Visual Adjacency Lists for Dynamic Graphs

Marcel Hlawatsch, Michael Burch, and Daniel Weiskopf, *Member, IEEE Computer Society*

Abstract—We present a visual representation for dynamic, weighted graphs based on the concept of adjacency lists. Two orthogonal axes are used: one for all nodes of the displayed graph, the other for the corresponding links. Colors and labels are employed to identify the nodes. The usage of color allows us to scale the visualization to single pixel level for large graphs. In contrast to other techniques, we employ an asymmetric mapping that results in an aligned and compact representation of links. Our approach is independent of the specific properties of the graph to be visualized, but certain graphs and tasks benefit from the asymmetry. As we show in our results, the strength of our technique is the visualization of dynamic graphs. In particular, sparse graphs benefit from the compact representation. Furthermore, our approach uses visual encoding by size to represent weights and therefore allows easy quantification and comparison. We evaluate our approach in a quantitative user study that confirms the suitability for dynamic and weighted graphs. Finally, we demonstrate our approach for two examples of dynamic graphs.

Index Terms—Graph visualization, weighted graphs, dynamic graphs, adjacency lists

1 Introduction

Many visualization techniques exist for the analysis and exploration of static graphs [1], [2]. A prominent visual metaphor is the node-link diagram, which can become cluttered for dense graphs [3]. In contrast, matrix representations result in clutter-free diagrams but may require more space. Furthermore, it is difficult to compare and aggregate the weights of links with both representations.

An even more challenging task is the visualization of dynamic graphs. An animated sequence of node-link diagrams is intuitive, albeit questionable in many cases [4]. Static diagrams allow a better comparison of subsequent time steps as demonstrated for node-link diagrams [5] and matrix-based visualizations [6].

We introduce a novel approach to visualizing dynamic graphs exploiting the concept of adjacency lists. By using a list-based representation, we achieve a compact and clutter-free graph visualization. Furthermore, we are able to analyze evolving graphs in a side-by-side display, focusing on the investigation of dynamic weights and link duration. We evaluate and compare our approach with node-link diagrams and adjacency matrices in a quantitative user study. We further illustrate the utility with two case studies: dynamic data representing worldwide migration and the temporal evolution of a visualization workflow.

2 Related Work

Visualizing dynamic graphs is typically done by a natural time-to-time mapping exploiting the concept of animation. Two sub-concepts exist: The first requires the whole graph sequence in advance, denoted as off-line dynamic graph visualization [7], [8], [9]. Complementing that, online dynamic graph visualization [10], [11] just requires the next element of the sequence at the time when it is rendered. Both approaches are subject to high algorithmic complexity to produce aesthetically pleasing graph layouts [12], [13], [14], [15] that preserve the viewer's mental map [16] and achieve a high degree of dynamic stability [17]. Extraction and visualization of clusters can also be done for dynamic graphs [18], [19]; recent work also considers the dynamics of associated attributes [20].

Our visualization technique is based on a static representation for dynamic graphs, i.e., a time-to-space mapping. This is a common concept in the visualization of time-dependent data [21], often realized by using bars or timelines [22], [23], [24], [25]. One of our layout variants results in a visualization similar to Gantt charts [26]. However, to the best of our knowledge, none of the above charts or diagrams have been applied before to show the structure of dynamic graphs. The static visual representation has several benefits: Graphs can be compared visually and the viewers do not have to rely on their short term memory [4], [27] as it would be required with animation. The mental map is preserved and dynamic stability is achieved, which also allows for easy integration of interaction techniques.

The concept of static diagrams for time-varying node-link diagrams has recently been used in TimeArcTrees [28]. Another example of static visualization relying on the node-link visual metaphor is parallel edge splatting [5]. A further possibility is to use a layered approach of node-link diagrams for visually encoding the dynamics of a network in a stacked 3D representation [29]. Finally, a hybrid approach combining animated and static node-link visualization was recently presented [30].

For dense graphs, matrix-based representations are more suitable [31]. However, they exhibit problems when dealing with path-related tasks such as finding a shortest route from a source to a target node. Matrix

• *Marcel Hlawatsch, Michael Burch, and Daniel Weiskopf are with the Visualization Research Center (VISUS), University of Stuttgart, Germany. E-mail: {Marcel.Hlawatsch, Michael.Burch, Daniel.Weiskopf}@visus.uni-stuttgart.de*

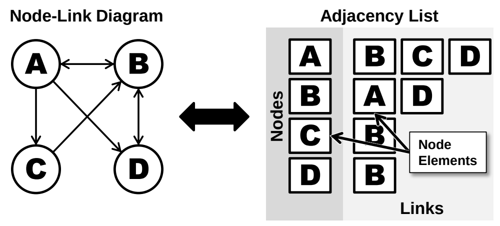


Fig. 1: Concept of visual adjacency lists. Our example graph is shown with the classic node-link diagram (left) and with the visual adjacency list (right). Our list-based approach uses two orthogonal axes: one axis for all nodes in the graph (vertical), and one axis for the corresponding target nodes of outgoing links (horizontal). Links are defined by their vertical position relative to the node axis and their label, e.g., node “A” has outgoing links to “B”, “C”, and “D”.

visualizations hence trade visual clutter for path reading problems. In the case of layered graphs, Quilts [32] are an approach to visualizing them with a compact matrix-based layout. DAGView [33] uses a combination of node-link and matrix representation. Generally, matrix-based techniques can visually encode dynamic graphs [6], [34], [35] in a space-filling and clutter-free approach to illustrate time-varying weights by attaching the time axis to each of the matrix cells.

In contrast, our approach uses a list-based representation and is not specifically designed for a certain type of graph. This representation shares a few characteristics of matrix techniques: it avoids link crossings, but following paths is difficult (see Section 7). However, the list approach results in a more compact visualization, which is especially useful for sparse graphs and dynamic data.

3 Visualization of Static Graphs

Our graph visualization is based on adjacency lists, which is a well-known concept for graph representation [36]. However, adjacency lists are typically only used to internally represent graphs in computer memory and algorithms. We use them now to generate a visual representation of the graph, called *visual adjacency list* (Figure 1). We choose two orthogonal axes for our list, but the concept is not restricted to this; other layouts like radial approaches are possible.

Nodes are represented by graphical primitives, called *node elements*, which can have a label and/or a color. We use box shapes for the node elements. Links are represented by the node elements of the target nodes. We decided to list all nodes of the graph on a vertical axis and list the target nodes of their outgoing links on a horizontal axis. Labels specify the target nodes. With this layout, people used to reading from left to right can intuitively read the list.

Typical graph visualization techniques like node-link or adjacency matrix visualizations are symmetric with respect to links: source and target of a link are represented with the same visual encoding. In contrast, our proposed layout results in an asymmetric representation of links. While the source nodes are

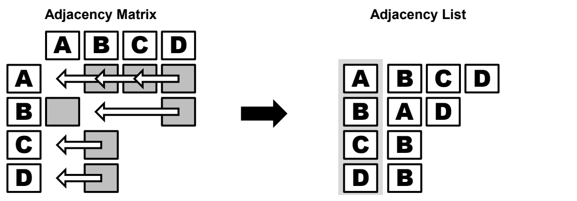


Fig. 2: Relationship between adjacency matrix and list. The adjacency list can be derived from the adjacency matrix by shifting all entries to the left until there are no gaps (and by labeling the entries).

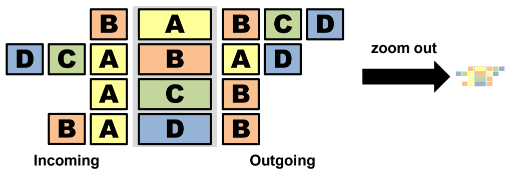


Fig. 3: Axis for incoming links and color coding. Incoming links can be explicitly represented with an additional axis (to the left here) to support tasks on them. Node “A” has, e.g., only one incoming link from node “B” (see Figure 1, left). To improve scalability, colors are used in addition to labels to specify nodes; they are still recognizable when scaling down the graph. The color map is defined on the node axis in the center. Increasing the width of the node axis visually separates it from the link axes even on zoomed-out levels (right).

spatially arranged, the target nodes of links are only encoded with labels and/or colors. In this way, the visualization is more focused on source nodes and more suitable for tasks centered on them.

As a consequence of the underlying concept of adjacency lists, this visualization is a complete representation of an unweighted graph. It is directly applicable to both directed and undirected graphs. We discuss the inclusion of weights in Section 3.4. Furthermore, adjacency matrices and lists are strongly related (Figure 2). The list can be regarded as a version of a matrix without gaps because the horizontal position is not required to specify links. Therefore, the list can be derived from the matrix by shifting the entries to the node axis, removing all gaps in between, and the matrix can be derived from the list by assigning the proper horizontal positions to the node elements.

3.1 Incoming Links

With the current representation, it is hard to extract all incoming links of a specific node because the whole list has to be read. Hence, this operation is linear in the number of all links, while extracting all outgoing links of a specific node is only linear in the number of the outgoing links of that node. To improve on this, we explicitly represent incoming links with an additional horizontal axis from the center to the left (Figure 3). In this way, the reading scheme for links remains from left to right, i.e., source nodes are always located left from target nodes. Since the node axis is now placed in the center, it should be visually separated from the link axes. To this end, we increase the size of the node axis to support scalability to large graphs.

The explicit representation of incoming links doubles the spatial extent, but the completion of many tasks related to incoming links is accelerated.

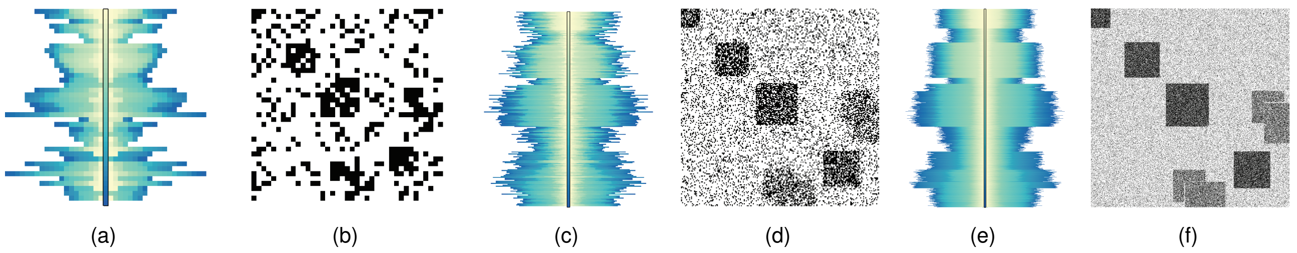


Fig. 4: Scalability of adjacency lists (a, c, e) and matrices (b, d, f) with respect to the number of nodes. Synthetic graphs with the same link structure and increasing node count ((a, b): 40 nodes, (c, d): 200 nodes, (e, f): 1000 nodes) were generated. Large-scale structures are visible independently of the node count. However, with increasing node count, the structures become clearer with both types of visualizations.

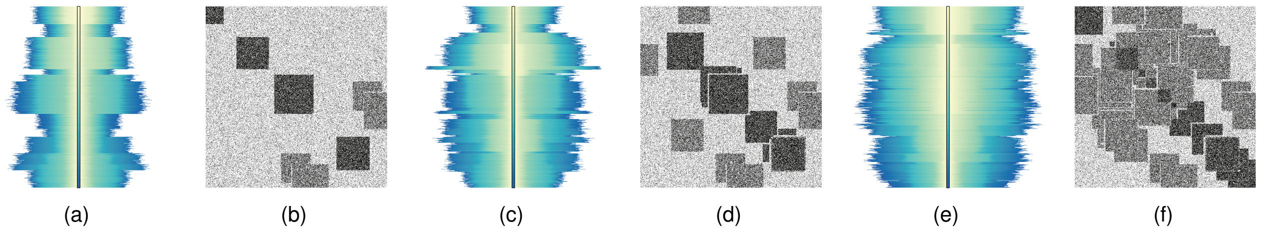


Fig. 5: Scalability of adjacency lists (a, c, e) and matrices (b, d, f) with respect to increasing number of clusters (from left to right) in synthetic graphs with fixed node count (1000). Even nearby cluster structures can be differentiated in adjacency matrices. This is not possible with adjacency lists due to the color encoding. Only structures with a larger distance with respect to the node order can be differentiated there.

3.2 Scalability

So far, target nodes of outgoing links (and source nodes of incoming links) are only specified by their labels. This poses scalability problems: the visualization scales only to a size, for which the labels are still readable. To improve scalability, we additionally use color to represent nodes (Figure 3). The color map is defined and shown on the node axis. With color coding, the visualization scales up to a level at which a node is represented by a single pixel. Going to the sub-pixel level, where several nodes are represented by a single pixel, requires appropriate schemes to select the color of this pixel and is part of future work.

So far, we use standard interaction techniques: zooming and panning, and displaying tooltips. Specifically designed interaction techniques, e.g., zoom lenses or highlighting, may further improve the work with large graphs and are part of future work.

For a more detailed discussion of the scalability of our approach, we compare visual adjacency lists with adjacency matrices because of their close relationship (see Figure 2). Figures 4 and 5 show results for both techniques for synthetic graphs with varying node count or numbers of clusters. The graphs were generated from the adjacency matrices to avoid issues with row and column ordering. For the basic graph structure, links between randomly selected nodes were created, using uniformly distributed random numbers, until the given link density was reached. To generate the cluster structures, this procedure was repeated in subareas of the matrix with a higher density.

In the first case, the link structure and density was kept constant, only the node count was changed (Figure 4). We can see that even in the case of small graphs (Figures 4(a) and 4(b)), identifying individual nodes or links requires interaction on typical

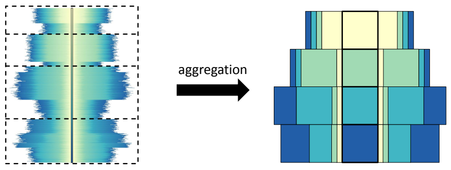


Fig. 6: Aggregation for adjacency lists. To improve the analysis of large graphs (left), nodes and the respective links are aggregated. The resulting adjacency list (right) shows the representatives of the aggregated nodes and their links. The aggregated number of links or weights are shown with horizontal weight mapping (see Section 3.4).

displays—regardless of the chosen technique. Large scale structures, however, are directly visible in both visualizations. They even tend to appear clearer in graphs with higher node count (Figures 4(e) and 4(f)). The difference between both techniques is that the color coding of visual adjacency lists usually provides a lower resolution on typical output devices than the spatial encoding of adjacency matrices. Therefore, the scalability of adjacency matrices with respect to the number of cluster structures (Figure 5) is better. While small and nearby structures are still visible and distinguishable in adjacency matrices (Figure 5(f)), this is not the case for adjacency lists (Figure 5(e)).

In summary, directly applying visual adjacency lists to large graphs can provide only an overview of large scale structures. They offer good scalability with respect to the number of nodes. However, the representation of cluster structures is of lower resolution compared to adjacency matrices. Similar to many other techniques, a more detailed representation of large graphs requires their aggregation (Figure 6).

3.3 Flexibility

There are two constraints in our approach for defining links. First, target nodes must have the correct position with respect to the node axis to identify source

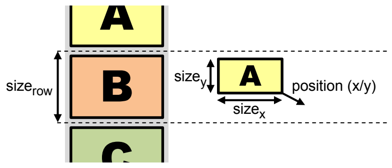


Fig. 7: Flexibility in link representation. Since links are defined by the associated row in the list and labels/colors, flexibility remains in the position and size of node elements. Node elements can be moved and scaled as long as they stay inside the respective row.

nodes (Figure 7), i.e., the node element of the target node must be in the correct row to be associated with the respective source node. Second, the node element is assigned with the label and color of the target node.

This leaves some flexibility to the size and position of the node element. In principle, the size in x-direction ($size_x$) is not restricted. However, the visualization becomes more difficult to read with increasing size. In y-direction, the size ($size_y$) is restricted by the size of the row, but the row size ($size_{row}$) does not have to be uniform and can vary from row to row. This is used, e.g., for the Gantt layout in Section 4.2.

The position of the node element in x-direction is only required to be to the right of the node axis in case of outgoing links (and to the left in case of incoming links). If $size_y < size_{row}$, the position in y-direction is also flexible, as long as the node element stays inside the respective row. This is again used in Section 4.2.

3.4 Weighted Links

The previously described flexibility for the node elements can also be used to visualize weighted graphs. We have chosen two different ways to incorporate weighted links (Figure 8). In both cases, the weight is mapped to the size of the node element in one dimension, which is a concept similar to bar charts. The advantage of this is that the weights can be easily quantified and compared [37], [38].

The first variant uses horizontal weight mapping, i.e, the weight is mapped to $size_x$ of node elements (Figure 8(a)). In this case, there must not be any gaps between the node elements in a row. This results in a representation similar to a stacked bar chart. Weights for different rows can be compared and the sum over all weights is visible. However, it is more difficult to compare the weights inside a row. Furthermore, it can be misleading to use a non-linear scale (e.g., logarithmic), because the total size does not correspond anymore to the sum over all weights.

The second variant uses vertical weight mapping, i.e, the weight is mapped to $size_y$ of node elements (Figure 8(b)). In this case, the weights have to be normalized in a way that the node elements do not exceed the row size. This normalization can be done by considering the maximum weight inside the row only or over all rows. In the case of dynamic graphs, the normalization can also consider all time steps. Depending on the choice of normalization, the weights

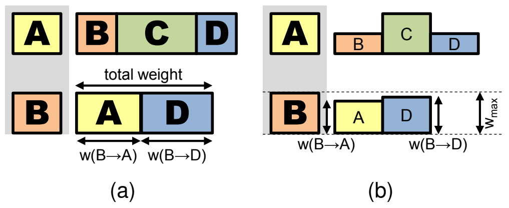


Fig. 8: Weight representation in visual adjacency lists. (a) Weights can be mapped to $size_x$ of the node elements. The summed weight of all outgoing links can be derived from the visualization. (b) Weights can be mapped to $size_y$ of the node elements. In this case, the weights have to be normalized so that the node elements stay in their row to meet the constraints from Section 3.3.

can be compared only inside a single row, or between rows, or, in the case of dynamic graphs, even between time steps. With the second variant, weights inside a row can be easily compared. Furthermore, non-linear scales can be used. However, the sum over all weights is not visible, it is difficult to compare weights between rows, and the weights are not visible when scaling to single pixel level.

Similar approaches may be used to represent weighted nodes, but this rather uncommon case is beyond the scope of this paper.

3.5 Node Order and Color Coding

Order and color coding of the nodes have a huge impact on the resulting visualization and influence the identification of graph structures, especially on the overview level. They typically go hand in hand, since color coding is applied after ordering the nodes on the node axis. A good combination of node order and color coding supports the analysis of the graph.

Ordering the nodes on the node axis and the links along the link axis is independent from each other and flexible. Even ordering the links in every row differently would result in a correct representation of the graph. In this case, however, it is very difficult to see similarities in different rows and detect clusters. It is therefore important to find an adequate ordering for the node axis. Along the link axis, it is reasonable to use the same ordering as for the node axis.

A simple scheme is to order with respect to certain properties of the graph. For example, the nodes can be ordered according to the number or summed weight of outgoing or incoming links. If nodes exhibit certain semantics or hierarchies, it is reasonable to order them accordingly if possible: we order the nodes of the visualization workflow in Section 6.3 according to the classic visualization pipeline [39]. More advanced ordering methods are outside the scope of this paper and may be inspired by reordering methods for adjacency matrices [40], [41]. Furthermore, graph clustering methods [42] can be used to create hierarchies.

It is reasonable to also use the hierarchy for the color coding of the nodes. For the visualization workflow, we use, e.g., four base colors for the four groups of nodes (see Figure 18(b)). Inside these groups, we

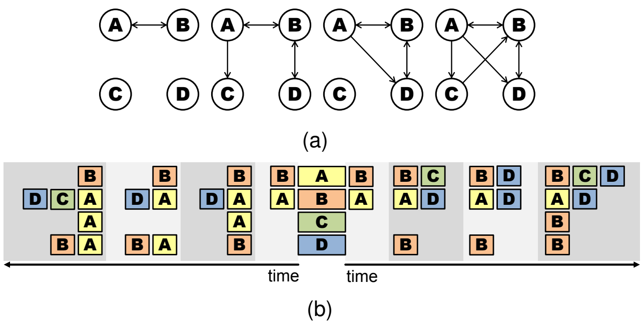


Fig. 9: Visual adjacency lists for dynamic graphs. (a) Our example graph consists of four time steps. (b) Visual adjacency lists display the individual time steps next to each other; incoming and outgoing links are grouped and time grows from the center to the outside.

modulate the brightness of the base color according to the position of the nodes inside the group.

The node order and color coding may be interactively changed by the user. By this, the user can filter or emphasize certain elements and may reveal different aspects of the graph, such as highlighting all links from and to a selected node.

4 Dynamic Graphs

To extend our concept for dynamic graphs, we exploit the compactness of our visual representation to create a static image of the full graph sequence. It is also possible to combine visual adjacency lists with animation, with all its advantages and disadvantages. However, we focus on static visualization in this paper.

4.1 Discrete Time Visualization

Visual adjacency lists can be directly adopted for dynamic graphs by putting links for varying times side-by-side (Figure 9). Here, we assume that time is discretized so that clearly separated time steps can be shown. Although the node and link structure can be dynamic, we assume that only the time spans of links are of interest and therefore use the union of all nodes from all time steps in the visualization.

Our layout positions the node axis in the center and groups the time steps of incoming (left) and outgoing (right) links. Time evolves from the center to the outside. Slight changes in the background color and small offsets visually separate the time steps. Furthermore, by normalizing with the maximum number of links or total weight, a uniform width of the time steps can be achieved. This is used in Figure 15.

The advantage of this layout is that it is easy to analyze the evolution of incoming and outgoing links. Furthermore, the symmetry of the concept remains for all time steps. However, it is difficult to compare the incoming and outgoing links of a specific time step. Additionally, the direction of increasing time is different for the two types of links, which is less intuitive to read. Therefore, other layouts are possible, e.g., the incoming and outgoing links of each time step can be grouped with time evolving from left to right.

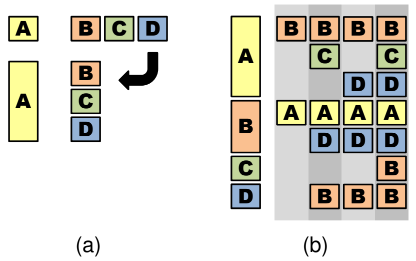


Fig. 10: Gantt layout. (a) It is possible to subdivide the row for a single node, so that every link has its own sub-row. (b) In this case, if a single link exists for several time steps, it is visually connected through these time steps. The result is a visualization similar to Gantt charts, displaying the time spans of the different links.

Finally, it is quite difficult to track a single link over time. Its relative position on the link axis can vary in every time step due to links appearing or disappearing. Hence, at least parts of the link list have to be scanned in each time step. This can be even more difficult if links are only identified by colors. Therefore, we have developed another layout for sparse and dynamic graphs described in the following.

4.2 Gantt Layout

With the above approach, dynamic graphs can be displayed and analyzed, but certain tasks are still time-consuming. This is especially the case for tasks related to the time span of links because all links of an individual node are horizontally stacked. Hence, deciding if a link exists at a specific time step requires scanning all links of the respective node.

To address this problem, a variant similar to Gantt charts can be created for sparse graphs (Figure 10), which exploits the flexibility discussed in Section 3.3. As already mentioned, the size of the individual rows can vary. Therefore, it is possible to vertically stack all links of an individual node inside its respective row (Figure 10(a)). Drawing the time steps next to each other, time spans are represented by bars that result from the merging of neighboring node elements (Figure 10(b)). Time grows from the center to the left (incoming links) and to the right (outgoing links).

With this visualization scheme, time spans can be easily analyzed even in graphs with many time steps and it is possible to detect temporal clusters, i.e., links with identical time spans (see Figure 14(b)). The layout scales well with the time discretization. It does not even require time discretization, it is possible to represent time spans on a continuous time scale. Another interesting property is that the height of a row corresponds to the number of links summed up over time. Hence, we can see which nodes have a high number of links over the full time range of the data.

However, this representation requires usually more rows, depending on the number of links per node. It is therefore only suitable for sparse graphs like the visualization workflow shown in Section 6.3. There, the space requirements change like this: The normal layout requires 61 rows (one for every node) and 529 columns for incoming and outgoing links. Most of the time steps require more than one column because at least one node exhibits multiple links. The Gantt layout requires 121 rows because the nodes with multiple links require multiple rows. However, only 252 columns are required, twice the number of time steps because incoming and outgoing links are shown. Furthermore, in the case of the normal layout, we use some extra space in the columns to visually separate the individual time steps. Hence, for this dynamic graph, the Gantt layout requires twice the space in vertical direction but less than half of the space in horizontal direction (compare Figures 17 and 18).

Other issues are that representing weights by scaling the node elements in x-direction reduces the effect of visual fusion between neighboring time steps. Furthermore, the direction of increasing time is different for incoming and outgoing links.

5 User Study

We conducted an objective and quantitative laboratory user study to compare our technique with two standard graph visualization techniques for four different tasks and two data set sizes. We used a within-subjects study design based on repeated measures. Separately for each task, two independent variables are of interest in the study: the visualization technique and the graph size.

5.1 Stimuli and Tasks

We compared three different graph visualization techniques: visual adjacency lists, adjacency matrices, and node-link diagrams. Four different tasks for two graph sizes (small: 8 nodes, 22 to 40 links; large: 20 nodes, 147 to 264 links) were investigated:

- **Task 1:** Decide if a link exists between the two marked nodes.
- **Task 2:** Decide if incoming or outgoing links are more equally distributed over the nodes.
- **Task 3:** Select the node, where the weights of its incoming links cover the largest value range.
- **Task 4:** Select the node, where the weights of all incoming links have a large increase between two subsequent time steps.

The first two tasks were performed on static unweighted graphs, the third on static weighted graphs, and the last task on dynamic weighted graphs. According to the task taxonomy for graphs by Lee et al. [43], the first two tasks are topology-based, whereas the last two tasks are attribute-based. Therefore, our selection covers typical and representative tasks on graphs. Furthermore, all tasks except for the first one can be seen as overview tasks that require the exploration of the complete graph structure. These tasks were chosen to test the suitability of

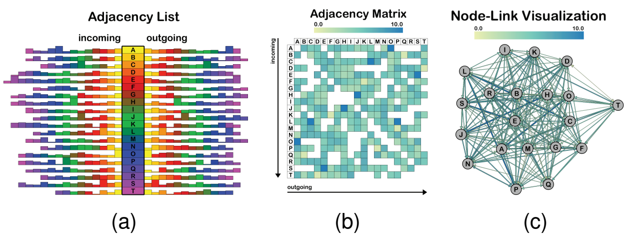


Fig. 11: Examples of stimuli of the user study for a graph from Task 3: (a) visual adjacency list, (b) adjacency matrix, (c) node-link visualization.

our approach for tasks related to the connectivity and weights of graphs. Furthermore, we wanted to evaluate its suitability for time-discretized data. We have chosen time-discrete data because it is generic (it can be obtained even from time-continuous data by temporal sampling) and it is also supported by previous dynamic graph visualization techniques based on node-link and adjacency matrix diagrams. Our hypothesis is that visual adjacency lists are more suitable for Tasks 2–4 due to their asymmetric mapping, and that especially dynamic graphs benefit. However, we wanted to perform a general evaluation and test our approach also for static graphs.

Since we wanted to keep the overall study duration around one hour to avoid boredom and fatigue, we were only able to test a small set of tasks. Besides Task 1, tasks were chosen that we expected to be suitable for visual adjacency lists to test our hypothesis that there are tasks that benefit from our approach. The goal of the study was not to provide a general in-depth comparison of our approach with adjacency matrices and node-link diagrams.

The reason for using rather small graphs is that we wanted to avoid the use of interaction techniques. Interaction would introduce another degree of freedom in the study design and it is not trivial to select adequate interaction techniques for a fair comparison. Furthermore, we first wanted to test the capability of our method for small graphs before doing the next step and going to large graphs. We used dense graphs in favor of adjacency matrices to obtain more general results. Since the layout of visual adjacency lists benefits from a low number of links, the respective results for sparse graphs should not be worse.

Every task had to be performed on every combination of technique and graph size. The task order was fixed. The order of techniques and graph sizes were counterbalanced to avoid learning effects. For every task, a set of 10 different graphs were randomly generated for both graph sizes. Five different graphs were randomly selected from 9 graphs of the set; the tenth graph was used for the test cycle. Hence, all visualization techniques were applied to the same data. Overall, 120 stimuli were tested for each participant.

Figure 11 shows examples of the visualization stimuli. Adjacency lists use the vertical mapping (Section 3.4) to represent weights, adjacency matrices the

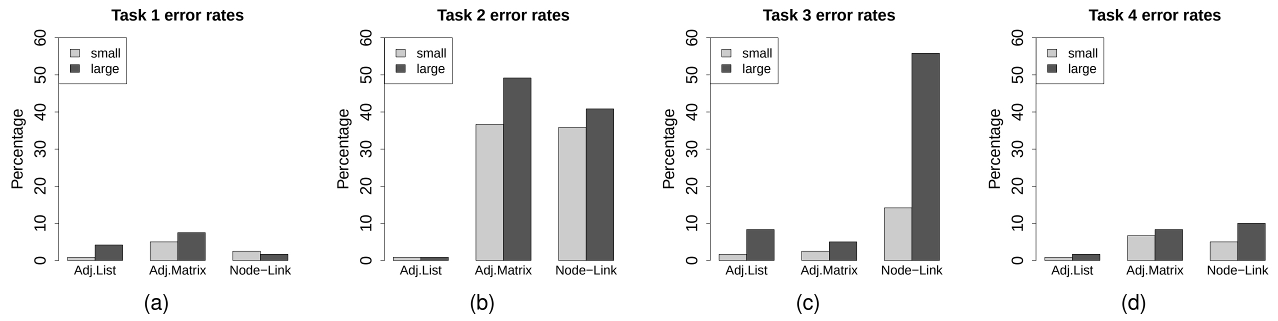


Fig. 12: Error rates from the user study. Bars are grouped according to the graph size to show the scalability of the methods.

linear color map shown, and the node-link visualization the line thickness of links in addition to the color map. Links in the node-link visualization are sorted according to their weight before drawing; links with smallest weights are drawn last on top of the others. This improves the detection of low weighted links. In the case of dynamic graphs, the adjacency list visualization uses the discrete time layout (see Section 4.1). The other two techniques showed the individual time steps from left to the right next to each other as small multiples. The node-link layout was generated with the Fruchterman-Reingold algorithm [44] implemented in the Open Graph Drawing Framework [45]. To allow for a fair comparison, we removed the labels for the links in the visual adjacency lists. Furthermore, we designed weight related-tasks in favor of the node-link visualization by restricting them to incoming links. Hence, the participants could concentrate on the arrowheads of the link representation.

5.2 Participants

26 students from our university participated. 85% of them had a major in computer science, sciences, or engineering; 77% had knowledge of graphs, but only 38% had experience with visualization. They were compensated with 10 € each.

5.3 Study Procedure

First, the participants had to fill out a questionnaire about their personal background, then they were tested for visual acuity (Snellen chart) and color blindness (Ishihara test). Short introductions to graphs and the visualization techniques used were given, before the participants had to solve the tasks. Each upcoming task was explained to the participants. Then, they performed a test cycle with the tool for one example graph for each graph size and visualization technique. The correct answers were displayed during the test cycle. When the participants confirmed that they understood the task, they continued with the measured task cycle. They had to click a button to activate the next graph visualization and time measurement. Once they had found the answer, they clicked the button again. Then, the graph visualization disappeared, time measurement stopped, and buttons to input the answer appeared. Thus, finding the correct answer button did not influence the time measurement. The participants were asked to solve all tasks as accurately as possible; the answer time serves only as a secondary dependent variable. After completing all four tasks, the participants filled out a further questionnaire, in which they rated the suitability of the different techniques for the different tasks on a Likert scale ranging from 1 (good) to 5 (bad). The full study required around one hour on average.

5.4 Results

We removed the data of one color-blind person because color was used for all techniques. However, looking at the results, it is interesting to see that this participant could solve all tasks with visual adjacency lists correctly. We also removed data from a participant with low visual acuity. Hence, the results from 24 participants were analyzed.

The measured error rates, which are the percentage of wrong answers, are shown in Figure 12. The Shapiro-Wilk test rejected normal distribution for all data and the Kruskal-Wallis test revealed that significance ($p < 0.05$) between techniques occurs only in Task 2 for small and large graphs (Figure 12(b)), in Task 3 for small and large graphs (Figure 12(c)), and in Task 4 for large graphs (Figure 12(d), dark bars). According to the posthoc Wilcoxon rank-sum test ($p < 0.05$), adjacency lists provide significant lower error rates for both sizes in Task 2; there is no significant difference between adjacency matrices and node-link visualization. In the case of Task 3, there is no significant difference between adjacency lists and matrices, but node-link visualization has significantly higher error rates, again for both sizes. Finally, for the large graph in Task 4, node-link visualization has significantly higher error rates than adjacency lists, but there is no difference between the other pairings. We can see that adjacency lists always have an error rate below 10%, in most cases even below 5%, and they exhibit the lowest error rates of all techniques in many cases. Especially for Task 2, where the distribution of the number of links has to be analyzed, only adjacency lists provide a visual representation to

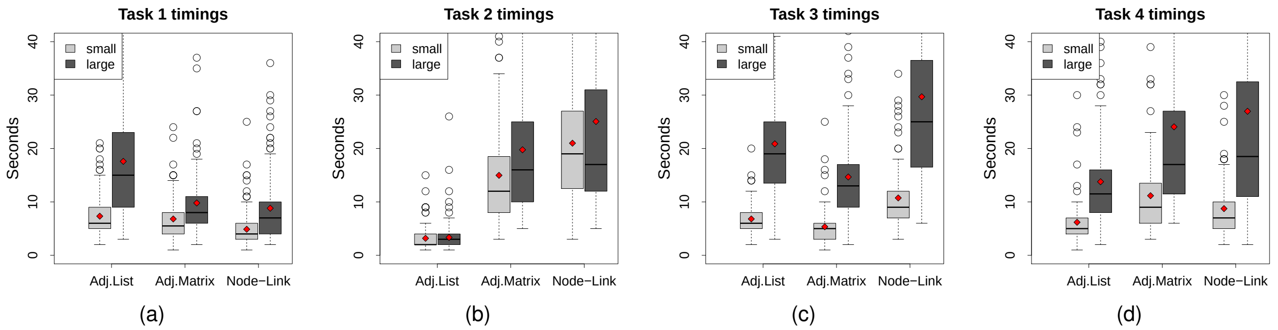


Fig. 13: Answer times (in seconds) from the user study shown with box plots; red dots represent average values.

accurately solve the task. In this case, the graph size also had no measurable influence on the accuracy.

The measured answer times are shown in Figure 13. They do not exhibit a normal distribution according to the Shapiro-Wilk test. The Kruskal-Wallis test shows a significant ($p < 0.05$) difference in all cases; posthoc pairwise comparison was then performed with the Wilcoxon rank-sum test. A significant ($p < 0.05$) difference between all techniques occurs in: Task 2 small graph (Figure 13(b), light boxes), Task 3 small and large graph (Figure 13(c)), Task 4 small graph (Figure 13(d), light boxes). In the case of Task 1 large graph (Figure 13(a), dark boxes), Task 2 large graph (Figure 13(b), dark boxes), and Task 4 large graph (Figure 13(d), dark boxes), there is only a significant difference between adjacency lists and the other two techniques. For the small graph in Task 1 (Figure 13(a), light boxes), only the answer times of the node-link visualization are significantly faster. While adjacency lists have the slowest answer times in Task 1, where the existence of a link between two nodes had to be determined, they lead to faster answer times than the other techniques in the case of Tasks 2 and 4. It is also interesting to see that the answer times with adjacency lists did not increase for the large graph in Task 2.

5.5 Discussion

The study shows that visual adjacency lists can compete with adjacency matrices and node-link diagrams for certain tasks and thus confirms our hypothesis. They are especially suitable for the dynamic weighted graph scenario in the study; we therefore focus on this scenario in our case studies (Section 6). They also provide the only visual representation for a fast and accurate completion of the second task related to the distribution of the number of links. Furthermore, it is also interesting to see that adjacency lists exhibit low error rates in the first task despite their color-based link representation. Another surprising result is that adjacency matrices enable the fastest answer times for the third task related to weights. Finally, the results show that the participants required only a short explanation of the concept of adjacency lists to perform all tasks with good accuracy. Hence, the concept seems to be suitable for practical usage.

The results are also reflected by the ratings of the participants. They rated visual adjacency lists on average between 1 and 2 for Tasks 2 to 4. Only for Task 1, where the existence of links between nodes had to be determined, the average rating was around 3. All ratings from the questionnaire are documented in detail in the supplemental material for this paper.

6 Case Studies

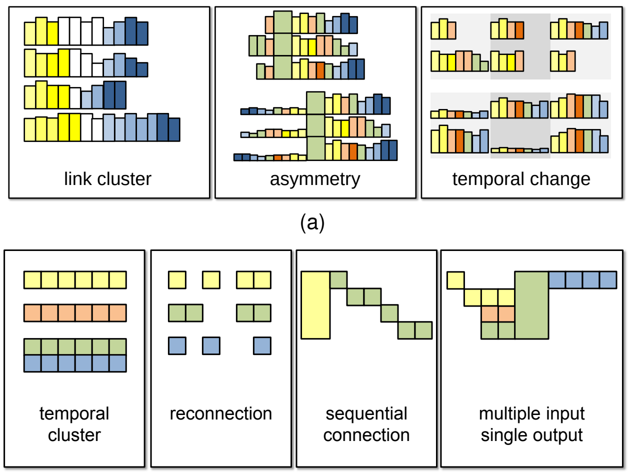
We compare our technique in two case studies with adjacency matrices as a common visualization technique for graphs. To support the comparison, we start with a description of some of the possible visual patterns in our visualization. Additionally, we provide a detailed step-by-step analysis of the data, further results, and additional data sets for the second case study in the supplemental material for this paper.

6.1 Visual Patterns

Analyzing the graphs requires the detection of visual patterns and signatures. Besides clusters in general [43], typical patterns in dynamic data include trends, periodicities, shifts, and anomalies in the graph structure over time [5].

Figure 14 shows some of the important patterns that can occur in visual adjacency lists. For the normal layout (Figure 14(a)), we exemplify patterns for the vertical weight mapping (see Section 3.4): Clusters are visible as patterns of nearby colors of the color map (see also Figure 5). Asymmetries with respect to incoming and outgoing links can appear in the link distribution or the weights of links. Finally, links and their weights can vary over time in dynamic graphs. For horizontal weight mapping, patterns known from bar charts can appear, e.g., outliers with very large or small bars (see, e.g., Figure 15(b)). However, for a strongly uneven weight distribution, the link structure is less visible than for vertical weight mapping.

The Gantt layout (Figure 14(b)) emphasizes temporal dynamics of the graph. Therefore, most patterns result from changes in the graph. Links with equal time spans create a temporal cluster. Links can recur after they disappeared and a single node can also have a sequence of links to different nodes. Finally, a single



(b)

Fig. 14: Typical patterns in visual adjacency lists with (a) normal layout and vertical weight mapping, and with (b) Gantt layout.

node may have multiple incoming links but only a single outgoing link and vice versa.

6.2 Migration Data

The first dynamic graph represents the worldwide migration of people. This is a dense and weighted graph with 226 nodes, 34,968 links on average per time step (16,485 links minimum, 45,449 links maximum), and 5 time steps. Every time step contains the migration rates for a decade, beginning from 1960. The weights cover a huge value range, from 1 to 9,367,910.

Migration data can be used to draw conclusions about political and social conditions. In this context, trends and anomalies in the data are of interest. In Figure 15(a), we can see, e.g., that Belarus, Ukraine, and the Russian Federation had immigration only from a few different countries in the early decades. This changed in the last decade for Ukraine and the Russian Federation. A reason for this could be the end of the Cold War and changes in the politics of these countries. Such a connectivity analysis can be better performed with the vertical weight representation (see Section 3.4) because it is less affected by the weights. We can, e.g., detect cluster patterns in the marked area (black box) that are asymmetric with respect to incoming and outgoing links. A red cluster exists for the outgoing links. This cluster is missing for incoming links; only in the last time step, a smaller red cluster appears. Hence, for this part of Asia, African countries were more an emigration target than an immigration source. Furthermore, outgoing links are quite constant over time, whereas the structure of incoming links changes much. These countries as well became more attractive for immigration over the decades.

The visual adjacency list with horizontal weight mapping (Figure 15(b)) provides a view on aggregated weights and is strongly influenced by the high value range of the weights. Therefore, it allows only the analysis of the countries with high migration rates. For example, we can see a cluster structure in the marked area: this part of Europe had immigration mainly from other European countries. Furthermore, we can compare the migration rates and analyze their trend, e.g., immigration into the U.S. is highest, especially at later time steps. We can also see due to the color mapping that the distribution of the home countries was changing and that Mexico dominated in the last decade. Another example is the tremendous increase of emigration from Bangladesh in the third decade, mainly to India.

Zooming-in provides a detailed view on the migration rates of single countries. Vertical weight mapping reveals, e.g., the following details: The immigration from Asia to the Netherlands in this time step was mainly from Indonesia, whereas the immigration from European countries was more uniformly distributed (Figure 15(d)). Spain had no immigration from Asia in the first decade (Figure 15(e)). Moreover, Morocco was the only African country from which people immigrated to Spain; their number is comparable to the number of immigrants from Germany. With horizontal weight mapping, we can see that Algeria had higher emigration than Egypt in the beginning (Figure 15(f)); this changed in the last two decades. The color mapping additionally shows that people from Algeria emigrated mainly to Europe, while Asia was the primary destination of emigration from Egypt.

The adjacency matrix (Figure 16) provides insights into the connectivity of the graph and clusters in the link structure. For example, clusters along the diagonals can be seen, i.e., much migration occurred inside continents. However, it is difficult to sum up or compare weights. The usage of color for weight encoding allows us only to detect larger differences between weights. Although, e.g., the high immigration rates of the U.S. are visible, it is hard to extract that this is the number one immigration country and their immigration rates increased much over the years.

In summary, visual adjacency lists provide different views on migration rates, including their temporal evolution. With horizontal weight mapping, a view on aggregated migration rates is provided, whereas vertical weight mapping allows a closer look on the country distribution. With adjacency matrices, it is difficult to obtain the total migration of single countries or track the changes in migration rates over time.

6.3 Visualization Workflow

This case study analyzes a visualization workflow created and modified in VisTrails [47]. VisTrails allows users to build custom visualizations by combining modules in a visualization workflow. Since the focus of VisTrails lies on provenance, all user modifications on the workflow are logged and stored. We have extracted the evolution of such a visualization workflow as a dynamic graph. Our example is a sparse

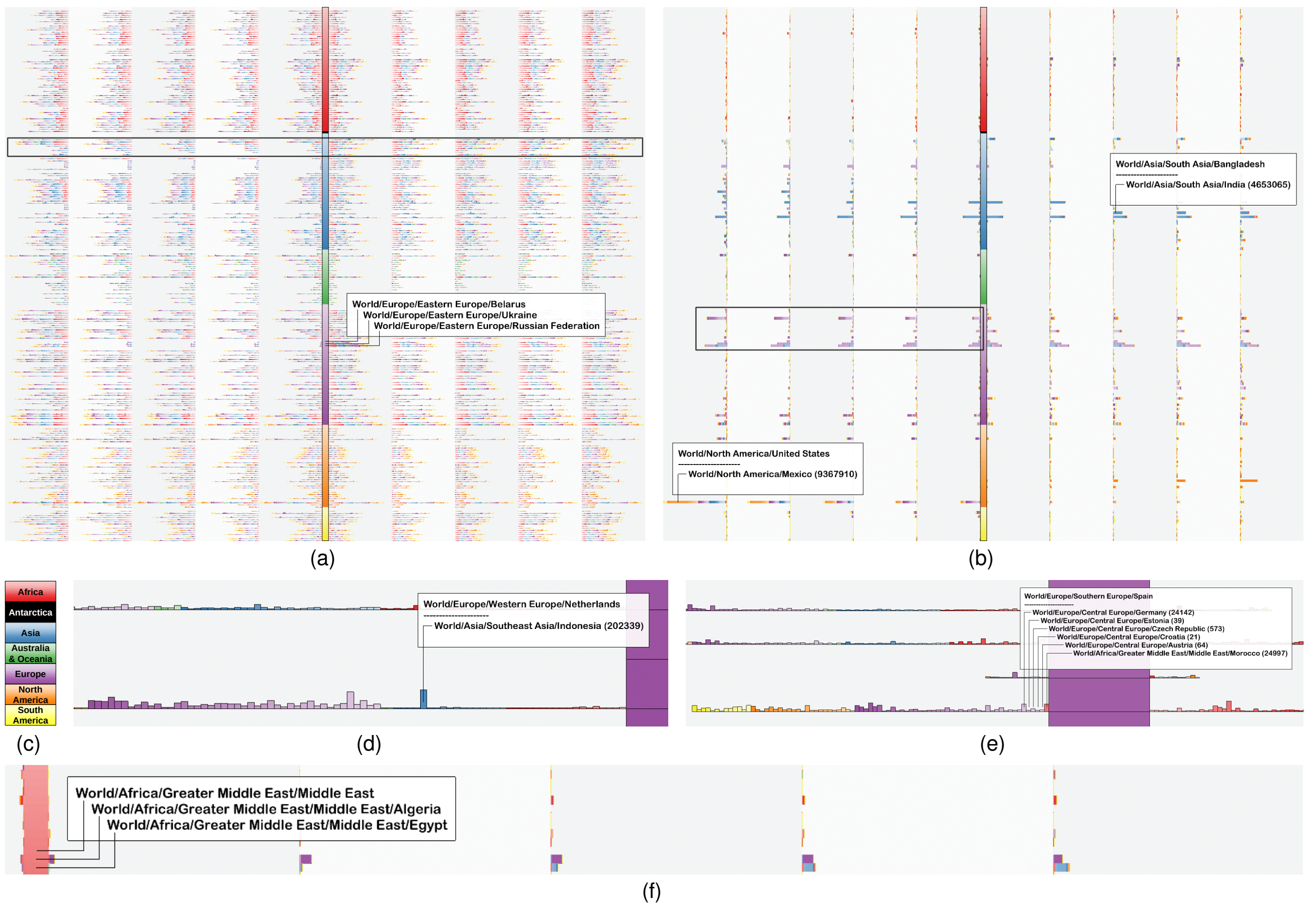


Fig. 15: Visual adjacency lists for the migration graph. The graph can be visualized with (a) vertical weight representation or (b) horizontal weight representation. (c) A hierarchical color map is used, with different colors for the continents, and changing brightness and saturation for the countries of the same continent. (d)–(f) Zooming-in allows us to analyze and compare the migration rates of individual countries. The horizontal weight mapping is linear, whereas the vertical weight representation uses a non-linear mapping. The shown info boxes are displayed on mouse interaction, the numbers in brackets are the weights of the links (in this example the number of migrating people).

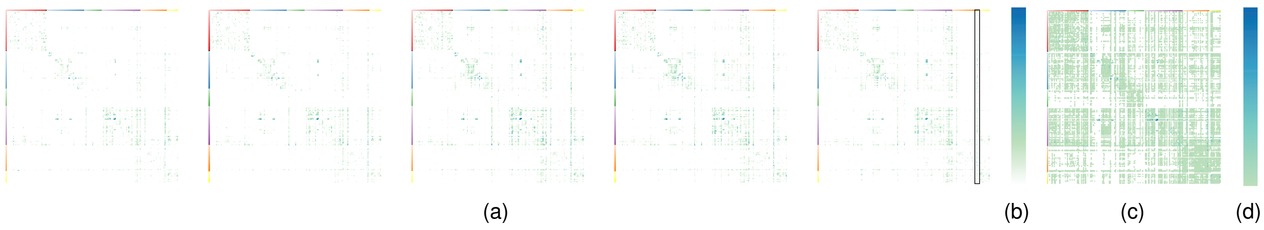


Fig. 16: Adjacency matrices for the migration graph. (a) The five time steps are shown from left to right with (b) a color map filtering out low weighted links. The immigration rates of the U.S. are marked in the last time step. (c) The first time step is shown with (d) the second color map. Low weighted links are not filtered and a better view on the connectivity is provided. The same non-linear weight mapping is used as in Figure 15. The color maps were taken from the set of sequential, colorblind safe, and print friendly color maps of ColorBrewer [46].

graph with 61 nodes, 33 links in average per time step (2 links minimum, 53 links maximum), and 126 time steps. Every time step represents a modification of the workflow by the user. In this example, the user created and modified a combined visualization of a 3D scalar and a 3D vector field.

Analyzing the link structure, which represents data flow in this context, provides insight into user behavior and the architecture of the underlying system. Persistent links, e.g., build the backbone of the custom visualization and temporal clusters indicate a direct relation of links and the respective modules. Trial and error behavior of the user typically results in shortly existing links and applying the visualization to different data sources creates a sequential link pattern.

Applying the visual adjacency list with normal layout to the dynamic graph of the visualization workflow results in a visualization with large horizontal extent due to the large number of time steps (Figure 17). Therefore, displaying the entire data set at once provides only an overview of the graph and its connectivity. For example, we can see that this is a very sparse graph, most modules have only one or two connections. Patterns and outliers can be detected, e.g., the filtering module is connected only to a single module for a rather short time range, but analyzing individual time steps is difficult. This requires zooming-in and allows us to see, e.g., the

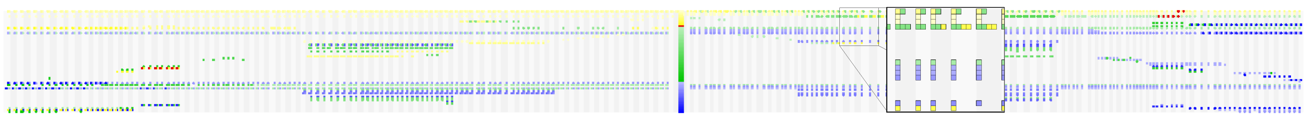


Fig. 17: Visual adjacency list for the visualization workflow. Node order and color coding consider the visualization pipeline (see Figure 18(b)). An example area is shown in an enlarged display.

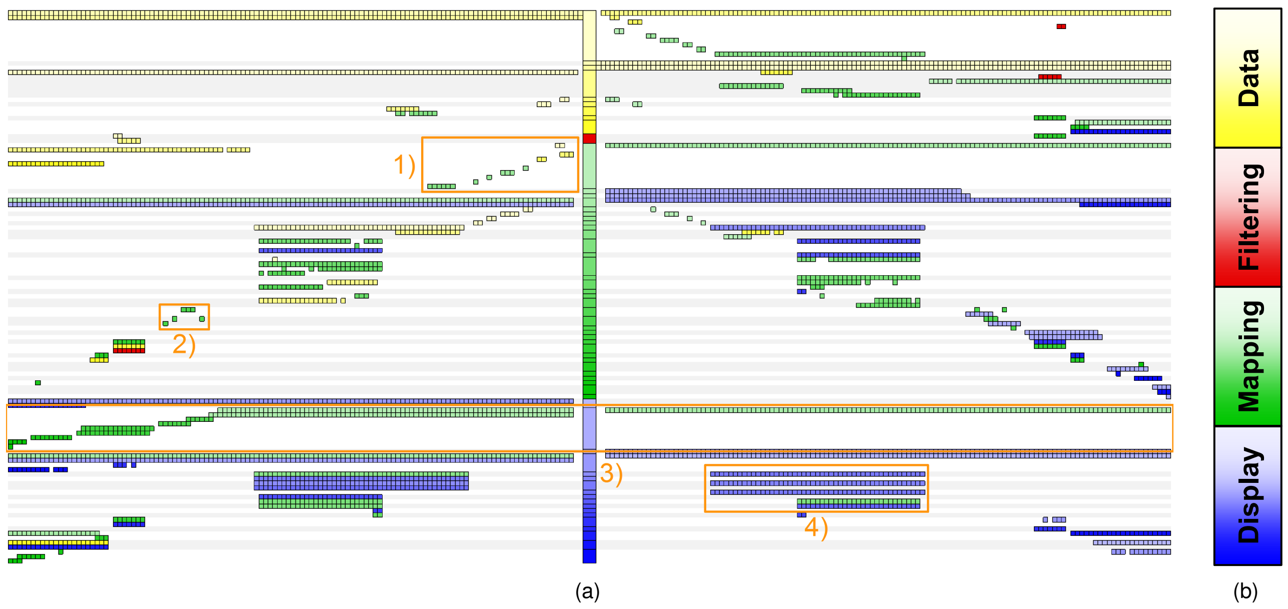


Fig. 18: Visual adjacency list with Gantt layout for the visualization workflow. (a) The full data set is visualized. Areas of interest are marked with numbers. (b) The nodes are ordered and color coded with respect to their classification in the visualization pipeline by Haber and McNabb [39]. Nodes of the same classification are ordered according to the time the respective modules are created and the brightness of their color is changed to visually separate them.

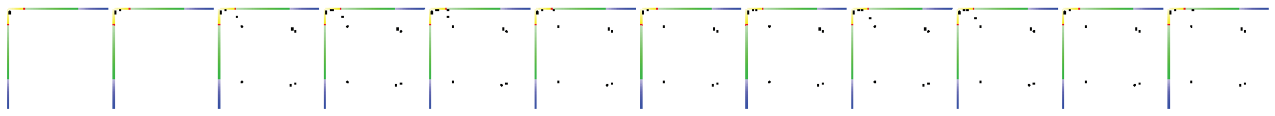


Fig. 19: Adjacency matrices for the visualization workflow. The image shows the first 12 of 126 time steps of the data set from left to right. The axes are ordered and colored in the same way as the node axis of the visual adjacency list (Figure 18).

minimum and maximum number of links as shown in the enlarged area. However, it is still difficult to determine the time spans of links because they are not aligned and can only be separated by their color.

The Gantt layout results in a more compact visualization (Figure 18). It does not only provide an overview of the entire data set but also shows details without zooming, e.g., temporal clusters and outliers can be detected. Furthermore, the time spans of links and the temporal evolution of the connectivity can be analyzed. We can see, among other things, the following interesting areas marked in the visualization:

1) The sequential connection pattern. The user probably tried out different modules as data source for this module.
2) These links exist only for a few time steps and the reconnection of a single link is visible. This can indicate trial-and-error behavior of the user.
3) A sequence of connections and partly multiple inputs, while there is only a single outgoing link for almost the full time range. Hence, this module processes data from changing sources but always provides the results to the same module. The module seems to be important for the basic visualization setup.
4) Two temporal clusters; the upper cluster consists only of links from the category “display”. These modifications of the visualization persist for longer time spans and involve several modules.

In contrast, the adjacency matrix visualization has high spatial requirements in a side-by-side view (Figure 19). It is therefore hard to obtain an overview of a large time range. Hence, time-related tasks, e.g., analyzing time spans and detecting temporal clusters, are difficult. However, the matrix representation has its advantages for the spatial analysis of the link structure and the detection of spatial clusters.

In summary, visual adjacency lists with normal layout can provide an overview of the visualization workflow, but a detailed analysis requires zooming. The Gantt layout has a time-aligned representation of links and therefore improves the analysis of time spans and the detection of temporal clusters. The adjacency matrix is more suitable for the analysis of the spatial link structure than for a temporal analysis.

7 Discussion

The following discussion is based on the results of our user study, the case studies, and general criteria like scalability, compactness, and visual clutter [12]. Table 1 summarizes the discussion for key aspects.

7.1 Advantages

First, even though color coding for node correspondence is perceptually less accurate than geometric visual mapping by lines (i.e., links in node-link diagrams), some tasks benefit from the asymmetric mapping of links as the user study and case studies show. This is the case for tasks with asymmetric characteristics, i.e., tasks where it is not important to identify target nodes, like Tasks 2–4 in the user study.

Second, like adjacency matrices, our visualization is not cluttered: there are no overlapping visual elements. Generating the layout does neither require high computational effort nor complex algorithms. This can be the case for other graph visualization techniques such as node-link diagrams.

Furthermore, the asymmetric mapping of links results in many cases in a more space-efficient visual representation than adjacency matrices. Especially in the case of sparse graphs, our approach results in a very compact visualization (see Section 6.3). This does not only improve the scalability but also allows a faster recognition of certain aspects of the graph, e.g., which nodes have the largest or smallest number of links. The reduced space requirements are also advantageous for the visualization of dynamic graphs.

In the case of weighted graphs, using the size of the node elements to represent weights (see Section 3.4) allows easy comparison of weights, similar to bar charts—and unlike adjacency matrices. Furthermore, if the weights are represented by the width of node elements, the sum over all weights of a node is directly visible. Our user study (see Section 5) confirms the suitability of adjacency lists for tasks related to weights, especially in the case of dynamic graphs.

Our approach can directly handle multiple occurrences of the same link (multigraphs, see Figure 20). Node-link diagrams and adjacency matrices have to be extended for this case.

Visual adjacency lists can be easily transformed into an adjacency matrix visualization and vice versa (see Figure 2). When both methods are used, the transition between them can be shown, e.g., by smooth animation. This may provide further insight or support the understanding of the visualization.

TABLE 1: Comparison of graph representations.

Aspects	Adj. List	Adj. Matrix	Node-Link
asymmetric tasks	+	o	-
clutter-free	+	+	-
space-efficient	+	o	o
weight encoding	+	o	-
multigraphs	+	o	o
cluster detection	o	+	+
following paths	-	o	+
dense graphs	-	+	-

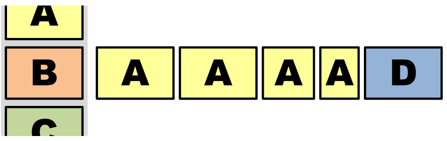


Fig. 20: Handling of multigraphs. The method can be directly applied to multigraphs: multiple links (links from "B" to "A" in this example) appear with multiple node elements.

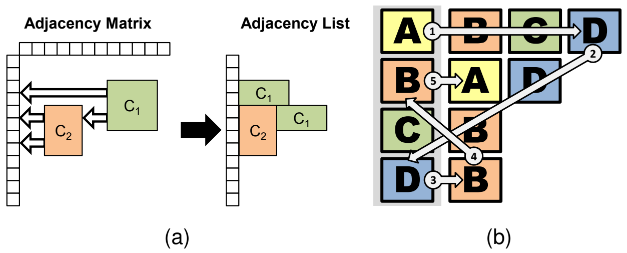


Fig. 21: Issues with cluster detection and following paths. (a) The clusters C_1 and C_2 are both clearly visible in the adjacency matrix (left). In the corresponding adjacency list (right, see Figure 2), only C_2 remains clearly visible. Cluster C_1 is distorted because the lower part is "blocked" by C_2. (b) Following a path in the adjacency list requires one to jump between the link lists and the corresponding positions on the node axis. In this example, a path from "A" to "A" over "D" and "B" is followed.

Some of the properties of visual adjacency lists can be derived from the underlying concept of adjacency lists as data structures. The complexity of graph-related tasks [43] can be derived to some extent from the computational complexity of adjacency lists as memory layout. Topology-based tasks related to direct connections are linear in the number of links of the respective nodes. In the case of adjacency matrices, these tasks are linear in the number of nodes, i.e., the maximum number of possible links. The same holds for attribute-based tasks related to links. Attribute-based tasks related to nodes are linear in the number of nodes for both adjacency list and matrix.

7.2 Disadvantages

Our approach also exhibits several drawbacks. While some tasks benefit from the asymmetric mapping of links, there also tasks that become more difficult.

The detection of clusters (see Figure 21(a)) and other graph structures can be difficult. Especially in large graphs, the color coding may hinder the recognition of nearby clusters and small structures (see Figure 5). One approach to reducing this problem may be a hybrid representation exploiting both concepts, adjacency lists and matrices (see Figure 2). For example, clusters can be shifted closer to the node axis as long as they do not break up.

Next, browsing tasks or topology-based tasks related to indirect connections [43] that require one to follow paths are time-consuming (see Figure 21(b)). Our user study shows (see Section 5) that tasks on direct connections can already be quite time-consuming.

One cannot directly go to a connected node, like it is the case in node-link diagrams; the node axis has to be scanned for the respective entry. Following several links requires one to jump between the node axis and the link list back and forth. There are similar problems in visualizations with adjacency matrices [31]. The usage of adequate interaction techniques can improve the handling of paths, e.g., by highlighting possible paths after selecting a specific node.

In the case of large graphs, adjacency lists provide only an overview of large scale structures (see Figure 4); small scale details are hardly visible. Aggregation (see Figure 6) and interaction techniques can alleviate these issues.

Similar to adjacency matrices, the ordering inside the node and link axes is flexible and impacts the result. Adjacency lists are even less restrictive than matrices: the ordering along the link axis does not have to be identical in each row. This flexibility allows us to better adapt the visualization to the tasks and data (Section 3.3). However, more parameter adjustment may be required to create good visualizations.

7.3 Application Guidelines

For sparse dynamic graphs, the Gantt layout is typically a better choice due to its representation of time spans. For all other cases, the normal layout is recommended. In the case of weighted graphs, horizontal weight mapping is good for tasks where large weights or the sum of all weights are in the focus. If the connectivity of the graph is of interest or the weights of individual nodes should be analyzed, vertical weight mapping is recommended.

8 Conclusion and Future Work

Our concept can be applied to graphs with any kind of characteristics, but mainly graphs (e.g., sparse dynamic graphs like workflows) and tasks (e.g., analysis of link distribution) with asymmetric characteristics benefit from it as the results of our user study show.

Like the underlying concept of adjacency lists, our visualization is space-efficient. This results in a compact graph representation without the need for a complex layout algorithm. Furthermore, the representation of links requires only one spatial dimension. This allows a flexible usage of spatial position and extent; there are different ways to represent weights and it is also possible to create visualizations for dynamic graphs similar to Gantt charts. These variants provide different insight and help analyze trends, detect temporal clusters and outliers, and show the evolution of connectivity and weights. Hence, especially dynamic graphs benefit from the characteristics of visual adjacency lists. We therefore see them as an addition to the tool set for graph visualization.

However, it is clear that not all tasks and graphs benefit from the presented method because there are also drawbacks, e.g., regarding cluster detection and the handling of paths. Reducing these problems and other open questions belong to future work. Augmenting the link list with arrows or curves could improve path-related tasks. A hybrid representation combining list and matrix visual metaphors may improve the detection of clusters. Further research will address interaction techniques to improve the analysis of very large graphs, and extensions of the concept that help follow paths in graphs. In the case of large graphs, appropriate aggregation methods and schemes to compute the color of pixels that represent multiple nodes are of importance. Other layouts for the graph and the time steps could also be explored, e.g., radial approaches. An improved representation of weights with a large value range is also of interest.

Acknowledgments

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the *Cluster of Excellence in Simulation Technology* (EXC 310) at the University of Stuttgart.

References

[1] G. di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[2] M. Kaufmann and D. Wagner, Eds., *Drawing Graphs, Methods and Models*. Springer, 2001.

[3] R. Rosenholtz, Y. Li, J. Mansfield, and Z. Jin, "Feature congestion: A measure of display clutter," in *Proceedings of Human Factors in Computing Systems*, 2005, pp. 761–770.

[4] B. Tversky, J. B. Morrison, and M. Bétrancourt, "Animation: can it facilitate?" *International Journal on Human-Computer Studies*, vol. 57, no. 4, pp. 247–262, 2002.

[5] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf, "Parallel edge splatting for scalable dynamic graph visualization," *IEEE Trans. Visualizations and Comp. Graphics*, vol. 17, no. 12, pp. 2344–2353, 2011.

[6] J. S. Yi, N. Elmqvist, and S. Lee, "TimeMatrix: Analyzing temporal social networks using interactive matrix-based visualizations," *International Journal of Human Computer Interaction*, vol. 26, no. 11&12, pp. 1031–1051, 2010.

[7] S. Diehl and C. Görg, "Graphs, they are changing," in *Proceedings of Graph Drawing*, 2002, pp. 23–31.

[8] C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. V. Yee, "Graphael: Graph animations with evolving layouts," in *Proceedings of Graph Drawing*, 2003, pp. 98–110.

[9] G. Kumar and M. Garland, "Visual exploration of complex time-varying graphs," *IEEE Trans. Visualization and Comp. Graphics*, vol. 12, no. 5, pp. 805–812, 2006.

[10] Y. Frishman and A. Tal, "Online dynamic graph drawing," *IEEE Trans. Visualization and Comp. Graphics*, vol. 14, no. 4, pp. 727–740, 2008.

[11] Y.-Y. Lee, C.-C. Lin, and H.-C. Yen, "Mental map preserving graph drawing using simulated annealing," in *Proc. of Asia-Pacific Symposium on Inf. Visualisation*, 2006, pp. 179–188.

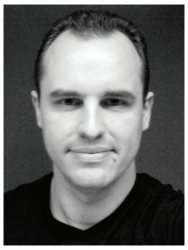
[12] F. Beck, M. Burch, and S. Diehl, "Towards an aesthetic dimensions framework for dynamic graph visualisations," in *Proc. of the Int'l Conf. on Inform. Visualisation (IV)*, 2009, pp. 592–597.

[13] C. Bennett, J. Ryall, L. Spalteholz, and A. Gooch, "The aesthetics of graph visualization," in *Proc. of Computat. Aesthetics in Graphics, Visualization, and Imaging*, 2007, pp. 57–64.

[14] H. C. Purchase, "Metrics for graph drawing aesthetics," *Journal of Vis. Lang. and Computing*, vol. 13, no. 5, pp. 501–516, 2002.

[15] C. Ware, H. C. Purchase, L. Colpoys, and M. McGill, "Cognitive measurements of graph aesthetics," *Information Visualization*, vol. 1, no. 2, pp. 103–110, 2002.

[16] K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout adjustment and the mental map," *Journal of Vis. Lang. and Computing*, vol. 6, no. 2, pp. 183–210, 1995.
[17] S. C. North, "Incremental layout in DynaDAG," in *Proceedings of Graph Drawing*, 1995, pp. 409–418.
[18] A. Sallaberry, C. Muelder, and K.-L. Ma, "Clustering, visualizing, and navigating for large dynamic graphs," in *Proceedings of Graph Drawing*, 2013, pp. 487–498.
[19] Y. Hu, S. Kobourov, and S. Veeramoni, "Embedding, clustering and coloring for dynamic maps," in *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis)*, 2012, pp. 33–40.
[20] S. Hadlak, H. Schumann, C. H. Cap, and T. Wollenberg, "Supporting the visual analysis of dynamic networks by clustering associated temporal attributes," *IEEE Trans. Visualization and Comp. Graphics*, vol. 19, no. 12, pp. 2267–2276, 2013.
[21] W. Aigner, S. Miksch, H. Schumann, and C. Tominski, *Visualization of Time-Oriented Data*. Springer, 2011.
[22] W. Aigner, S. Miksch, B. Thurnher, and S. Biffl, "PlanningLines: novel glyphs for representing temporal uncertainties and their evaluation," in *Proceedings of the IEEE Symposium on Information Visualization*, 2005, pp. 457–463.
[23] E. R. Tufte, *The Visual Display of Quantitative Information*. Graphics Press, 1986.
[24] T. Wang, C. Plaisant, B. Shneiderman, N. Spring, D. Roseman, G. Marchand, V. Mukherjee, and M. Smith, "Temporal summaries: Supporting temporal categorical searching, aggregation and comparison," *IEEE Trans. Visualization and Comp. Graphics*, vol. 15, no. 6, pp. 1049–1056, 2009.
[25] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman, "Temporal event sequence simplification," *IEEE Trans. Visualization and Comp. Graphics*, vol. 19, no. 12, pp. 2227–2236, 2013.
[26] H. L. Gantt, *Work, Wages, and Profits*. The Engineering Magazine Co., 1913.
[27] D. Archambault, H. Purchase, and B. Pinaud, "Animation, small multiples, and the effect of mental map preservation in dynamic graphs," *IEEE Trans. Visualization and Comp. Graphics*, vol. 17, no. 4, pp. 539–552, 2011.
[28] M. Greilich, M. Burch, and S. Diehl, "Visualizing the evolution of compound digraphs with TimeArcTrees," *Computer Graphics Forum*, vol. 28, no. 3, pp. 975–982, 2009.
[29] U. Brandes and S. R. Corman, "Visual unrolling of network evolution and the analysis of dynamic discourse?" *Information Visualization*, vol. 2, no. 1, pp. 40–50, 2003.
[30] S. Rufiange and M. J. McGuffin, "Diffani: Visualizing dynamic graphs with a hybrid of difference maps and animation," *IEEE Trans. Visualization and Comp. Graphics*, vol. 19, no. 12, pp. 2556–2565, 2013.
[31] 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 the IEEE Symposium on Information Visualization*, 2004, pp. 17–24.
[32] J. Bae and B. Watson, "Developing and evaluating quilts for the depiction of large layered graphs," *IEEE Trans. Visualization and Comp. Graphics*, vol. 17, no. 12, pp. 2268–2275, 2011.
[33] E. M. Kornaropoulos and I. G. Tollis, "Dagview: an approach for visualizing large graphs," in *Proceedings of Graph Drawing*, 2013, pp. 499–510.
[34] M. Burch, F. Beck, and S. Diehl, "Timeline Trees: visualizing sequences of transactions in information hierarchies," in *Proceedings of Advanced Visual Interfaces*, 2008, pp. 75–82.
[35] M. Burch and S. Diehl, "TimeRadarTrees: Visualizing dynamic compound digraphs," *Computer Graphics Forum*, vol. 27, no. 3, pp. 823–830, 2008.
[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2009.
[37] W. S. Cleveland and R. McGill, "An experiment in graphical perception," *International Journal of Man-Machine Studies*, vol. 25, no. 5, pp. 491–500, 1986.
[38] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Transactions on Graphics*, vol. 5, no. 2, pp. 110–141, 1986.
[39] R. B. Haber and D. A. McNabb, "Visualization idioms: A conceptual model for scientific visualization systems," in *Visualization in Scientific Computing*, G. M. Nielson, B. Shriver, and L. Rosenblum, Eds. IEEE Comp. Soc. Press, 1990, pp. 74–93.
[40] N. Henry, "Exploring social networks with matrix-based representations," Ph.D. dissertation, Université Paris Sud, France, and University of Sydney, Australia, 2008.
[41] I. Liiv, "Seriation and matrix reordering methods: An historical overview," *Statistical Analysis and Data Mining*, vol. 3, no. 2, pp. 70–91, 2010.
[42] S. E. Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27–64, 2007.
[43] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry, "Task taxonomy for graph visualization," in *Proceedings of BELIV workshop*, 2006, pp. 1–5.
[44] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software–Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991.
[45] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel, "The Open Graph Drawing Framework (OGDF)," in *Handbook of Graph Drawing and Visualization*, R. Tamassia, Ed. CRC Press, 2013, pp. 543–570.
[46] M. A. Harrower and C. A. Brewer, "ColorBrewer.org: An online tool for selecting color schemes for maps," *The Cartographic Journal*, vol. 40, no. 1, pp. 27–37, 2003.
[47] C. Silva, J. Freire, and S. Callahan, "Provenance for visualizations: Reproducibility and beyond," *Computing in Science and Engineering*, vol. 9, no. 5, pp. 82–89, 2007.



Marcel Hlawatsch received the Diplom (MSc) degree in computer science from the University of Stuttgart, Germany. Since 2008, he has been a PhD student at the Visualization Research Center, University of Stuttgart (VISUS). His research interests include scientific visualization, GPU methods, uncertainty, graph visualization and visualization workflows.

Michael Burch is a postdoctoral researcher at VISUS, University of Stuttgart, Germany. He received his Dr. rer. nat. (PhD) degree in computer science from the University of Trier in 2010. His main research interests are in information visualization, in particular, visualizing dynamic directed and weighted graphs with an additional hierarchical organization of the nodes in a static diagram. Other areas of expertise are the large-scale visualization of hierarchical data, eyetracking evaluation of information visualization techniques, and the visualization of eyetracking data.

Daniel Weiskopf received the Diplom (MSc) degree in physics and the Dr. rer. nat. (PhD) degree in physics, both from Eberhard-Karls-Universität Tübingen, Germany, and he received the Habilitation degree in computer science at the University of Stuttgart, Germany. From 2005 to 2007, Dr. Weiskopf was an assistant professor of computing science at Simon Fraser University, Canada. Since 2007, he has been a professor of computer science at the Visualization Research Center (VISUS) and the Visualization and Interactive Systems Institute (VIS) at the University of Stuttgart. His research interests include visualization, visual analytics, GPU methods, perception-oriented computer graphics, and special and general relativity. He is member of ACM SIGGRAPH, Eurographics, the Gesellschaft für Informatik, and the IEEE Computer Society.