performance and high usage scores.) We have opted to use a node-link view over an adjacency matrix because we want to minimize the learning curve of the tool and facilitate easy path exploration.

References

[1] N. Henry and J. Fekete, *Matrixexplorer: a dual-representation system to explore social networks*, Visualization and Computer Graphics, IEEE Transactions on, 12 (2006), pp. 677–684.

[2] N. Henry and J.-D. Fekete, *Matlink: Enhanced matrix visualization for analyzing social networks*, in Human-Computer Interaction INTERACT 2007, C. Baranauskas, P. Palanque, J. Abascal, and S. Barbosa, eds., vol. 4663 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 288–302.

[3] N. Henry, J.-D. Fekete, and M. J. Mcguffin, *Nodetrix: a hybrid visualization of social networks*, IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS (IEEE VISUALIZATION CONFERENCE AND IEEE CONFERENCE ON INFORMATION VISUALIZATION PROCEEDINGS, 13 (2007), pp. 1302–1309.

[4] M. J. McGuffin, *Simple algorithms for network visualization: A tutorial*, Tsinghua Science and Technology, 17 (2012), pp. 383–398.

[5] T. Munzner, *Visualization Analysis and Design*, A.K. Peters visualization series, A K Peters, 2014.

[6] S. van den Elzen and J. J. van Wijk, *Multivariate network exploration and presentation: From detail to overview via selections and aggregations*, IEEE Trans. Vis. Comput. Graph., 20 (2014), pp. 2310–2319.

[7] F. van Ham, H.-J. Schulz, and J. Dimicco, *Honeycomb: Visual analysis of large scale social networks*, in Human-Computer Interaction INTERACT 2009, T. Gross, J. Gulliksen, P. Kotz, L. Oestreicher, P. Palanque, R. Prates, and M. Winckler, eds., vol. 5727 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 429–442.

[8] M. Wattenberg, *Visual exploration of multivariate graphs*, in Proceedings of ACM CHI 2006 Conference on Human Factors in Computing Systems, ACM SIGCHI CHI - Human Factors in Computing Systems conference series, ACM Press, 2006, pp. 811–819.