Adjasankey: Visualization of huge hierarchical weighted and directed graphs

Joris Sansen, Frédéric Lalanne, David Auber, Romain Bourqui

Adjasankey: Visualization of huge hierarchical weighted and directed graphs

Joris Sansen, Frédéric Lalanne, David Auber and Romain Bourqui
Université de Bordeaux
{jsansen, lalanne, auber, bourqui}@labri.fr

Abstract

Visualization of hierarchical weighted and directed graphs are usually done with node-link or adjacency matrix diagrams. However, these representations suffer from various drawbacks: low readability in a context of Big Data, high number of edge crossings, difficulty to efficiently represent the weighting. With the stated goal of reducing these drawbacks, we designed Adjasankey, a hybrid visual representation of weighted and directed graphs using hierarchical abstractions. This technique combines adjacency matrices readability of large graphs and flow diagrams visual design efficiency for weighting depiction. Associated to Big Data computing and light-weight web rendering, our tool allows to depict and interact in real time on huge dataset and supports user multi-scale exploration and analysis. To show the efficiency of Adjasankey, we present a case study on the analysis of a Customer to Customer *website.*

1 Introduction

In various fields, analysing how data flow through a system is central to monitor, understand and predict variations of this system. For example determining in social network, how rumors spread and fade, who starts or stops rumors and if they are transmitted to different communities are important questions. In the same way, in web analytics, users navigation information is valuable and its analysis brings plenty of information on buying habits, fashionable articles, what leads a customer to a purchase or on the contrary, what hinders him/her. Due to improvement in data acquisition techniques, the size and complexity of such data prohibit manual representation. This is particulary true in the era of Big Data as dataset size can exceed Mega/Giga/Tera bytes. Information visualization therefore focuses on the design of automatic, fast and efficient techniques.

A common approach consists in modeling such data with a directed weighted graph where the nodes represent the network entities, the edges and their associated weight represent the information flowing between entities. Various techniques have been proposed during the last decades to visualize such directed weighted graphs. Among these techniques, the most popular are node-link diagrams and adjacency matrices. In node-link diagrams (*e.g.* [13, 16]), nodes are visually encoded with a shape (usually a square or a circle) and an edge between two nodes is represented as a poly-line between the corresponding shapes. In adjacency matrices (*e.g.* [8, 17]), nodes are displayed twice, on the abscissa and on the ordinate, and an edge between two nodes is represented by a shape in the corresponding row and column. These visualization techniques have been experimentally evaluated [9, 12] in order to determine which depiction performs better in different cases. These studies showed that adjacency matrix outperforms node-link diagram when the considered graph becomes large and dense. Other researches defined hybrid encodings that try to take advantage of several methods (*e.g.* [6, 10, 20]).

To highlight interesting parts of the network, a hierarchical partition of the graph can be used. Such a hierarchical partition can either be inherent to the data or computed with a clustering algorithm (*e.g* [5, 18]). In that case, the main idea is to emphasize each set of the partition as well as the relationships within and between them. Again, both node-link diagram and adjacency matrix have been used to show this hierarchical partition (*e.g.* [7, 11, 14]).

Even if the techniques mentioned above can scale up to millions of elements, three importants issues still remain: *i*) clutter *ii*) lack of efficiency for weighting representation, and *iii*) slow rendering. The clutter (node-node overlaps, node-edge overlaps and edge-edge crossings) increases with the size and the complexity of the graph, resulting in unreadable visualizations. Moreover, depicting weighting of the graph increases this effect. Also, a large number of elements to be displayed may lead to slow rendering and therefore hinder interactive exploration.

To reduce visual clutter and speed up the rendering, a common approach is to build an *abstraction* of the original graph. It allows to guide the user in his/her analysis by identifying interesting parts of the network and reduces the focus of his/her study. This is usually done by replacing each subset of a partition of the nodes by a single node (called *metanode*); and replacing all edges linking two of these subsets by one single edge (called *meta-edge*), linking the corresponding metanodes. This operation reduces the number of displayed nodes and edges and therefore reduces visual clutter and increases rendering speed. This abstraction is usually called a *compound* graph, associated with the original graph and the partition. If repeated iteratively, this process allows to build abstractions of higher and higher levels. This use of such method have already been widely studied during the last decade and applied to various depiction techniques [1, 3, 4, 8].

Depicting weighting of relations is possible for most of the above techniques but they suffer from important drawbacks (*e.g.* lack of efficiency, clutter, readability). A common and effective weighting depiction relates on representation inspired from flow diagrams. Since the first diagrams to modern works [2, 15, 19, 21] its visual design kept its efficiency. Such technique emphasizes major trends while depicting quantities and thus allows to locate dominant contribution to the overall flow. However, it faces the problem of scalability and clutter when the number of elements to represent increases. In our knowledge of the visualization domain, there is a lot of publications focusing either on directed graphs, or on weighting depiction, or on hierarchical depiction and usability as a tool but very little on efficient depiction of the three all together.

In this paper, we present *Adjasankey*, a new visualization technique for representing weighted directed graphs together with a hierarchical partition. To do so, Adjasankey takes advantages of compound graph visualization, adjacency matrix layout and Sankey's diagram that supports the visualization of flows between entities. The aim of Adjasankey is to help the user to answer the following questions: *i*) Which entities are connected to important flows? *ii*) Does information mainly flow between or within clusters ? *iii*) How many flows start from or end to a particular cluster/entity?

The remainder of this paper is structured as follows. In Section 2, we present Adjasankey: design and technical aspects. Then we describe it's implementation in Section 3. We provide some results with a case study based on the analysis of a website in Section 4. Finally, we present conclusions and discuss possible future work.

2 Adjacency matrix layout for Sankey diagram : Adjasankey

This section presents the design of Adjasankey which uses a node positionning method similar to adjacency matrices combined to Sankey's diagram design for edges. As mentioned in Section 1, we consider the data as a directed and weighted graph together with a hierarchical tree. One can see in Fig. 2(a) an example of a clustered directed graph and the associated hierarchy tree as used as input for our representation.

2.1 Positioning and sizing

In Adjasankey, such as in an adjacency matrix, nodes are displayed on rows and columns. As for matrices that

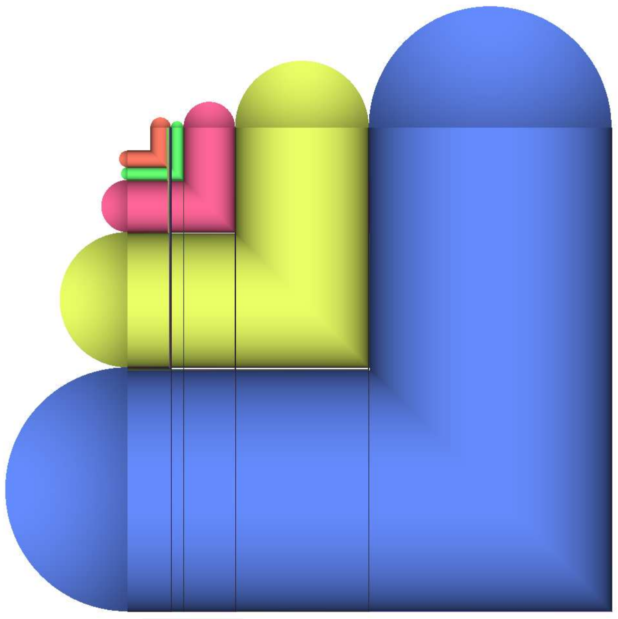


Figure 1: Visualization of the highest level of abstraction with Adjasankey obtained on a dataset of $46,500,000$ clicked hyperlinks in a C2C website.

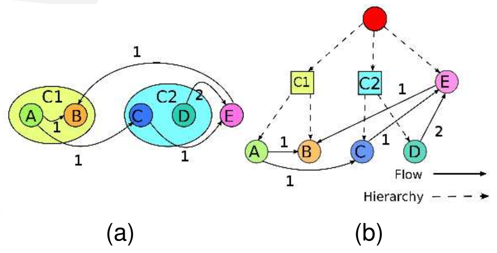
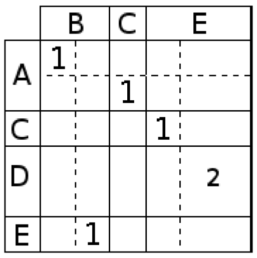
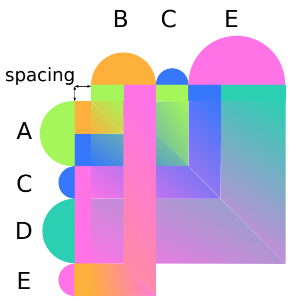


Figure 2: Example of directed weighted graph (a) and associated hierarchical partition (b).

represent a directed graph, source nodes are laid out on ordinate and target nodes on abscissa. A well-known matrix issue lies in nodes ordering which can reveal or hide various information. Such node ordering problem is known to be NP-hard and is not the aim of this study. In our method, the height of a node row (resp. width of a node column) is set according to its weight (sum of weights of its incident edges). It helps to identify nodes (and clusters) that are incident to edges with high weights. Since node height (resp. width) is set according to its weight, a node having an out-degree (resp. in-degree) equal to 0 is not displayed on abscissa (resp. on ordinate) which reduces the matrix size. In Fig. 3(a), one can see that the height of E on ordinate is set to 1 while the width of E on abscissa is set to 3. One can also notice that node B is not drawn on ordinate



(a) Node and edges placement



(b) coloring

Figure 3: (a) Nodes and edges positionning on matrix layout according to the input graph Fig. 2; (b) resulting visualization with node and edge coloring.

as its out-degree equal to 0. The same observation can be done for nodes A and D on abscissa.

In adjacency matrix, node column and row contains multiple connections to depict all its incident edges. As a node height or width has been set according to its out- and in-degree, it can be split to provide a complete row/column to each of its incident edge and to set their size proportionally to their weight. In Fig. 3(a), the column of node E have been split into two row of width 1 and 2 as node E has two in-coming edges of weight 1 and 2.

Such node and edge positioning allows to bundle together edges having the same source or target without misrepresentation of weights. In order to strengthen this bundling, edges are ordered according first to the in-degree nodes order and then to the out-degree nodes order.

2.2 Coloring

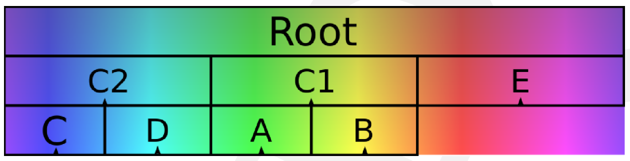


Figure 4: H-range of HSV colorspace distribution for node coloring.

In order to emphasize the hierarchical partition of the graph, we designed an algorithm which tries to affect similar colors to nested clusters. The main idea of that algorithm is to assign to each node of the hierarchy tree, a H-range of value from the HSV colorspace. First an H-range of $[0, 360]$ is assigned to the root node of the hierarchy tree. Then, for each internal node, its H-range is split according to the weight of its sons in the hierarchy tree and the resulting ranges are assigned to them. For instance in Fig. 4, the H-range $[0, 360]$ of the root node is equally split and assign to C1,C2 and E as their total weight are all equal to 4. Finally, the hue value of each node of the hierarchy tree is set to the middle of its H-range. The positioning of nodes and edges allows to preserve for each edge an entire row and column. These row and column can therefore be colored to emphasize edges source and target. Edges color are set according to their source and target colors with a progressive and reversed interpolation. On the source (resp. target) side, edges are colored with target (resp. source) color. Such interpolation results in a cumulative histogram that shows the distribution of weights over the successors (resp. predecessors). For instance, in Fig. 3(b), we can see that in-coming flows of node E on abscissa are emitted from node C and D. Furthermore, the colors emphasize the contribution of node C and D (resp. one third and two third). So as to guarantee the visibility of edges and histogram, a spacing is kept near the entities.

2.3 Rendering

Due to node placement, adjacent edges are bundled together. In order to emphasize these bundles of edges, they are depicted with a luminance texture that creates a tubular effect. Furthermore, it eases the distinction between edges crossings and edges bends. Indeed crossings identification without such tubular effect can be tedious (see Fig. 3(b)) even when edge borders are drawn. However, we are aware that such design have the side effect to modify color intensity which is a limitation of the technique.

2.4 Interaction

The first depiction of the graph is the highest level of abstraction provided by the hierarchical partition (see Fig. 1). We provide together with Adjasankey an interaction tool that allows to modify the diplayed level of detail. Using that interaction tool, the user can request more (resp. less) details on a given cluster. It induces the appearance of descendant nodes and the break up of meta-edges in the corresponding edges, potentially metanodes and meta-edges themselves. In addition to that interaction tool, we also provide a classical zoom-and-pan as well as a neighbor identification interaction tools. The latter allows to highlight in the representation the neighborhood of a given node but also the source and target of a given edge/bundle of edges (see Fig. 7).

3 Implementation

The size of the data that Adjasankey have to handle makes impossible to store and manipulate them on a single computer. In this section, we present some technical aspects of Adjasankey, and explain how we combined a cloud architecture for the pre-processing of the data and a light-weight client that make possible to visualize an abstraction in real-time in web browser.

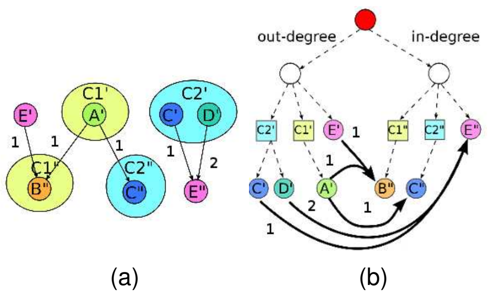


Figure 5: Nodes having both ingoing and outgoing edges in the graph of Fig. 2 are duplicated in both the graph (a) and the hierarchy (b).

Pre-processing on a cloud architecture To request more or less detail on a particular metanode in reasonable time, the first pre-processing step consists in duplicating the graph and hierarchy tree nodes. For instance in Fig. 5, one can see the result of that duplication step on the graph and hierarchy of Fig. 2. During this process, entities E and C are duplicated since they emit and receive flows while A, B and D are not (see Fig. 2(b)). The hierarchy tree is also updated and clusters containing both sources and targets are duplicated as well. As C1 and C2 contain both emitting and receiving entities (A and B for C1, C for C2), they are duplicated into C1'/C1" and C2'/C2". The second step consists in generating the meta-edge connecting each node of the *duplicated* hierarchy tree to all other possible neighbors. These two steps are achieved using Apache Spark (a distributed data processing framework) and then stored in an HBase database.

A web service finally manages access requests and connection to HBase for data transmission. To improve query response time, our system also keep in memory nodes and metanodes that have already been requested. Using such an architecture, the system can answer in reasonable time when the user requests more (or less) details on the cluster of the hierarchy.

Graphic User Interface Once the duplicated graph and hierarchy tree have been computed, the visualization program is only responsible for the rendering of the coumpound graph. The rendering program is implemented in C++ associated to openGL shader programming. It is then transformed into Javascript and WebGL using the llvm-based project Emscripten [22]. The generated program is integrated within the visualization client written in html.

4 Case study: website user navigation

This section presents the results that we have obtained on a website user navigation dataset. More precisely, the dataset contains a set of $46,500,000$ clicked hyperlinks in a *Customer To Customer* website containing approximatively $50,000$ webpages and $823,000$ hyperlinks. The dataset have been modeled as an oriented (hyperlinks direction) and weighted (number of clics per hyperlink) graph. The hierarchical tree associated to the graph is given by the hierarchical partition in categories (sub-categories and so on) of products for sale. After the duplication pre-process and the computation of the meta-edges, we obtain a set of $96,300$ nodes/metanodes and $1,291,300$ edges/meta-edges.

Fig. 1 shows the representation in Adjasankey of the highest level of abstaction of the data, *i.e.* the representation of the main categories of products for sale on the website. One can easily notice that the website contains five main categories (or webpages) and that the flows are mainly within category. Among these categories, the *Professional equipment* category (in blue in Fig. 1) represents approximatively 50% of the visited webpage while the *House equipment* represents approximatively 30%. Surprisingly, the home page of the website only represents 4% of the visited pages. Two factors can explain that phenomena, first many users enter on the website using search engines that redirect them directlty on particular webpages and second, the number of times the homepage is visited is low compared to the number of visited webpages per visit.

Within the *Professional equipment* category (see Fig. 6(b) and 7(a)), most of the flows are again within their sub-categories which means that, at this level of organization, the website seems also well-structured. One can also notice that the *Agricultural equipment* sub-category represents about 50% of the visited webpages of *Professional equipment* (in yellow in Fig. 7(a)). Other sub-categories either refer to other kinds of equipement or to territorial regions. Two type of behaviours can be observed: *i*) Users who navigate in territorial regions sub-categories, *ii*) Users who navigate in semantic sub-categories and then, at lower levels, in territorial regions categories. Such behaviours reveal a distinction between users who wants to promote either territorial proximity or accuracy of their search. As no offer can appear in two categories, it also indicates that users can obtain different results if they choose to follow a territorial strategy or not.

Considering the *House equipment* category (see Fig. 6(c) and 7(b)), one can observe two major sub-categories (*Gardening* and *Shoes*), one medium category (*Baby clothes*) and two small categories (*Luggage* and *Tools*). While major part of the flows are within categories, two interesting behaviors can be noticed: *i*) almost all of the flows connected to *Tools* are emitted from and toward *Gardening*, *ii*) almost all of the flows connected to *Luggage* are emitted from and toward *Shoes*. Fig. 7(c) and 7(d) show

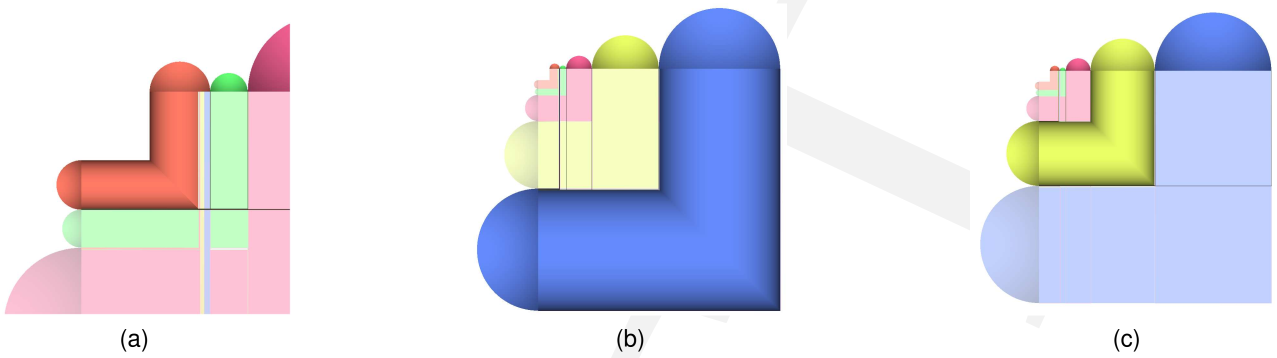


Figure 6: Focus on the home page (a) and on the *Professional equipment* (b) and *House equipment* (c) categories.

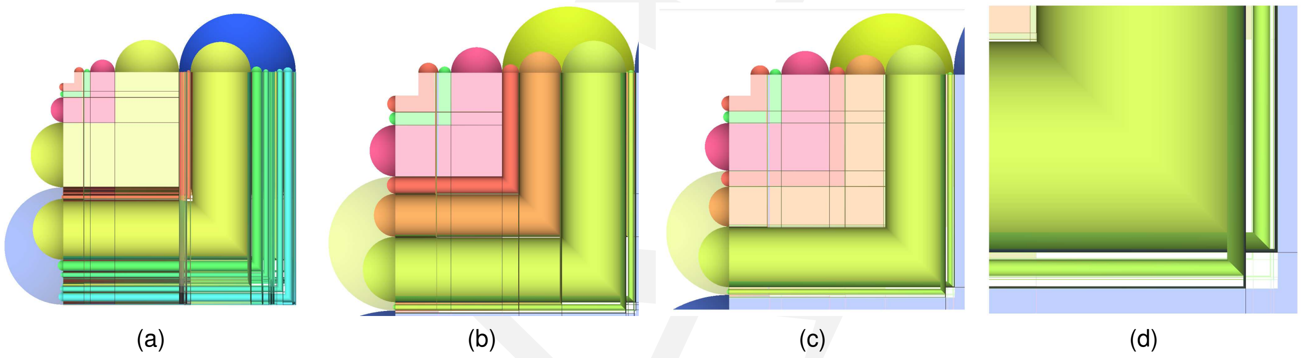


Figure 7: Different levels of detail and zoom levels. (a) Expansion of *Professional equipment* metanode; (b) Expansion of and zoom on the *House equipment* metanode; (c) Flows emitted from the *Gardening* category ; (d) Details of *Tools* (in green) and *Gardening* (in yellow) flows: most of the flow emitted from *Tools* is emitted toward *Gardening*.

the case of *Tools* and *Gardening*, when approximatively 90% of the flows from *Tools* are emitted toward *Gardening*. We assume that this is the result of a miss-classification of the offers and that the *Gardening* category should be included in the *Tools* one. Indeed many offers of the *Gardening* category refers to gardening tools offers. The same reasoning applies to the flows between *Luggage* and *Shoes* flows case.

5 Conclusion and future work

We have presented Adjasankey, a novel visualization technique designed for weighted directed graphs associated to an hierarchical partition. Our visualization method combines Sankey's diagram visual design with an adjacency matrix layout and thus emphasizes edge weighting while optimizing overall readability. To scale to huge dataset, Adjasankey supports the multi-scale exploration of various levels of abstraction. We have implemented our solution in a Big Data infrastructure with cloud computing for data pre-processing and a light-weight client for visualization. This system allows a time-responsive interaction and visualization of various abstractions of large network. A trial of this system was made with a case study on website user navigation and was positively received by domain experts.

As future work, we plan to extend Adjasankey to path or sequence visualization as a combination of our depiction and of space-filling curves. We also plan to perform an experimental evaluation for comparing classical adjacency matrix diagram and Adjasankey for the depiction of directed and weighted graphs.

Acknowledgement

This work has been carried out as part of SpeedData project supported by the French Investissement dAvenir Program (Big Data - Cloud Computing topic) managed by Direction Gnrale des Entreprises (DGE) and BPIFrance.

References

[1] J. Abello, F. van Ham, and Neeraj Krishnan. Askgraphview: A large scale graph visualization sys-

tem. *Vis. and Comp. Graphics, IEEE Trans. on*, 12(5):669–676, 2006.

[2] H. Alemasoom, F.F. Samavati, J. Brosz, and D. Layzell. Interactive Visualization of Energy System. In *Cyberworlds (CW), 2014 International Conference on*, pages 229–236, 2014.

[3] Daniel Archambault, Tamara Munzner, and David Auber. GrouseFlocks: Steerable Exploration of Graph Hierarchy Space. *IEEE Transactions on Vis. and Comp. Graphics*, 14(4):900–913, 2008.

[4] M. Balzer and O. Deussen. Level-of-detail visualization of clustered graph layouts. In *2007 6th International Asia-Pacific Symposium on Visualization, 2007. APVIS '07*, pages 133–140, 2007.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[6] K. Dinkla, M.A. Westenberg, and J.J. van Wijk. Compressed Adjacency Matrices: Untangling Gene Regulatory Networks. *Vis. and Comp. Graphics, IEEE Trans. on*, 18(12):2457–2466, 2012.

[7] Peter Eades and Qing-Wen Feng. Multilevel visualization of clustered graphs. In *Graph drawing*, pages 101–112. Springer, 1997.

[8] N. Elmqvist, Thanh-Nghi Do, H. Goodell, N. Henry, and J. Fekete. ZAME: Interactive Large-Scale Graph Visualization. In *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific*, pages 215–222, 2008.

[9] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, 2005.

[10] M. Hlawatsch, M. Burch, and D. Weiskopf. Visual Adjacency Lists for Dynamic Graphs. *Vis. and Comp. Graphics, IEEE Trans. on*, 20(11):1590–1603, 2014.

[11] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Trans. on Vis. and Comp. Graphics*, 12(5):741–748, 2006.

[12] Ren Keller, Claudia M. Eckert, and P. John Clarkson. Matrices or Node-link Diagrams: Which Visual Representation is Better for Visualising Connectivity Models? *Information Visualization*, 5(1):62–76, 2006.

[13] E. M. Kornaropoulos and I. G. Tollis. DAGView: An Approach for Visualizing Large Graphs. In *Graph Drawing*, volume 7704 of *Lecture Notes in Computer Science*, pages 499–510. Springer Berlin Heidelberg, 2013.

[14] Steven Noel and Sushil Jajodia. Understanding complex network attack graphs through clustered adjacency matrices. In *Computer Security Applications Conference, 21st Annual*, pages 10–pp. IEEE, 2005.

[15] P. Riehmann, M. Hanfler, and B. Froehlich. Interactive sankey diagrams. In *Information Vis., 2005. INFOVIS 2005. IEEE Symposium on*, pages 233–240, 2005.

[16] D. Selassie, B. Heller, and J. Heer. Divided Edge Bundling for Directional Network Data. *IEEE Trans. on Vis. and Comp. Graphics*, 17(12):2354–2363, 2011.

[17] Zeqian Shen and Kwan-Liu Ma. Path Visualization for Adjacency Matrices. In *EuroVis07: Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 83–90, 2007.

[18] Stijn Marinus Van Dongen. *Graph clustering by flow simulation*. PhD thesis, University of Utrecht, 2000.

[19] C. Vehlow, F. Beck, P. Auwrter, and D. Weiskopf. Visualizing the Evolution of Communities in Dynamic Graphs. *Comp. Graphics Forum*, 34(1):277–288, 2015.

[20] B. Watson, D. Brink, M. Stallmann, R. Devarajan, M. Rakow, T. M. Rhyne, and H. Patel. Visualizing very large layered graphs with quilts. *IEEE Info. Vis. Poster*, 2007.

[21] K. Wongsuphasawat and D. Gotz. Exploring Flow, Factors, and Outcomes of Temporal Event Sequences with the Outflow Visualization. *IEEE Trans. on Vis. and Comp. Graphics*, 18(12):2659–2668, 2012.

[22] Alon Zakai. Emscripten: an LLVM-to-JavaScript compiler. In *Proc. of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 301–312. ACM, 2011.