Int J Software Informatics, Vol.?, No.?, ??? 2015, pp. 1–**??** E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
©2015 by Institute of Software, Chinese Academy of Sciences. All rights reserved. Tel: +86-10-62661040

Flip-Book Visualization of Dynamic Graphs

Michael Burch, Daniel Weiskopf

(VISUS, University of Stuttgart, Germany)

Abstract Dynamic graph visualization techniques can be based on animated or static diagrams showing the evolution over time. In this paper, we apply the concept of small multiples to visually illustrate the dynamics of a graph. Node-link, adjacency matrix, and adjacency list visualizations are used as basic visual metaphors for displaying individual graphs of the sequence. For node-link diagrams, we apply edge splatting to improve readability and reduce visual clutter caused by overlaps and link crossings. Additionally, to obtain a more scalable dynamic graph visualization in the time dimension, we integrate an interactive Rapid Serial Visual Presentation (RSVP) feature to rapidly flip between the sequences of displayed graphs, similar to the concept of flipping a book's pages. Our visualization tool supports the focus-and-context design principle by providing an overview of a longer time sequence as small multiples in a grid while also showing a graph in focus as a large single representation in a zoomed in and more detailed view. The usefulness of the technique is illustrated in two case studies investigating a dynamic directed call graph and an evolving social network that consists of more than 1,000 undirected graphs.

Key words: Dynamic graph visualization, rapid serial visual presentation, flip-book

M Burch, D Weiskopf. Flip-Book Visualization of Dynamic Graphs. *Int J Software Informatics*, 2015, ?(?): 1–**??**. **http://www.ijsi.org/1673-7288/3/1.pdf**

1 Introduction

Quickly flipping a book's pages is a simple way to get a first impression of a book's content. We were inspired by a book-flipping technique described by de Bruijn and Spence [14] based on the concept of Rapid Serial Visual Presentation (RSVP) [27]. Although RSVP was originally applied as a reading aid, it can easily be extended to a concept for quickly presenting a sequence of visualizations, in particular, representations of time-oriented data [1] that can be displayed as a sequence of individual snapshots.

Dynamic graph visualization is a field that benefits from this concept. Exploration scenarios such as the questions "What is there?" or "Is it here?" are important in time-varying data representations to get a quick overview of the whole dataset or rapidly search for anomalies or outliers. These two questions can be answered by interaction techniques like browsing or weighted browsing (if one knows what to search for).

In this paper, we map dynamic relational data to a dynamic graph visualization based on node-link diagrams visually enhanced by edge splatting [11], on adjacency

† This work is sponsored by DFG SFB 716/D.5.
‡ Corresponding author: Michael Burch, Email: michael.burch@visus.uni-stuttgart.de
♯ Manuscript received 2008-06-19; revised 2008-06-21; accepted 2008-06-23; published online 2008-08-25

matrices, as well as adjacency lists [20]. The node-link diagrams are shown in a pre-computed and fixed graph layout and are placed as small multiples [30, 31] in a 2D grid. Also the adjacency matrices and adjacency list representations are always displayed with the same vertex order to preserve a viewer's mental map. We use a left-to-right and top-to-bottom reading direction in the grid. To achieve good visual scalability we map the graphs to as many grids as needed to display the whole dataset.

To allow good comparisons of graphs in longer graph sequences on an overview-based design while still supporting details of a single graph, we support a focus-and-context visualization. Moreover, if an analyst is more interested in additional graph properties such as solutions to graph-theoretic problems we support an interactive way to have a look at the dynamics of shortest paths for example.

This article is an extended version of a previous paper [13] in which we address the issue of visually exploring long graph sequences. Compared to our previous work, we added visual metaphors for adjacency matrices, adjacency lists, and we now support hybrid small multiples which is a combination of several individual metaphors in the graph sequence. Moreover, a focus-and-context design is integrated showing small multiples together with large singles in order to see the dynamic graph at small and large scale. The visualization can also be used to highlight the dynamics of shortest paths in any of the visual metaphors.

Overall, we extend existing dynamic graph visualization by five contributions:

- **Modifiable grid for small multiples:** The graphs of a longer sequence are rendered on a modifiable grid similar to a small multiples representation. The user can interactively change the row and column number in the grid which shows the graphs at different scales. In contrast, our work on parallel edge splatting [11] displays the graph sequence in a 1D row only, which limits the number of displayable graphs. The grid-based technique is applicable to all visual metaphors, i.e. node-link diagrams, adjacency matrices, and adjacency lists.
- **Flip-book feature:** To further improve the scalability in the time dimension we use a flip-book feature. This is an extension to the RSVP visualization in which a graph subsequence is shown in a comic strip-like representation and gets smoothly animated [4]. The flip-book approach makes use of the perceptual ability of the human visual system to rapidly recognize time-varying visual patterns.
- **Focus-and-context design:** We display longer graph sequences in an overview-based representation, but the resulting visualization might suffer from the small multiples representation. In particular, when many graphs are shown side-by-side on a grid, details are typically hidden. To mitigate this situation, we also show large singles (focus), in addition to the small multiples (context), i.e., graphs of the sequence can be shown at small and large scale.
- **Interactive visual metaphors exchange:** Since the visualization tool supports graph visualizations with the major visual metaphors, the user can interactively decide if the graph data is visually encoded as node-link diagrams, adjacency matrices, or adjacency lists. Depending on the graph properties, one can do this individually for each graph in the sequence or for all of them at once.
- **Dynamics of shortest paths:** Typically, a dynamic graph visualization can unhide interesting time-varying graph patterns. However, in many scenarios, the

graph data is too dense to explore it for certain aspects over time. This is in particular the case when an analyst is interested in the time-dependent behavior of shortest paths while simultaneously seeing the complete graph sequence.

Flipping the modifiable grids quickly on user's demand is used as key interaction principle in this paper, similar to RSVP, to manage the visualization of longer graph sequences, which is the major focus of this work. We illustrate the usefulness of this flip-book concept by applying it to a dynamic call graph from software development and to a dynamic social network consisting of more than 1,000 graphs.

2 Related Work

There are many application domains dealing with dynamic relational data. Social networks, call graphs between functions in software systems, or protein-protein interactions in bioinformatics are examples of this kind of data.

There are two basic approaches in dynamic graph visualization: time-to-time mapping as it is used in animation is one way to show the time dependency, whereas time-to-space mapping is another way to visually depict time-varying graphs [3]. Several comparative studies focus on the question which dynamic graph visualizations lead to better user performance [2, 18, 32]. Also, hybrid visualizations making use of difference maps and animations were designed and evaluated [24].

When animating a graph, a node-link diagram is generally laid out and smoothly transformed into a sequence of layouts. This process demands for a good layout for both each single graph in the sequence and the whole sequence in order to preserve the viewer's mental map [23], obtained by a high degree of dynamic stability. Offline [15] and online [16] approaches are investigated for their suitability to represent dynamic graphs, depending on whether the graph sequence is known beforehand or graphs have to be added on-the-fly.

Animation has some general drawbacks apart from high algorithmic complexity. The viewer can only see one graph at a time, leading to problems when comparing several graphs in the sequence to identify time-varying visual patterns such as trends, countertrends, or anomalies. For this reason, time-to-space mappings [5, 7, 28] were developed that present a subsequence of the evolving graph in one view. This concept allows users to visually analyze a dynamic graph by having a look at all the graphs side by side in a small multiples representation [30, 31].

One drawback of a small multiples diagram is its poor scalability with time. For example, in the parallel edge splatting technique [11], only one representation row is used for a graph sequence, which can be enhanced by applying the concept of RSVP [4]. However, in this RSVP variant, the graph sequence is animated and only one row containing a subsequence of graphs is displayed.

As an enhancement in our work, instead, we do not smoothly animate the graph subsequence by moving this sliding time window, but we propose an approach that places a graph subsequence on a rectangular grid and allows users to flip these grids just as simulating the flipping of a book's pages. This concept allows users to see several graphs in one image before flipping to the next grid [13].

Moreover, we support another graph layout apart from the bipartite one used by Burch et al. [11]. We exploit a radial graph layout, but also hierarchical [29] or force-directed layouts [17, 21] could easily be integrated in our tool by just extending the set of layout methods by new source code implementations in the corresponding class. We also support matrix representations, which have benefits for dense graphs [19, 25] but problems when solving path-related tasks. Also, adjacency lists [20] are integrated in the tool, showing dense graphs in a compact and space-filling way.

In a traditionally animated diagram, only one graph at a time is shown on the entire screen, whereas in a time-to-space mapping, several graphs of the sequence have to be displayed, limiting the display space for each single graph. This demands for an alternative visualization strategy, i.e., some kind of hybrid representation benefiting from both concepts: animation and static displays. Our method utilizes this combination of concepts.

In a recent paper, van den Elzen and van Wijk [33] illustrate the usefulness of small multiples compared to what they call large single representations. In our work, we are also able to show large singles with a details-on-demand function, but we focus on showing time-varying patterns for which a small multiples representation is required to first give an overview of the dataset [26] and then to visually compare the individual time steps.

3 Data Model

We model a directed weighted graph mathematically as

$$G = (V, E)\,,$$

where

$$V := \{v_1, \ldots, v_n\}$$

denotes the set of $n \in \mathbb{N}$ vertices and

$$E \subseteq V \times V$$

the set of edges. Each edge $e \in E$ has a weight

$$w(e) \in \mathbb{R}$$

given by a weight function

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

In the context of this work, a layout L of a graph G is a function that maps vertices to positions on a display by

$$v_i \mapsto (x_i, y_i)\,.$$

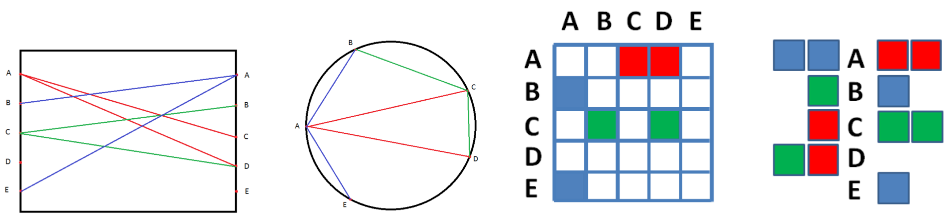
Edges are represented as links between two related vertices in the node-link metaphor and as color-coded squared graphical primitives in the adjacency matrix and list metaphors.

A dynamic graph

$$\Gamma := (G_1, \ldots, G_k)$$

consists of a sequence of $k \in \mathbb{N}$ graphs. We define the union of all graphs as $G_{\bigcup} := (V_{\bigcup}, E_{\bigcup})$ with

$$V_{\bigcup} = \bigcup_{i=1}^{k} V_i$$



(a) (b) (c) (d)

Fig. 1. Single node-link graph layouts: (a) A bipartite graph layout in a left-to-right reading direction. The vertex set is first copied and then the original edges point from left to right. (b) A radial layout that maps vertices to the circle circumference; the same graph is depicted as in (a) but the edge directions are not explicitly shown. Further visual metaphors for graph data: (c) An adjacency matrix maps vertices to rows and columns and shows graph edges as color-coded matrix cells at the intersection points. (d) An adjacency list removes the empty cells from a matrix by horizontally stacking the color-coded cells. Incoming and outgoing edges can be split at the vertical vertex line.

and

$$E_{\bigcup} = \bigcup_{i=1}^{k} E_i\,,$$

with $G_i = (V_i, E_i)\ \forall\, 1 \leq i \leq k$.

The sum of a graph subsequence

$$\Gamma_{l-m} := (G_l, \ldots, G_m)$$

from index l to index m is defined as

$$G_{l-m} := (V_{\bigcup}, E_{l-m})\,,$$

where

$$E_{l-m} := \bigcup_{i=l}^{m} E_i\,.$$

The weight of each $e \in E_{l-m}$ depends on the aggregation mode.

4 Visualization Technique

We describe a technique for representing dynamic graphs in a small multiples visualization. For each graph, we exploit the node-link, adjacency matrix, or adjacency list visual metaphors.

4.1 Individual Graph

Before rendering a sequence of graphs in a small multiples representation, we have to decide how to represent each individual static graph G_i, i.e., in which specific layout. The technique can work with two different node-link graph layouts for individual static graphs: bipartite and radial layouts, see Figure 1 (a) and (b). Our visualization technique also supports adjacency matrix and adjacency list layouts, for which we use color-coded squared graphical primitives, see Figure 1 (c) and (d). The adjacency matrix uses color-coded matrix entries for expressing the corresponding weights of the directed graph edges. For the list representation, we visually encode outgoing edges

Algorithm 1 Generation of a Bipartite Graph Layout

```
**BipartiteLayout**$(G = (V, E), c)$:
  $V := \{v_1, \ldots, v_n\}$; // Vertex set
  $E := \{e_1, \ldots, e_{|E|}\}$; // Edge set
  $c$; // Colorscale

  $w$; // Weight function
  $x_{left}$; // Left side axis
  $x_{right}$; // Right side axis
  $h$; // Height of both axes

  **for all** $e \in E$ **do**
    $(v_i, v_j) := e$;
    $color(w(e), c)$; // Color code link
    $drawline(x_{left}, \frac{i}{n} \cdot h, x_{right}, \frac{j}{n} \cdot h)$; // Graph link
  **end for**
```

to the right in a stacked fashion and consequently, incoming edges to the left by using color-coded squares similar to the adjacency matrix.

For all visual metaphors, we first compute the union of all graph vertices $V_{\bigcup}$ occurring in the whole sequence Γ, similar to our previous work [11]. This vertex union is needed to support dynamic stability in order to preserve a viewer's mental map, which is important for the visualization technique.

The next step for the bipartite node-link diagram is to generate a copy of the vertex set $V_{\bigcup}$ denoted by $V'_{\bigcup}$. The visual mapping of a single graph G_i from the sequence Γ is computed by first mapping the vertices of both sets $V_{\bigcup}$ and $V'_{\bigcup}$ equidistantly on two parallel vertical lines (see Figure 1 (a)) in a way that each vertex copy lies on the same horizontal line cutting through each of the vertical axes.

Then, the edges of the laid out graph G_i are processed by plotting them from left to right as straight lines between their corresponding vertex positions. This layout produces a node-link graph diagram in a left-to-right reading direction, i.e., there are only links between the vertical axes, but no links are present between vertices located on one single axis. This visualization strategy results in a bipartite graph layout, see Algorithm 1.

For the radial layout, we also first compute the union set of all vertices occurring in the whole graph sequence Γ. Then, a single graph G_i is depicted by equidistantly mapping all the vertices to the circle circumference. Edges in a graph G_i are visualized by straight links connecting the corresponding vertices on the circle circumference. This strategy results in a graph layout represented in Figure 1 (b). Algorithm 2 illustrates the layout generation process.

It may be noted that in a radial layout, we cannot easily derive the direction of an edge without explicitly attaching arrow heads pointing to the target node, but in such a layout we only have one representative node for each vertex. Consequently, path-related tasks are easier to solve for non-directed graphs in a radial layout because the eye does not have to jump to and fro between start and target nodes when tracing a path.

Algorithm 2 Generation of a Radial Graph Layout

```
**RadialLayout**$(G = (V, E), c)$:
  $V := \{v_1, \ldots, v_n\}$; // Vertex set
  $E := \{e_1, \ldots, e_{|E|}\}$; // Edge set
  $c$; // Colorscale

  $w$; // Weight function
  $r$; // Radius of circle

  for all $e \in E$ do
    $(v_i, v_j) := e$;
    $color(w(e), c)$; // Color code link
    $drawline(r \cdot \sin(\frac{i2\pi}{n}), r \cdot \cos(\frac{i2\pi}{n}),$
              $r \cdot \sin(\frac{j2\pi}{n}), r \cdot \cos(\frac{j2\pi}{n}))$; // Graph link
  end for
```

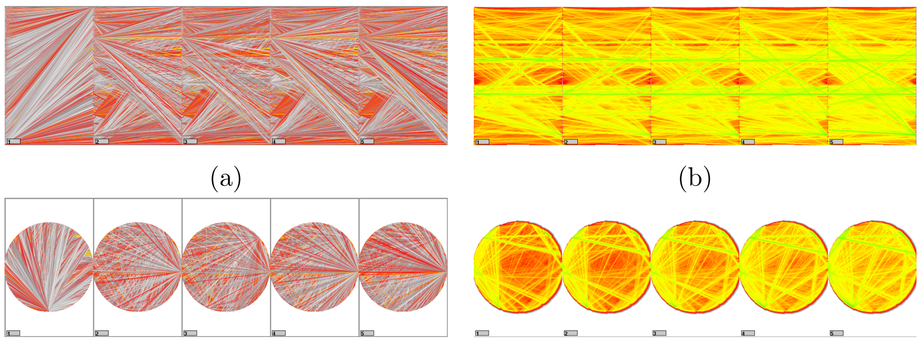

(a) (b) (c) (d)

Fig. 2. A node-link diagram in a bipartite graph layout without (a) and with (b) edge splatting. A node-link diagram in a radial graph layout without (c) and with (d) edge splatting.

If a graph visualization has to be used efficiently for solving path-related tasks, node-link diagrams are preferable compared to adjacency matrices or lists [19]. But, on the positive side, matrix and list representations benefit from a high degree of visual scalability. Since link crossings are avoided, the viewer can easily explore the dynamic graph for evolving cluster regions, which can be a challenging task for node-link diagrams. For this reason, we also integrated matrix and list visual metaphors in our visualization tool to allow the visual analysis of evolving dense graphs.

4.2 Edge Splatting

When drawing dense graphs by node-link diagrams, we soon reach a situation where many link crossings occur. In the bipartite and radial layouts where graph vertices are only allowed to be placed on one-dimensional lines, the probability for such link crossings increases rapidly with the number of vertices and edges.

To address this problem we apply edge splatting [11]. In a splatted node-link diagram, the links are rasterized into a sequence of weighted pixels (depending on the edge weights); weights for the pixels are accumulated by incrementally processing all graph links. This finally generates a matrix (i.e., image) consisting of edge density

Algorithm 3 Generation of a Splatted Graph

```
**Splatting**($L, G = (V, E), c$):
  $L$; // Layout of graph G
  $V := \{v_1, \ldots, v_n\}$; // Vertex set
  $E := \{e_1, \ldots, e_{|E|}\}$; // Edge set
  $c$; // Colorscale

  $A$; // Matrix for density field
  $w$; // Weight function

  for all $e \in E$ do
    $(v_i, v_j) := e$;
    $A+ = rasterize\_line(e, w(e))$; // Map link to $A$
  end for
  $boxfilter(A)$; // Smoothing
  for all $0 \leq i \leq A.length - 1$ do
    for all $0 \leq j \leq A[0].length - 1$ do
      $color(A[i][j], c)$;
      $drawPixel(i, j)$;
    end for
  end for
```

values, which can again be used to display the edge coverage by a density field visualized by a user-defined color scale. The box filter is applied to obtain smoother density fields and make the diagrams more aesthetically appealing. The generation process for edge splatting is illustrated in Algorithm 3. Figure 2 shows the differences between splatted and non-splatted node-link diagrams in the layouts generated in this paper.

4.3 Small Multiples Graphs

We apply the idea of small multiples representations to show a long graph sequence in a single static view. This has benefits when comparing several graphs in a side-by-side view in order to find temporal patterns such as trends, countertrends, temporal shifts, oscillations, or anomalies/outliers. These patterns are difficult to find in a corresponding animated diagram, which only depicts one graph at a time and then smoothly transforms one to the next one in the sequence.

Each graph in a small multiples view is represented in the same layout with the vertices at the same relative positions. This strategy supports us to preserve the viewer's mental map because it has the highest degree of dynamic stability. Figure 3 illustrates a small multiples representation for 8 graphs in a sequence.

As an additional feature, the user can visually encode each graph in the sequence individually by a different visual metaphor, i.e., a hybrid small multiples representation, see Figure 4. This is beneficial, when graphs in a longer sequence switch from a rather sparse behavior to a denser one. This means, for the sparse graphs, a node-link visual metaphor should be applied since the chance of link crossings is low. If the graphs become dense, hairball-like structures may be the result which cannot be made more readable by edge splatting. Consequently, the user wishes to switch to either an adjacency matrix or adjacency list visualization.

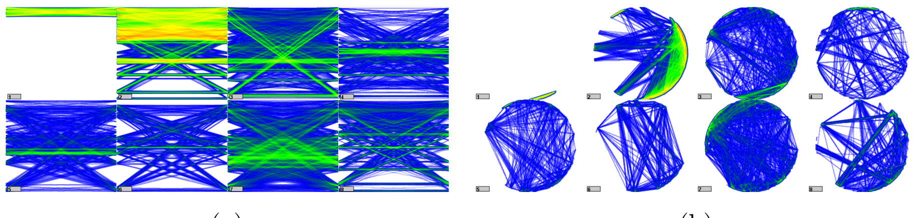


Fig. 3. A sequence of 8 graphs represented as two rows of four splatted graphs in a bipartite layout (a) and two rows of four splatted graphs in a radial layout (b).

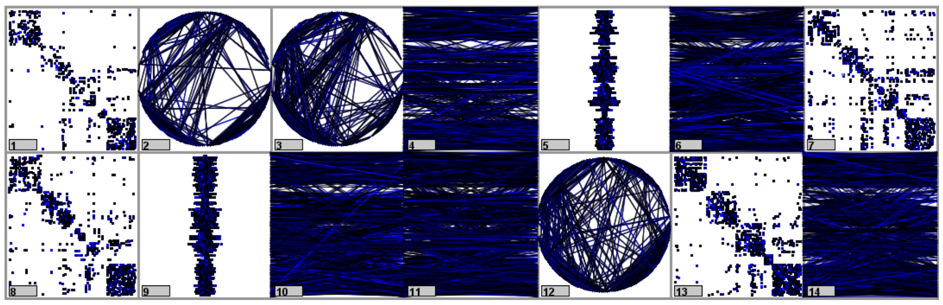


Fig. 4. A hybrid small multiples representation for representing a graph sequence in different visual metaphors for each single graph.

4.4 Interaction Techniques

Several interaction techniques are included in our tool to complement the small multiples visualization and support the data analysis.

- **Grid flipping:** Flipping through the grids can be used to analyze the dynamic data. The perceptual abilities and fast pattern recognition by the user are exploited by this concept with the goal to quickly get an overview of the evolution of longer graph sequences.
- **Scrolling:** Apart from directly flipping the grids, traditional scrolling interaction is implemented, allowing users to browse through the graph sequence one by one and not subsequence by subsequence.
- **Focus-and-context:** When many graphs are shown side-by-side on a grid, details are hard to see. For this reason, we support large singles (focus) together with the small multiples (context).
- **Grid subdivision:** The number of rows and columns can be defined by the user, which directly influences the dynamic visual graph patterns as well as the size of each single graph to be displayed in each grid cell.
- **Choice of layout:** We support bipartite and radial node-link layouts. These layouts can be selected interactively. Other layouts could easily be integrated in our visualization tool. Apart from node-link diagrams, also adjacency matrices and adjacency lists are integrated.
- **Graph aggregation:** Graph subsequences Γ_{l-m} can be aggregated on user's demand. All graph edges are transformed by applying aggregation techniques.

- **Graph comparison:** If there is much visual clutter in the diagrams and one is only interested in seeing changes in the data, our tool is able to compare subsequent graphs and only show the added or deleted edges between these neighboring graphs.
- **Weight filters:** To further reduce visual clutter, edges can be filtered by their weights, i.e., only those links are displayed whose weights are inside a user-defined weight interval.
- **Details-on-demand:** If the user selects one single graph from the displayed sequence, it is visualized in a separate frame in a larger scale. Additional textual information such as the number of vertices, edges, and time steps or the maximum weight can be shown on user's demand either in a separate frame or as a mouse tooltip.

4.5 Visual Patterns

In this section, we discuss typical visual patterns produced by our visualization technique. These patterns should correspond to patterns in the represented dataset. Therefore, we illustrate possible patterns in the visualization and map them to characteristics in the time-varying relational data. We subdivide these patterns into static and dynamic ones.

4.5.1 Static Patterns

There are several visual patterns that can be derived from a static graph alone. Aesthetic graph drawing criteria emphasized by the applied layout algorithm or vertex ordering are responsible for a readable diagram, which helps unhide these patterns. The most important ones are described in the following.

- **Clusters:** A cluster is indicated by a set of nodes displayed close to each other and connected by several links. These typically generate dense graph regions.
- **Graph densities:** A graph can have a certain degree of density, which can be visually derived by having, e.g., a look at the color coding of the density field (in a splatted node-link diagram). A graph might also be globally sparse but locally dense, which is typically the case when it consists of several stand-alone clusters.
- **Symmetries:** Some subgraphs may contain similar visual patterns, which can indicate that for these subsets of vertices the corresponding relations behave in a similar way.
- **Inter-cluster relations:** If there are several clusters (i.e., dense subgraphs), these may be connected by a few links, which can be directly detected in the graph.
- **Anomalies and outliers:** Unexpected or "strange" behavior can be classified as anomaly. Such a phenomenon can be visually derived from the visualization by looking for visual patterns that do not match any of the described categories.

4.5.2 Dynamic Patterns

Based on these static patterns, a dynamic graph may also contain several time-varying ones. In the TimeEdgeTrees visualization [12], we illustrated several dynamic patterns for quantitative data that can easily be adapted to dynamic graph visualization.

- **Trends:** A trend is a steadily increasing or decreasing pattern. For example, a cluster pattern might get denser or sparser or the weight of an inter-cluster relation might grow or shrink over time.
- **Countertrends:** A countertrend is a behavior that shows characteristics that are opposite to another identified trend pattern; for example, if a cluster pattern gets denser and denser, another cluster pattern gets sparser and sparser in the same time interval.
- **Stabilities:** A stability pattern is apparent if over a longer time a graph substructure behaves constantly, i.e., no changes occur in this time interval.
- **Oscillations and periodicities:** A visual pattern may periodically change over time. Seasonal effects might be the reason for such a phenomenon, i.e., clusters may disappear for some time and then reappear again.
- **Temporal shifts:** If a visual pattern occurs in a subgraph for some time and with some delay a similar pattern occurs in another subgraph, this is called a temporal shift.
- **Anomalies and outliers:** Also in the dynamic graph case, some unexpected or "strange" behavior might occur over time that cannot be described by any of the patterns above.

5 Case Studies

We illustrate the usefulness of our dynamic graph visualization technique by means of two case studies: a dynamic call graph and a dynamic social network.

5.1 Call Graphs

The evolution of software and, in particular, the evolution of call relations provide interesting datasets to be analyzed [10]. In this work, we will have a look at dynamic call graphs extracted from open source software development by first preprocessing data from configuration management systems [8, 9]. Today's software systems typically consist of several thousands of methods that may be related by their calling mechanisms. The changes of these relations from release to release make the analysis and visualization of such data challenging. Such information is important for software developers, maintainers, and managers to detect and identify design flaws such as unwanted call relations for example.

In this case study, we explore the dynamic call relations of the JUnit regression testing framework (www.junit.org). The dynamic graph data contains 20 graphs (releases). These were obtained by running the Java bytecode compiler and by using the *DependencyFinder* (depfind.sourceforge.net) to extract method call dependencies. This process resulted in 2,817 graph vertices related by more than 10,000 edges. Moreover, the graph vertices are hierarchically organized and ordered by the software system structure. This project hierarchy can be used in the visualization as a meaningful 1D vertex order (which could further be improved by traversing the subhierarchies).

Figure 5 shows a small multiples view of the 20 call graphs from the JUnit project in a single static view. In this figure, we experiment with all possible visual metaphors supported by our visualization tool, i.e., bipartite node-link layouts, radial node-link layouts, adjacency matrices, adjacency lists, and a hybrid small multiples representation. By inspecting the individual graphs, we can observe several cluster patterns, inter-cluster relations, dense regions, symmetries, and outliers.

Looking at the graph evolution shown in Figure 5 (a) in the bipartite layout, one can easily see some interesting patterns. There are significant visual differences, i.e., major changes in the development of the system. For example, from time step 11 to 12, the stability pattern is changed by an abrupt increase of functionality. For the next 4 time steps, the call relations seem to remain stable again apart from some outliers in graphs 14 and 15. Then from graph 15 to 16, there seems to be another increase in call relations, which seems to remain nearly stable until the very last of the software releases.

Having a look at the same dynamic graph data depicted as small multiples of radial layouts (Figure 5 (b)), we lose the information about the direction of the call relation on the one hand. On the other hand, we obtain a more clutter-free diagram because the long crossing links from top left to bottom right (as in the bipartite layout) are avoided in a radial layout. Here, we can confirm the observations made earlier in the bipartite layout, but we can get even more insights due to the more clutter-free representations. It may be noted that it is of great importance to play around with several layouts for dynamic graph data because it can help unhide patterns that may be hidden in one layout but not in the other.

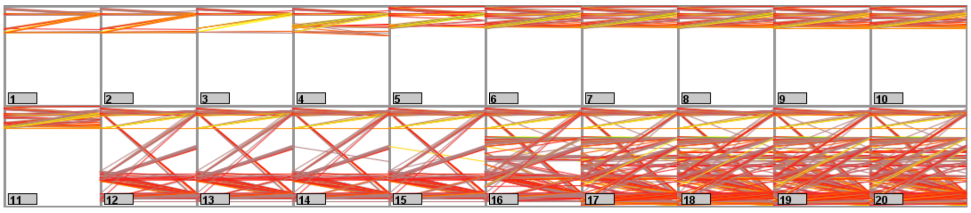
Asking for details-on-demand shows that the `junit.framework` package is not changed very much, an information that can also be observed by computing and visualizing the added and deleted relations between subsequent graphs as in [11]. The flip-book feature can further support the change detection.

The evolution of clusters can easily be explored by either looking at the horizontally occurring dense structures in the bipartite small multiples or looking for denser circle segments in the radial layouts. These clusters for call relations indicate high cohesion in the corresponding software artifact, i.e., classes, directories, or packages—depending on the level of hierarchical granularity. Examples of these patterns are given by the `Assert` class in the `org.junit` package as obtained by looking at the details-on-demand function. Such dense clusters can easily be detected in the corresponding matrix visualizations, see Figure 5 (c). If we are more interested in vertices (methods) that are called frequently and that call frequently, a list-based representation is useful, see Figure 5 (d). A combination of the visual metaphors is shown in Figure 5 (e), in which the matrix representations have benefits for dense clusters and in which node-link diagrams can be helpful to trace paths, also over time.

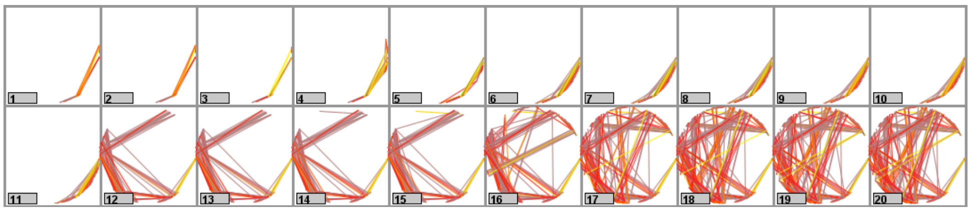
There are also some trend patterns depicted in the small multiples diagram. There is some kind of steady increase of call relations between some elements, e.g., in the `org.junit.runner` package. By having a look at the graph comparison mode for added relations, we can confirm such an increase of relations.

5.2 Social Network

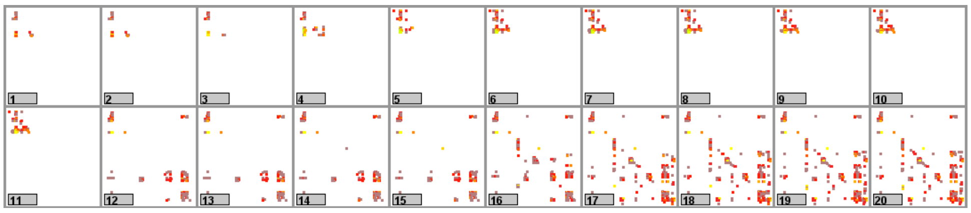
Another application is the visualization of dynamic social networks. As an example, we preprocessed data from the ACM Hypertext conference in 2009, as already described in [4]. In this freely available dataset, face-to-face proximities of conference attendees were monitored by RFID badges during the conference days. If two participants faced each other at a small distance over at least 20 seconds, this was recorded



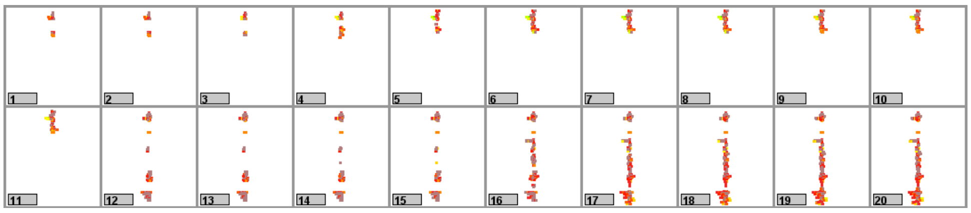
(a)



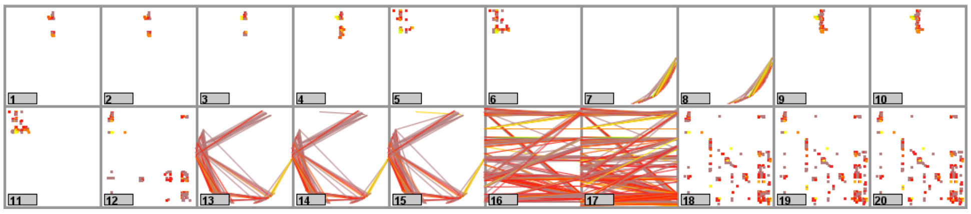
(b)



(c)



(d)



(e)

Fig. 5. 20 releases of the JUnit project and the call graphs visualized as a sequence of bipartite node-link layouts (a), a sequence of radial node-link layouts (b), a sequence of adjacency matrices (c), a sequence of adjacency lists (d), and hybrid small multiples (e).

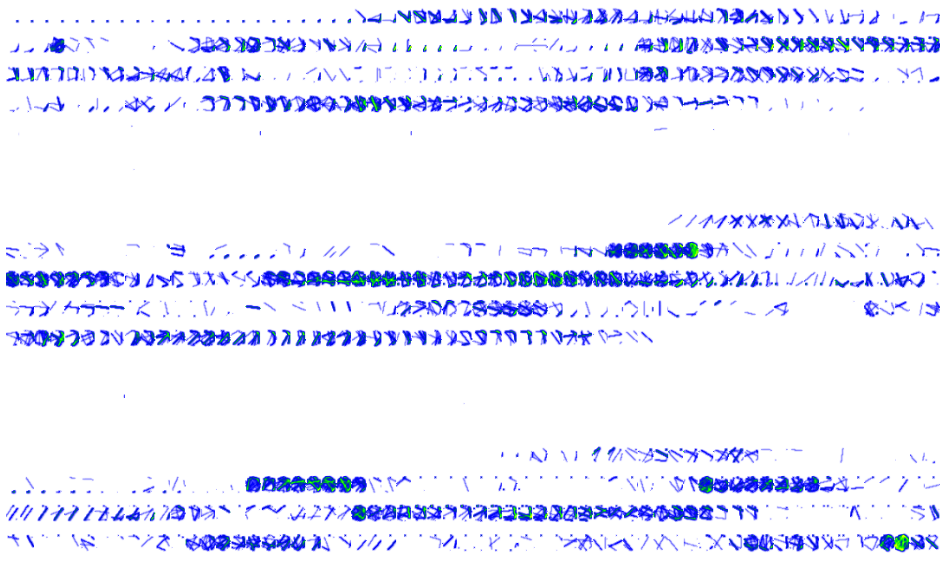


Fig. 6. A dynamic social network with 1,178 graphs as radial small multiples.

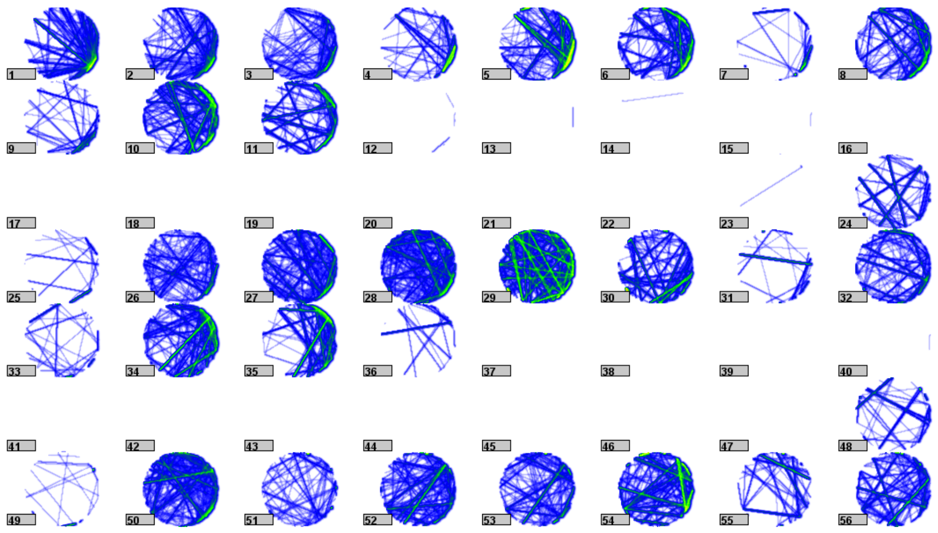


Fig. 7. Aggregation to 56 graphs of the same dataset as shown in Figure 6.

as a contact between both. Doing this for all attendees over the whole conference leads to a dynamic social network dataset.

In our scenario, the data consists of 20,818 face-to-face contacts of 113 conference attendees. The conference was held over three days and the data was anonymized. Hierarchical clustering was first applied to find a suitable order among the graph vertices, similar to the scenario in the dynamic call graph application where the hierarchy and vertex order was given by the software system structure.

Figure 6 shows 1,178 graphs recorded during the conference days. Looking at this overview small multiples diagram, one can easily see that the time-varying graphs can be visually separated into three subsequences, which comes from the conference days. This means that face-to-face contacts were occurring more frequently during the days than during the nights, which is an expected phenomenon. Getting such an overview is difficult by standard graph animation techniques and existing time-to-space mappings. A drawback of our overview-based visualization is the poor visual scalability. However, the overall dynamic patterns are clearly visible. Furthermore, the flip-book feature allows us to display graph subsequences in a larger display space.

If we are now interested in looking at a single day, it is difficult with this overview representation since the displayed graphs are far too small to be further analyzed. Therefore, we decide to modify the grid to only show the graphs of one day (from 0 a.m. to 24 p.m.) at a time. This results in three grids that can now be flipped by applying the flip-book feature. This further uncovers interesting graph patterns. We can see that for some time intervals the graphs are sparser than for other time intervals in which they become denser. This phenomenon can be explained by the conference sessions, i.e., there are only a few face-to-face contacts during the sessions but much more during the coffee breaks. This holds for all three days. Cluster structures and dense graph regions can be detected by looking for green color-coded graph regions. There are still some outliers, for example single relations during the sessions.

Picking out only individual graphs (grid size 1×1), one can use the flip-book feature to rapidly browse the graphs and to see how the time-varying behavior is changing from graph to graph. This is possible because the graph nodes remain at fixed positions in our layout and since we apply edge splatting for detecting graph patterns in dense graph regions. If one graph is of special interest, it can be further analyzed by filter functions or details-on-demand features, which can also be applied to any kind of graph subsequence.

Figure 7 shows an aggregated view of the same social network dataset as in Figure 6. In this scenario, 56 graphs remain which can now be displayed in larger space. The three conference days can still be visually separated by the subsequences of empty graphs during the nights. Aggregation is hence a functionality to obtain more display space, but, on the negative side, many time-varying details are hidden in the aggregated graphs.

6 Discussion

The small multiples based visualization for dynamic graphs has several benefits but also a number of drawbacks. We will discuss some aspects concerning positive and negative points on a comparative basis with time-to-space and time-to-time mappings in the following.

- **Visual scalability:** In an animated node-link diagram, a high degree of visual scalability is achieved since each graph can be displayed on the entire display. In a traditional time-to-space mapping, the display space has to be divided into as many slices as time steps have to be displayed. The same holds for our small multiples representation.
- **Algorithmic complexity:** When a node-link diagram is animated from one time step to the next, typically smooth animation is used to help preserve a viewer's mental map. Sophisticated layout algorithms have to be designed in order to

achieve a high degree of dynamic stability when the graph is animated. If the graph changes abruptly, this goal is hard to reach. In time-to-space mappings, instead, the vertex positions are computed once in the beginning and remain fixed over time. This helps preserve the mental map, which is a goal of this work, in particular, when longer graph sequences have to be explored for time-varying patterns.

- **Visual clutter and overdraw:** In time-to-space mappings, the vertices are oftentimes placed on 1D lines to better compare them in an aligned fashion. Layouts restricted to 1D have a higher probability of link crossings and consequently of visual clutter. Such overdraw and clutter are reduced by edge splatting, which is not required in traditionally animated node-link diagrams.
- **Further clutter reduction techniques:** Edge bundling might be used as another technique to reduce the amount of visual clutter. In contrast to edge splatting, which does not spatially change node and link positions, it bundles edges together. A combination of edge bundling and edge splatting might be a promising approach to reduce visual clutter that still gives a feeling about the number of links contained in a bundle.
- **Layout dependency:** Aesthetic criteria such as a minimization of link crossings, an even distribution of vertices in the plane, or a maximization of symmetries have to be taken into account by layout algorithms. This is problematic for animated diagrams that have to look for dynamic stabilities as one criterion and additionally for aesthetically appealing and readable layouts of each graph. In time-to-space mappings, this is not that problematic. We are also able to switch between several layouts on demand.
- **Solving comparison tasks:** A most important task in time-series data visualization is the comparison between time steps. This is difficult for animated diagrams, where only one graph at a time is displayed. In a time-to-space mapping, many graphs are displayed and the users can inspect them in a single view.
- **Data attachments:** In an animated diagram, it is difficult to attach additional data such as a hierarchical organization of the vertices or vertex attributes. If the vertices are aligned and mapped to 1D lines as in the bipartite layout, such additional information can be added to the static plot and aligned with the vertices. This is difficult in a small multiples representation laid out in a grid, but it is easier than in a corresponding animated diagram. If a suitable data attachment visualization is found for a single graph, this can be applied to each individual graph in the same way, making the data attachment easily comparable in a time-to-space mapping.

7 Conclusion and Future Work

In this article, we show an extension of our previous work on a flip-book interaction for edge-splatted small multiples visualizations of dynamic graphs [13]. We additionally integrate adjacency matrices and adjacency lists, support a hybrid small multiples approach, integrate a focus-and-context technique, and finally allow the user to explore time-varying shortest paths. To reach this goal, we visually map a graph sequence to a modifiable 2D grid, which can then be flipped on user's demand similar to the concept known as Rapid Serial Visual Presentation (RSVP). Here, the perceptual abilities of the human visual system are exploited for fast pattern recognition. We classify the visual patterns of our visualization technique into static and dynamic ones. In two case studies, we use these visual pattern categorizations to uncover static and dynamic characteristics in the example datasets coming from applications in software development and social networks. Interaction techniques are applied in these case studies to obtain more detailed insights into the static and dynamic graph data and to illustrate the usefulness of our novel visualization concepts.

For future work, we plan to add more node-link graph layout algorithms such as force-directed and hierarchical layouts. Also, a comparative user study should be conducted to assess the benefits and drawbacks of time-to-space mapping, time-to-time mapping, and our hybrid flip-book interaction. Eye tracking might be a good technology to uncover visual task solution strategies of the study participants [6, 22] and to find difficulties they have when using our technique. These insights could then be used to further enhance the presented visualization.

References

[1] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data*. Springer, 2011.

[2] D. Archambault, H. C. Purchase, and B. Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):539–552, 2011.

[3] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. The state of the art in visualizing dynamic graphs. In *EuroVis State-of-the-Art Reports*, EuroVis STAR, 2014.

[4] F. Beck, M. Burch, C. Vehlow, S. Diehl, and D. Weiskopf. Rapid serial visual presentation in dynamic graph visualization. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 185–192, 2012.

[5] U. Brandes and B. Nick. Asymmetric relations in longitudinal social networks. *IEEE Trans. on Visualization and Computer Graphics*, 17(12):2283–2290, 2011.

[6] M. Burch, G. L. Andrienko, N. V. Andrienko, M. Höferlin, M. Raschke, and D. Weiskopf. Visual task solution strategies in tree diagrams. In *Proceedings of the IEEE Pacific Visualization Symposium*, pages 169–176, 2013.

[7] M. Burch and S. Diehl. TimeRadarTrees: Visualizing dynamic compound digraphs. *Computer Graphics Forum*, 27(3):823–830, 2008.

[8] M. Burch, S. Diehl, and P. Weißgerber. EPOSee - A tool for visualizing software evolution. In *Proceedings of the 3rd International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT*, pages 127–128, 2005.

[9] M. Burch, S. Diehl, and P. Weißgerber. Visual data mining in software archives. In *Proceedings of the ACM 2005 Symposium on Software Visualization*, pages 37–46, 2005.

[10] M. Burch, C. Müller, G. Reina, H. Schmauder, M. Greis, and D. Weiskopf. Visualizing dynamic call graphs. In *Proceedings of the Vision, Modeling, and Visualization Workshop*, pages 207–214, 2012.

[11] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf. Parallel edge splatting for scalable dynamic graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2344–2353, 2011.

[12] M. Burch and D. Weiskopf. Visualizing dynamic quantitative data in hierarchies – TimeEdgeTrees: Attaching dynamic weights to tree edges. In *Proceedings of the International Conference on Information Visualization Theory and Applications*, pages 177–186, 2011.

[13] M. Burch and D. Weiskopf. A flip-book of edge-splatted small multiples for visu-

alizing dynamic graphs. In *Proceedings of the International Symposium on Visual Information Communication and Interaction, (VINCI)*, pages 29–38, 2014.

[14] O. de Bruijn and R. Spence. Rapid serial visual presentation: A space-timed trade-off in information presentation. In *Proceedings of Advanced Visual Interfaces*, pages 189–192, 2000.

[15] S. Diehl and C. Görg. Graphs, they are changing. In *Proceedings of Graph Drawing*, pages 23–30, 2002.

[16] Y. Frishman and A. Tal. Online dynamic graph drawing. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):727–740, 2008.

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

[18] S. Ghani, N. Elmqvist, and J. S. Yi. Perception of animated node-link diagrams for dynamic graphs. *Computer Graphics Forum*, 31(3):1205–1214, 2012.

[19] M. Ghoniem, J.-D. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *Proceedings of IEEE Symposium on Information Visualization*, pages 17–24, 2004.

[20] M. Hlawatsch, M. Burch, and D. Weiskopf. Visual adjacency lists for dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, 20(11):1590–1603, 2014.

[21] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[22] K. Kurzhals, B. D. Fisher, M. Burch, and D. Weiskopf. Evaluating visual analytics with eye tracking. In *Proceedings of the Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, pages 61–69, 2014.

[23] H. C. Purchase, E. E. Hoggan, and C. Görg. How important is the “mental map”? – An empirical investigation of a dynamic graph layout algorithm. In *Proceedings of Graph Drawing*, pages 184–195, 2006.

[24] S. Rufiange and M. J. McGuffin. DiffAni: Visualizing dynamic graphs with a hybrid of difference maps and animation. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2556–2565, 2013.

[25] S. Rufiange and G. Melançon. AniMatrix: A matrix-based visualization of software evolution. In *Proceedings of the IEEE Working Conference on Software Visualization*, 2014.

[26] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of Visual Languages*, pages 336–343, 1996.

[27] R. Spence and M. Witkowski. *Rapid Serial Visual Presentation – Design for Cognition.* Springer, 2013.

[28] K. Stein, R. Wegener, and C. Schlieder. Pixel-oriented visualization of change in social networks. In *Proceedings of the International Conference on Advances in Social Networks Analysis and Mining*, pages 233–240, 2010.

[29] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.

[30] E. R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 1983.

[31] E. R. Tufte. *Envisioning Information.* Graphics Press, 1990.

[32] B. Tversky, J. B. Morrison, and M. Bétrancourt. Animation: can it facilitate? *International Journal on Human-Computer Studies*, 57(4):247–262, 2002.

[33] S. van den Elzen and J. J. van Wijk. Small multiples, large singles: A new approach for visual data exploration. *Computer Graphics Forum*, 32(3):191–200, 2013.