DBL HTI + Webtech (2IOA0)
2019 Q4 - Visualization Project Description

Assistant Professor Dr. rer. nat. Dipl. Inform. Michael Burch

Eindhoven University of Technology
Department of Mathematics and Computer Science
Algorithms and Visualization W&I

Abstract. Interactively representing graphs and networks is a challenging task, but it is an important one for many people all over the world. This kind of 'relational' data is existing in many application domains like software engineering, biology, social networking, eye tracking, and many more. Building a visualization tool that is accessible online in a web browser is a suitable solution since then it can be used by many people, either from industry, academia, or just laymen playing around with the data, without installing complex software adjusted to their environments. Since many visualization techniques for graph data have been developed over the years, a multiple coordinated views design should provide several perspectives on the graphs with integrated brushing and linking techniques. Those perspectives should at least show node-link diagrams as well as adjacency matrices, supporting several layouts and link representation styles for the node-link diagrams and different ordering strategies for the adjacency matrices. Providing a web-based tool can also be beneficial to work in a collaborative manner, i.e., insights from many people all over the world might be stored with the goal to explore a graph dataset much more rapidly by exploiting the strengths of the perceptual abilities of many users instead of an individual one.

1 Introduction

Graphs and networks are omnipresent in many application domains [13]. For example, in biology, protein-protein interactions are oftentimes of special interest. Software systems contain call relations describing which functions are calling other functions. Also in social networking (like Facebook, Twitter, or Instagram) people interact with eachother (for example by sending messages or just by knowing eachother), building weighted networks. In scientific research, paper authors reference eachother building a citation network. There are various examples from the real world, providing a large repertoire for additional dataset scenarios.

However, what all of those dataset examples have in common is their relational structure, i.e., they consist of a set of objects/elements between which relations exist. Those relations can have many forms, they can be directed or undirected and can even carry a weight in their simplest case. In graph terminology we speak of vertices and edges that build a mathematical graph model

expressed by $G = (V, E)$ where V denotes the finite number of vertices and $E \subseteq V \times V$ the finite number of edges. An edge weight function

$$w : E \longrightarrow \mathbb{R}$$

may model weights on those edges, for example, how many messages have been sent between two people in a social network or how long a call between two software functions persisted. It may be noted that we speak about vertices and edges if we focus on the data model, but we talk about nodes and links if we refer to the visual representation as a node-link diagram.

Nowadays, graphs can grow large in size containing hundreds of thousands or millions of elements with even more relations between those elements. Figure 1 shows a small graph in a node-link visual metaphor. Here you see a radial/circular layout in which the 5 vertices of the graph are placed equidistantly on a circle circumference. The graph is a directed and weighted one containing 7 edges. If we have to deal with a directed and weighted graph, we refer this to as a **network**. Hence, a network is a special type of a graph.

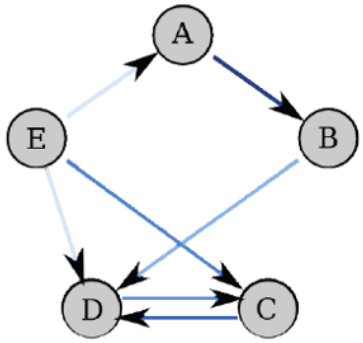


Fig. 1. A node-link diagram of a small graph in a radial/circular layout consisting of 5 vertices and 7 weighted and directed edges

Visualization can be of great help for getting an overview about this kind of data. Moreover, interaction techniques are important to let awake the otherwise static diagrams and to let the users or data analysts manipulate the views until they found insights or knowledge in them. For this reason it is important to also provide several views/perspectives with different layouts, edge repreentation styles, and matrix orderings that are interactively linked. It is also important to provide the visualization tool in a web-based version to allow access for a larger population of researchers from all over the world.

Your project consists of several steps. You should design your own visualization tool (be creative). This tool should show at least two visual metaphors of

the graph dataset, a node-link diagram and an adjacency matrix. For the node-link diagram you should provide at least three different layouts (for example, radial/circular [2], force-directed [5, 12], and hierarchical [19]) while your adjacency matrix should support at least 5 different reordering strategies [1]. From each of the 7 categories of interactions [22] you should have implemented at least one interaction technique. Finally, your tool should be web-based, allowing a user to just type in a URL in the browser, upload graph data (in a specified data format), and visually interact with this data. The data as well as insights from the data should be sharable with all other people using the web tool.

2 Graph Data and Data Format

Graph data is a specific kind of data that consists, in its simplest form, of vertices and edges, maybe carrying weights [4]. The vertices might be labeled, i.e. carrying a textual description. Mathematically, this can be modelled as

$$G = (V, E)$$

where V is the finite set of vertices and E is the finite set of edges. Generally, there are various forms of graphs like directed, undirected, weighted, or unweighted graphs. But also more complex types like Euler graphs, bipartite graphs, multivariate graphs, hyper graphs, multigraphs and so on are possible. However, to keep things simple, we will focus in this project on **directed and weighted graphs, i.e., networks!**

The data format that we use in this project is a matrix-based scheme (see Figure 2) with the vertex labels in the first row. Have a look into the Files folder in Canvas to get an impression about the data format we use. There we provided co-author networks to support you with your first attempts to parse and read network data. Try to get familiar with our data format and ask questions (if any) immediately. The data is the core ingredient in a visualization tool. You can change this data format later on if you like but your tool should at least be able and guarantee to import network data given in this specific format. This means if someone uploads a new dataset in this specific data format, your tool should be able to read it and generate the graph visualizations you support. If someone uploads data in another format, your tool does not guarantee to work properly, maybe it will not work at all.

The special advantage of this data format is that it already comes in some form of 2D array with the extra first line indicating the textual descriptions of the vertices. 2D arrays are best for accessing network data because they allow an access time of $O(1)$, i.e. only a constant number of time steps is required to get the data from the memory, making it of low runtime complexity. However, the layout algorithms for the node-link diagrams and the reordering strategies for the adjacency matrices have varying runtime complexities, from linear, squaric, cubic, to exponential runtimes. By aware of the fact that your tool can run into such runtime problems if you have to deal with large networks. Moreover, if the

Fig. 2. Here you see a small snippet from the text file that we provided. It shows you a part of the first line containing the labels (ordered by occurrence) for the vertices and the whole first line of edge weights (as real-valued numbers). Also a part of the second line is shown. The vertex labels and the edge weights are separated by semicolons.

algorithms are running slow, also your interactions will suffer from this, making your tool unusable.

If you face problems and challenges with runtime complexities, you can do two things: Either you try your best with finding, generating, or building advanced and fast running algorithms or you reduce the amount of data in the given datasets by just cutting off a part of the data. Actually, you could pre-filter the 2D array into a smaller 2D array. A better solution would be to aggregate the given network by building vertex groups and aggregate the corresponding edge weights. These so called super vertices build a much smaller network, but it may be noted that you always loose information if you aggregate a dataset. Moreover, with this strategy it remains unclear which vertices to group together. However, one goal of this DBL (apart from collaboration) is to face such data problems and try to discuss and find out a suitable solution. From a visual perspective, Figure 3 gives example node-link diagrams that can be found online.

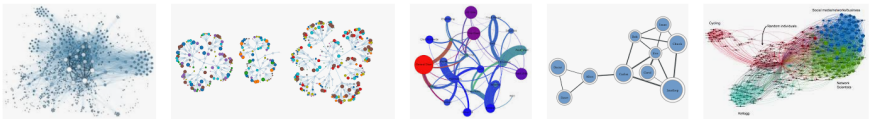


Fig. 3. A repertoire of graph visualizations if Google image search is used.

3 A Simple Application Scenario

Imagine, someone living in the US, Japan, or Brazil owns a network dataset that is pretty large in terms of the number of elements and relations between those elements. Maybe someone is interested in his/her Facebook network or a software engineer plans to visually inspect the call relations in the software system he/she is involved. Such a visualization tool could be helpful to find anomalies in the data, but also groups and clusters might be detectable which would be impossible by just scrolling through tons of textual data given in a text file or stored in a data base.

A more visual representation might be desired that has perceptual benefits since it exploits the strengths of the human's visual system for pattern recognition [20, 21] and can hence, support to find these patterns in the data more rapidly. Visualization is a great tool for data analysis, assumed it is designed in an effective and efficient way. Typically, several visualization techniques for the same data type exist, all having benefits and drawbacks. Consequently, it can be a useful strategy to design a so called multiple coordinated view [17] that shows the same dataset in as many visualization techniques as desired by the user.

To fully exploit the strengths of those multiple coordinated views, interaction techniques [22] are required that, for example, allow brushing and linking features, i.e., selecting and highlighting graphical elements in one visualization immediately forces all other visible visualizations to highlight the same graphical elements. This becomes one of the most important interactions when several visualizations are used in combination applied to the same dataset.

For sure, there are various interaction techniques, trivial and simple ones, but also complex and difficult to implement ones. In graph visualizations, it is useful to select groups and clusters or outliers, to change color codings, layouts, link representations, or reorderings in the adjacency matrices, just to mention a few. Other very useful interaction techniques are details-on-demand. If someone hovers or clicks a graphical element, a textual information, for example, a (currently not visible) label is shown. On the negative side, showing all labels at the same time can be problematic for large network data since then we run into a problem referred to as visual clutter [18] that is "a state in which excess items or their disorganization lead to a degradation of performance at some task." This does not only happen for too many labels, but it is a general problem for node-link diagrams.

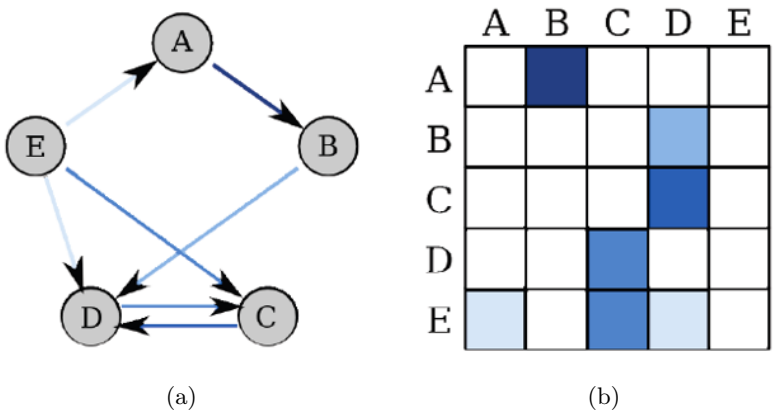
Another important interaction feature, in particular in visualization, would be to allow screenshots of a certain kind of view. This means if a data analyst found a certain kind of interesting visual pattern he/she might wish to store this snapshot of the visualization, for example, to show it in a presentation or to distribute the snapshot and share it with other data analysts. If such a screenshot feature is supported, the corresponding file of this figure should contain all parameters in its current configuration. This is important if the figure was used in a presentation later on, to remember all the parameters or to rebuild it again, in case a small modification has to be made afterwards. There are really

many interaction techniques, so many, that we cannot expect to implement all of them, but the most important ones are desirable.

To keep things simple we restrict the data uploaders to the specific data format having the labels in the first text line in the same order as the vertices and the relation weights appear in the followig lines. This 2D array format is the most popular and simplest network data format in use. Various libraries exist to parse and load such data into a tool.

4 Graph Visualizations and Layouts

In general, there are two major visual metaphors for graph data: node-link diagrams and adjacency matrices (see Figure 4). However, there are some more like adjacency lists or hybrid representations build from two or more of those. But to keep things simple, these are not the major focus of this project which does not mean that you are not allowed to use them in your tool.



(a) (b)

Fig. 4. The two major visual metaphors for network data: (a) A node-link diagram. (b) The same network as an adjacency matrix, look at the color coding of the edges that indicates the weights.

4.1 Node-Link Diagrams

Node-link diagrams (see Figure 4) (a) have been invented many years ago by Leonard Euler when trying to find an abstraction to the problem of 'The Seven Bridges of Königsberg'(see Figure 5). In these ancient times, node-link diagrams were powerful visualizations for graph data, but nowadays they come with several problems related to visual clutter [18] caused by link crossings and overdraw.

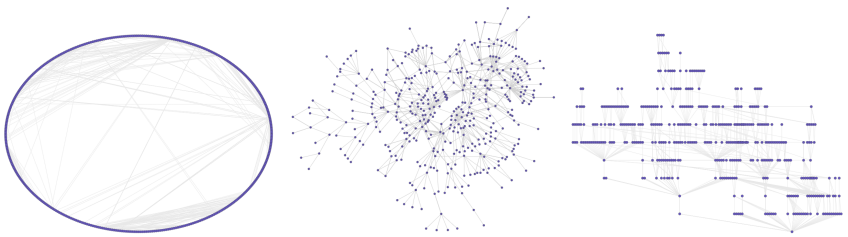
Layout algorithms are responsible for producing nicely looking node-link diagrams following certain aesthetic graph drawing criteria [16, 14, 15]. The most

Fig. 5. Leonard Euler found an abstraction for the 'Seven Bridges of Königsberg', a visual metaphor that we denote as node-link diagram today.

famous layouts are radial/circular [2], force-directed [5, 12], and hierarchical ones [19]. But even if the most advanced layout is chosen, we typically run into visual problems denoted by hairball effects. This means the node-link diagram contains so much overdraw that it is not readable anymore. Path-related tasks (i.e. how can we walk from a node A to a node B) become visually unsolvable, meaning interactions are the only solution to that, but if the graph is too dense it is better to start with an ordered adjacency matrix to get an overview first.

If such a problem in node-link diagrams occurs, researchers also typically try to change the link representation styles [11, 10] or draw links partially [3]. Moreover, edge bundling [9] might be a good solution to visually show some structure in the network data. Edge bundling is pretty fancy and algorithmically not that easy to build, however, if you feel experienced enough, have some time left, and enjoy this visualization course, there is no limitation to integrate edge bundling in your tool. Feel free to build as many features as you like, but to pass the course you need at least the minimum requirements that are given in Section 1.

However, if the graph data is pretty dense (containing many edges) and if the number of vertices exceeds a certain size we should better switch to an adjacency matrix visualization [6]. Moreover, as an interaction feature it might be useful if regions from an adjacency matrix can be selected and then from this region a node-link diagram is generated.



(a) (b) (c)

Fig. 6. Three different layouts for node-link diagrams: Radial/circular (a), force-directed (b), and hierarchical (c).

4.2 Adjacency Matrices

Figure 4 (b) shows an adjacency matrix of a very small network example dataset. The color coded matrix cells indicate the edge weights. The matrix is always read from left to top to interpret the links, meaning the start vertex is always in the corresponding row while the target vertex is in the corresponding column.

The great benefit of adjacency matrices is that they have no overdraw, link crossings, or visual clutter effects, but if they are unordered they are not useful to depict graph structures (see Figure 7). Moreover, path-related tasks cannot be solved in adjacency matrices. For path-related tasks there is nothing better than a node-link diagram if the network has less than 20 vertices [6]. This has to do with Gestalt principles and the fact that the humans' visual system is trained to interpret and follow lines very quickly. However, adjacency matrices serve as a great overview due to the fact that they can be scaled down to pixel size, a fact that we denote by visual scalability (see Figure 8).

But again, without an advanced ordering they are pretty useless. Read the work by Behrisch et al. [1] to get an impression about useful matrix reordering strategies and for which task they are suited best.

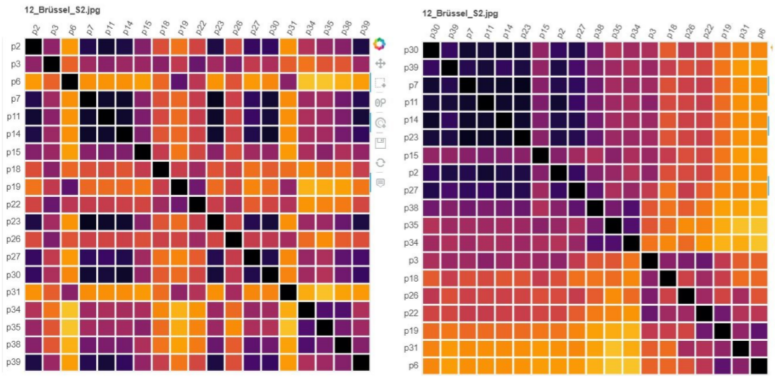


Fig. 7. Ordering the original adjacency matrix (left) by an advanced ordering strategy can reveal some insights on clusters, groups, but also anomalies (b).

4.3 Hybrid Representations

There are also options to combine two or more visualization techniques. These are then called hybrid visualizations. One famous example of a hybrid representation for networks is the so-called NodeTrix [8] originating from a combination of the two terms node-link and matrix (see Figure 9). Another example is MatLink [7], also originating from two terms namely matrix and node-link (see Figure 10).

It is allowed to build new visualization techniques by such hybrid representations. But be careful, there are really many ways for combinations in information

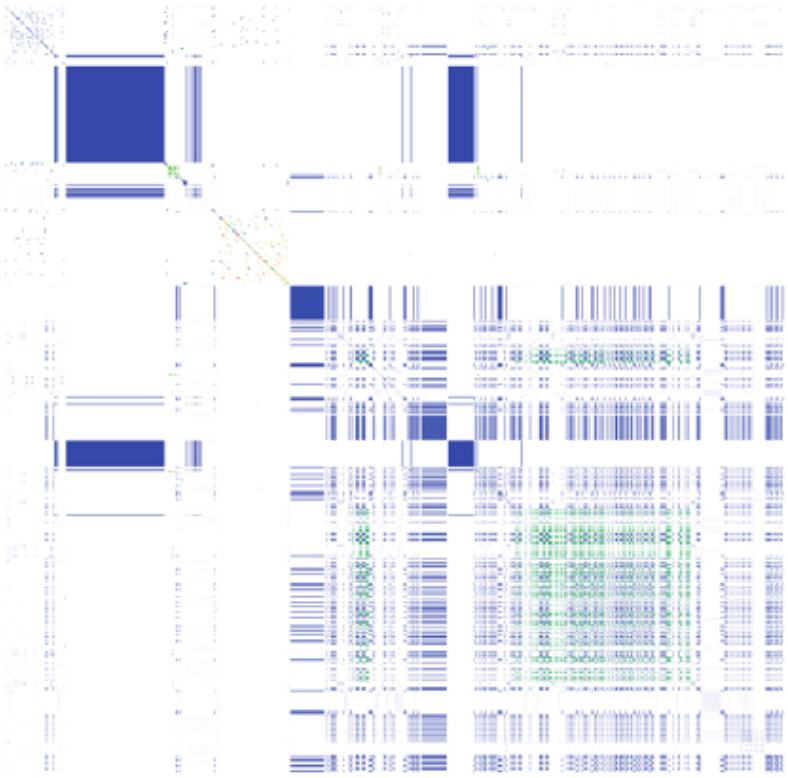


Fig. 8. Weighted relations between files on a computer: 36,856 files are analyzed for relations and then visualized as an adjacency matrix. It may be noted that the pairwise file comparisons result in more than 1,350,000,000 weighted relations.

visualization. You soon get lost in the large repertoire. The best strategy for your project is to pick out some candidates right from the beginning and then extend those. Only if you have time left and feel enjoyed by creating visualizations you should do more.

4.4 Textual Descriptions/Labels

Labels for the vertices are important for readability and understandability of the graph visualization. But be careful, if they are not implemented in a proper way they cause extra visual clutter. If you plan to solve this problem we recommend to have a look at navigation systems in cars and get inspired about the fact how they place labels and how many they place in which display region. Not all labels can be placed at the same time, but for contextual information you should depict the most important ones.

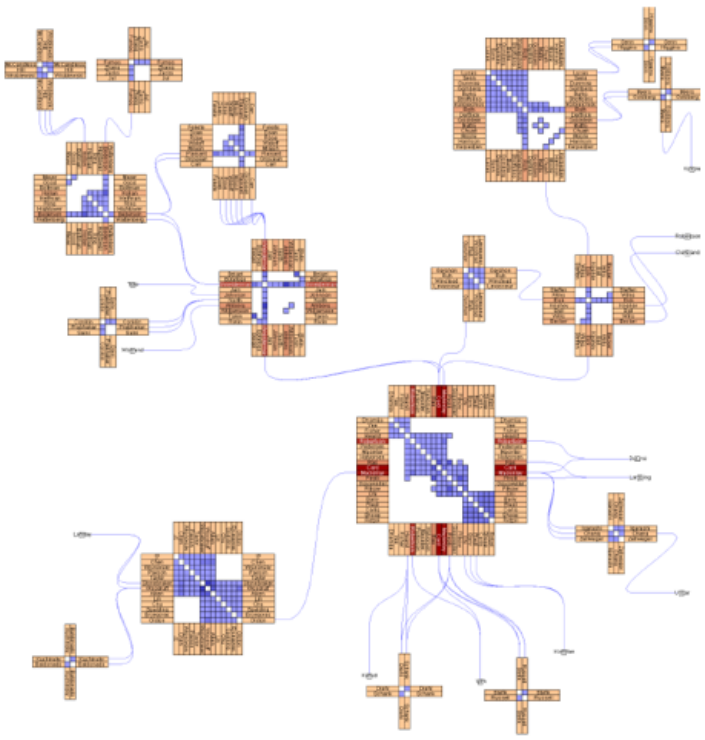


Fig. 9. An example of a NodeTrix visualization.

5 Interactions

Selecting a region in the matrix and showing this as node-link diagram (or highlight all selected elements in the node-link diagrams) and vice versa is a required interaction and can support a data analyst a lot. Moreover, clicking on visual elements and selecting them, getting extra information in form of tooltips, filtering, aggregation, color coding, change representations, and the like are some candidates of an endless list of interaction techniques.

There are lots of interactions in information visualization. You can have a look into the paper of Yi et al. [22] to see which interaction techniques are possible. The paper easily describes these techniques and tries to give many examples for interactions in several application fields. For those of you who cannot imagine what important interaction techniques are, I would strongly recommend to read the paper by Yi et al. It is necessary in your project that you implement at least one candidate from each of the 7 categories of interactions described in the paper.

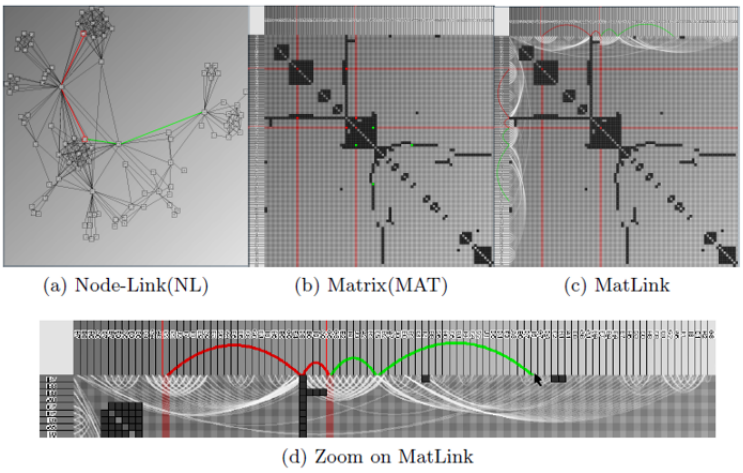


Fig. 10. An example of a MatLink visualization.

6 Programming Environment and Languages

The very first task in order to get started with the project is to discuss in your group which kind of programming language/languages you require to solve the given visualization problem, i.e., to provide a solution to the given simple scenario. Find a consensus between the student group members and also discuss with your tutor about your ideas. Do not forget to send the output of your discussions to the tutor and the group members.

There are lots of possibilities to reach the solution. You could, for example, use PHP as a server-side scripting language to allow for file uploads and to use it just like a standard programming language to transform the data into a graphical representation. Interaction is a bit tricky with PHP, hence I recommend to read a tutorial about this first

https://www.codeproject.com/Articles/604542/

Creating-Interactive-HTML-Graphs-in-PHP

or to watch a YouTube video about phpChart - a fully featured AJAX control for generating dynamic charts in PHP.

This is important to understand the general problems that developers of web-based visualizations have. It also shows you how to combine JavaScript into the implementation process. Chart.js is also one way to come to a solution. In general, JavaScript provides useful packages and libraries (like D3.js or Chart.js) for interactive graphics

https://thenextweb.com/dd/2015/06/12/20-best-javascript-chart-libraries/,

but also JAVA with its graphics libraries, Python and many other programming languages are useful to solve the visualization task.

In particular, Python could be your choice if you plan to use Bokeh that is a data visualization library that allows a developer to code in Python and output JavaScript charts and visuals in web browsers. This is actually what I would prefer. To start understanding this concept you can read the content on this webpage

https://www.fullstackpython.com/bokeh.html

or the user guide

https://bokeh.pydata.org/en/latest/docs/userguide/plotting.html

but you should find tutorials on the web to get more experience with the library and how it works in a browser.

As mentioned above, there are many opportunities, and one goal of this collaborative course is to develop **YOUR OWN** solution during the course. You have all the freedom to implement the tool, the only restriction is that the tool should be runnable in the end and the required features are integrated. You can add as many features as you like in case you really enjoy the project and developing visualization techniques. You can share your project with eyerybody since it is a web-based visualization that simply runs in a browser. The major goal of this project is the **COLLABORATION** with other students and to come to a common goal in the end. However, do not forget the **Best Project Award, Best Presentation Award, Scalability Award, and Aesthetics Award** in the final project presentation meeting!

In general, we recommend to read tutorials about web programming, in particular to bring information into a browser. Moreover, YouTube is a useful platform to find tutorials, in particular, for visualization techniques, interactions, and implementation details. Please discuss in your group how you solve this (probably most difficult) task in the beginning of the DBL course.

7 Suggested Working Plan and Initial Steps

We recommend to start the project with discussing the general problems in the course. From a programming perspective, you should discuss the programming languages that you require and the roles of the individual group members. Although we may hint you at certain programming options, you do not have to follow these advices in case you are an experienced programmer. The programming languages, libraries, or APIs we recommend are just to hint you at certain alternatives although they might not be the right choice for the tool you plan to build.

From a visualization perspective, you could also design your own network visualizations and you can implement as many visualization techniques and interaction features as you like. In some cases, people feel very enthusiastic about

a certain topic, that they wish to break free from the general rules and do a bit more. Feel free to do that, but be aware of the minimalistic repertoire of techniques and features that are required to pass the course.

https://canvas.tue.nl/courses/6672

References

1. Behrisch, M., Bach, B., Riche, N.H., Schreck, T., Fekete, J.: Matrix reordering methods for table and network visualization. Computer Graphics Forum 35(3), 693–716 (2016)
2. Brandes, U., Pich, C.: More flexible radial layout. Jornal on Graph Algorithms and Applications 15(1), 157–173 (2011)
3. Burch, M., Vehlow, C., Konevtsova, N., Weiskopf, D.: Evaluating partially drawn links for directed graph edges. In: Proceedings of the Symposium on Graph Drawing. pp. 226–237 (2011)
4. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, Upper Saddle River, NJ (1999)
5. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Software: Practice and Experience 21(11), 1129–1164 (1991)
6. Ghoniem, M., Fekete, J.D., Castagliola, P.: A comparison of the readability of graphs using node-link and matrix-based representations. In: Proc. IEEE Symposium on Information Visualization. pp. 17–24 (2004)
7. Henry, N., Fekete, J.: Matlink: Enhanced matrix visualization for analyzing social networks. In: Proceedings of Human-Computer Interaction - INTERACT. pp. 288–302 (2007)
8. Henry, N., Fekete, J., McGuffin, M.J.: Nodetrix: a hybrid visualization of social networks. IEEE Transactions on Visualization and Computer Graphics 13(6), 1302–1309 (2007)
9. Holten, D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. IEEE Transactions on Visualization and Computer Graphics 12(5), 741–748 (2006)
10. Holten, D., Isenberg, P., van Wijk, J.J., Fekete, J.D.: An extended evaluation of the readability of tapered, animated, and textured directed-edge representations in node-link graphs. In: Proc. IEEE Pacific Visualization Symposium. pp. 195–202 (2011)
11. Holten, D., van Wijk, J.J.: A user study on visualizing directed edges in graphs. In: Proc. SIGCHI Conference on Human Factors in Computing Systems. pp. 2299–2308 (2009)
12. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Information Processing Letters 31(1), 7–15 (1989)
13. von Landesberger, T., Kuijper, A., Schreck, T., Kohlhammer, J., van Wijk, J.J., Fekete, J., Fellner, D.W.: Visual analysis of large graphs: State-of-the-art and future research challenges. Computer Graphics Forum 30(6), 1719–1749 (2011)
14. Purchase, H.C.: Which aesthetic has the greatest effect on human understanding? In: Proc. Symposium on Graph Drawing. pp. 248–261 (1997)
15. Purchase, H.C., Carrington, D., Allder, J.A.: Empirical evaluation of aesthetics-based graph layout. Empirical Software Engineering 7(3), 233–255 (2002)

16. Purchase, H.C., Cohen, R.F., James, M.: Validating graph drawing aesthetics. In: Proc. Symposium on Graph Drawing. pp. 435–446 (1996)
17. Roberts, J.C.: State of the art: Coordinated & multiple views in exploratory visualization. In: Proceedings of Fifth International Conference on CMV. pp. 61–71 (2007)
18. Rosenholtz, R., Li, Y., Mansfield, J., Jin, Z.: Feature congestion: a measure of display clutter. In: Proc. SIGCHI Conference on Human Factors in Computing Systems. pp. 761–770 (2005)
19. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics 11(2), 109–125 (1981)
20. Ware, C.: Information Visualization: Perception for Design. Morgan Kaufmann (2004)
21. Ware, C.: Visual Thinking: for Design. Morgan Kaufmann Series in Interactive Technologies, Paperback (2008)
22. Yi, J.S., ah Kang, Y., Stasko, J.T., Jacko, J.A.: Toward a deeper understanding of the role of interaction in information visualization. IEEE Transactions on Visualization and Computer Graphics 13(6), 1224–1231 (2007)