An Overview+Detail Layout for Visualizing Compound Graphs

Chang Han*
The University of Utah

Justin Lieffers
The University of Arizona

Clayton Morrison†
Lum AI

Katherine E. Isaacs‡
The University of Utah

ABSTRACT

Compound graphs are networks in which vertices can be grouped into larger subsets, with these subsets capable of further grouping, resulting in a nesting that can be many levels deep. In several applications, including biological workflows, chemical equations, and computational data flow analysis, these graphs often exhibit a tree-like nesting structure, where sibling clusters are disjoint. Common compound graph layouts prioritize the lowest level of the grouping, down to the individual ungrouped vertices, which can make the higher level grouped structures more difficult to discern, especially in deeply nested networks. Leveraging the additional structure of the tree-like nesting, we contribute an overview+detail layout for this class of compound graphs that preserves the saliency of the higher level network structure when groups are expanded to show internal nested structure. Our layout draws inner structures adjacent to their parents, using a modified tree layout to place substructures. We describe our algorithm and then present case studies demonstrating the layout's utility to a domain expert working on data flow analysis. Finally, we discuss network parameters and analysis situations in which our layout is well suited.

Index Terms: compound graphs, network layout, graph drawing, ntework visualization, graph visualization

1 INTRODUCTION

Compound graphs are networks in which additional grouping information is available regarding the vertices. A group may contain other groups of vertices, leading to a hierarchical nesting of network contents [22]. These networks with grouping structures arise frequently in multi-step processes. For example, vertices relating to a biological process may be grouped into a compound node representing that process [24, 12]. Larger, more complicated processes may be composed of these compound nodes. Thus, the compound graph may have many levels of grouping to navigate.

A common way to visualize graphs is with a node-link diagram, which promotes tracing paths through the network. With compound graphs, the initial depiction of the graph may show only the higher level groupings, reducing clutter by allowing the user to interactively expand the groups to see their members [23]. Visually, the groups are often expanded in place, resulting in a layout that focuses on lower levels but can obsucre higher level structure, as shown in Figure 1(b).

For compound graphs where sibling clusters are disjoint, e.g., those with tree-like hierarchies like the multi-step process graphs discussed above, we can balance the display of both high level and low level structure through an overview+detail approach. We propose a layout (Figure 1(a)) that preserves the higher level structure while still supporting examination of lower level structure, placing the two near each other so users can simultaneously explore both the internal details of a group and how that group interacts with its peers at the higher level. We use two strategies to achieve this layout. First, we explicitly route edges that traverse group membership through "ports" that isolate the inner group layout while preserving path tracing. Second, we adapt the Flexible RT algorithm [20] to draw expanded groups near their collapsed representations within a group.

We present our contributed layout in section 3. We then demonstrate the utility of the layout through a collection of case studies in analyzing data flow graphs. We conclude with a discussion of situations in which our layout is appropriate. Specifically, we consider our layout most appropriate for directed compound networks that have multiple levels of nesting, especially when analysis tasks benefit from a strong understanding of higher level contexts along with expanded detail.
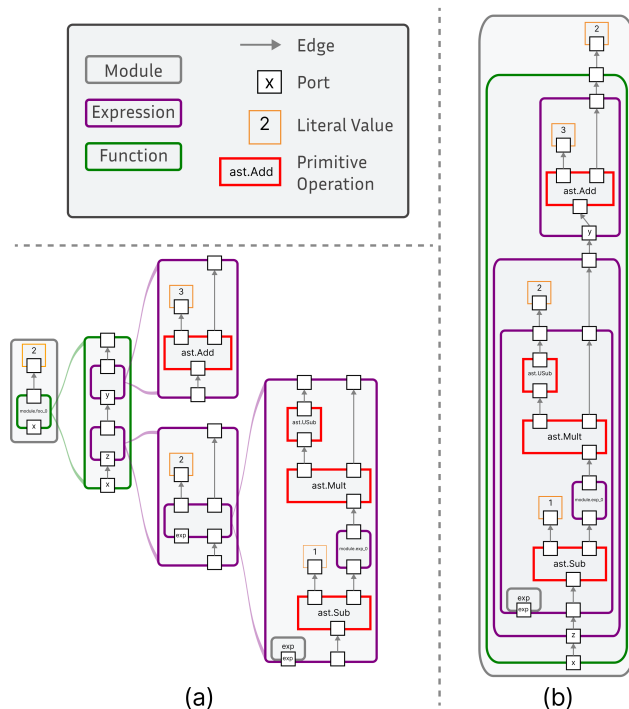

Figure 1: An overview+detail layout (a) vs. focus+context layout (b) of the same compound graph. The graph depicts the data flow of a small Python program. In our case study, a domain expert looks at compound networks of larger, more complicated programs.

2 RELATED WORK

A common way to visualize compound graphs is with Sugiyama-style algorithms [7, 10, 18, 16]. Sugiyama and Misue [18] initially proposed an algorithm to create a compound layout for directed graphs. Their method generates a layered layout within each cluster. Extending this approach, Sander [16] introduced global partitioning and then drew borders around each cluster. Despite employing various techniques to reduce edge crossings, the layout becomes cluttered as the size of the graph increases. To mitigate this

*e-mail: changhan@sci.utah.edu

†e-mail: clayton@lum.ai

‡e-mail: kisaacs@sci.utah.edu

issue, Wongsuphasawat et al.[23] applied edge bundling to compound graphs, routing edges between clusters so that they connect only nodes that are siblings within the hierarchy. This strategy also aids in maintaining layout stability during interaction, such as expanding groups [14]. However, edge bundling complicates the tracing of edges and the graph's appearance still changes significantly if an expanded group is large. Yoghourdjian et al. [25] proposed a grid layout method, designed for small compound graphs, that produces compact layouts with edges routed between grid cells.

Deriving compound graphs, such as through clustering, is a common approach for visualizing overviews of large and complex undirected graphs. Balzer and Deussen [5] proposed a level-of-detail technique for visualizing clustered graphs in both 2D and 3D. They employ implicit surfaces and edge bundling to simplify the visual representation of these graphs. Additionally, Ham and van Wijk [21] proposed a method to interactively inspect local structures of small-world graphs, while maintaining a global overview of the entire structure. Abello et al. [1] proposed an interactive system for large scale compound graphs that allows users to navigate through the hierarchies by expanding clusters. Archambault et al. [2] combined a tree view and a graph view to aid exploration of compound graphs. The tree view offers a clear representation of the graph hierarchy, while the graph view displays the current cut of the graph hierarchy, selected via the tree view. The same combined views [3] is also used to modify graph hierarchy. Our layout combines tree and graph notions, thereby providing both a clear hierarchy and the detailed graph structure within the same view.

3 PROPOSED COMPOUND GRAPH LAYOUT

The primary goal of the proposed layout is to preserve the original graph representations at each level of the hierarchy, thereby enabling viewers to understand the network at multiple levels simultaneously. Thus, we aim to have little-to-no distortion at each level, even when expanding down the hierarchy. Before introducing the algorithm, we first introduce the concept of *ports* that we have employed when visualizing compound graphs.

3.1 Ports and Inner Layout for Expanded Groups

Often in compound graphs, edges crossing into a compound node either are routed directly to their neighbor within the compound node or are represented as terminating at the compound node boundary, with the explicit connections elided, often to remove clutter. For example, TensorFlow graphs [23] bundle edges entering a compound node.

Several layouts have used a concept of 'ports' to guide the routing of edges. For example, in the *dot* algorithm implementation in GraphViz [8], ports specify which side of a node the edge should be routed to. Our layout draws explicit ports, shown in our figures as small squares (Figure 2). Edges crossing the border of a compound node are routed through ports for clarity, with one direction ("top") indicating 'in' ports and othe opposed direction ("bottom") 'out' ports. Our layout allows for any numer of ports per compound node. This choice allows us to layout the members of the compound node independently of its outer context, except for the direction in which the outer edges come (i.e., the direction of the ports).

Explicit boundary crossing ports has relevance in several domains. In computational graphs, operations (nodes) can receive data variables as inputs and produce new variables as outputs [23]. Similarly, chemical equations may have similar input and output semantics [4]. Note that our ports are agnostic to layout methods, meaning that the proposed technique is applicable to graphs both with and without an explicit presentation of ports.

We employ Dagre, a JavaScript library, to compute a Sugiyama-style layout for each expanded group [19]. We treat the ports as child nodes within the variable boxes and add dummy nodes between the input and output ports to separate them.
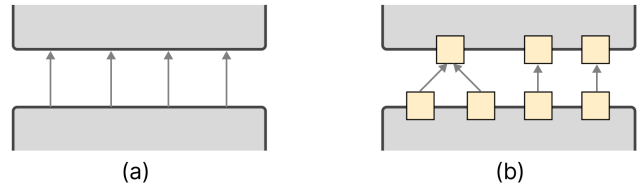

Figure 2: Compared to (a), we explicitly incorporate ports into the design in (b) to increase the clarity of links through compound graphs and isolate the the layout within compound nodes.

The selected node to expand   The expanded internal view

The selected node to expand
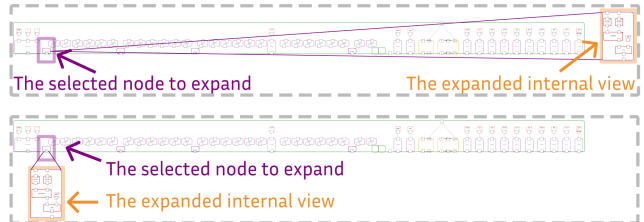The expanded internal view

Figure 3: (Top) The Reingold-Tilford algorithm can produce poor results when the compound node is oblong in the direction of tree growth. Even growing the layout vertically, the internal view would be placed in the middle of the oblong compound node rather than near its counterpart. (Bottom) The ideal layout would place expanded nodes near their collapsed counterparts.

3.2 Overview+Detail Metaphor for Compound Graphs

The overview+detail design in visualization context refers to *"the simultaneous display of both an overview and detailed view of an information space, each in a distinct presentation space"* [6]. Existing compound graph visualizations focus on "focus+context" [17, 11] and "semantic zooming" [23], both which distort the higher levels around the focus level. The zooming techniques also cause the structure of the pre-zoomed graph (the overview) to be less salient, making it difficult to maintain a sense of higher level connectivity and structure when viewing the internals of compound nodes.

To address this issue, we propose a new layout for compound graphs based on the overview+detail metaphor in interface design. This layout creates a distinct view of the subgraph at each cluster expansion, instead of expanding selected node and re-laying out the graph containing it. The nesting relationship between the original graph and the subgraphs enables us to employ tree layout methods to arrange these distinct views. However, traditional tree layout algorithms like Reingold-Tilford (RT) algorithm [15] cannot be directly applied for two main reasons: i) the boxes of different view frames have siginificant size and aspect ratio differences, and ii) placing child nodes at the center of one compound (parent) node direction, as is the RT default, may result in layouts where the collapsed and expanded representations are far from each other. Below we discuss our customized variant of the RT algorithm to address these two issues.

3.3 Orthogonal Non-layered Reingold-Tilford Algorithm

As shown in Figure 3 (Top), simply applying RT algorithm can yield poor results. Placing the expanded node far from its collapsed counterpart creates visual edge clutter and presents difficulties in simultaneously comprehending the node in both its collapsed and expanded states. Although changing the tree layout direction in RT can alleviate the issue in this case, it is not a viable solution because (1), even if centered vertically, the expanded node would still be some distance from its collapsed representation, and (2) the graph may contain elements that are optimized for left-to-right and top-to-bottom layouts seperately. Below we discuss our customized
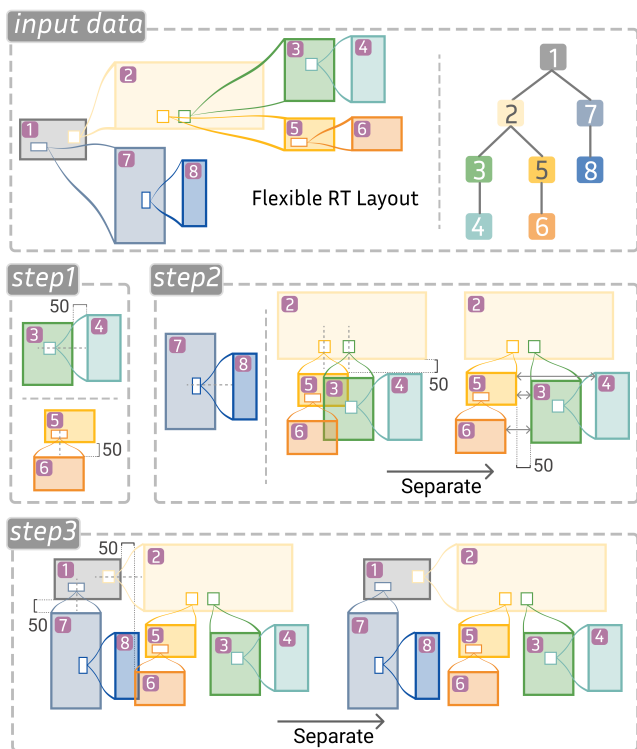
Figure 4: An illustration of our proposed variant of RT algorithm. The input data is shown in both our layout and a simplified tree view. Then in each step, we follow the RT bottom-up placement. We frist place group parents with respect to expanded children based on the position of their corresponding internal nodes, and then make separation passes in both directions of tree expansion.

variant of RT, which mitigates these issues. An implementation of this algorithm can be found on Github[1].

The primary design goal of our method is to **minimize the distance between the selected node and its expanded view**. We base our algorithm on the "Flexible Reingold-Tilford (Flexible RT)"algorithm [20], a non-layered variant of the traditional RT algorithm. Compared to the traditional RT, the Flexible RT achieves a higher space utilization rate and helps reduce edge length. We employ the non-layered RT because layers do not encode critical information in our use cases. For applications requiring a clear layer structure, traditional RT can be re-employed, albeit at the expense of space utilization and edge length.

Our proposed method adopts a bottom-up approach similar to both RT algorithms, as shown in Figure 4. The primary distinction of our algorithm is that it lays out the tree in two directions. As illustrated in Figure 4 *step1*, the children (nodes 4 and 6) are positioned to the right and bottom, respectively, based on their proximity to the corresponding boundaries of the parent boxes (nodes 3 and 5). This strategy effectively reduces the distance between the node and its expanded view.

Another notable difference from previous methods is our initialization of each child node's position. At each step, we initialize the position of the child according to its location within the parent box. For instance, in *step2*, before separation, the x-coordinates of the two subtrees (3-4 and 5-6) are set to align with their location in the parent box. Following this initialization, the algorithm checks for overlaps and applies a separation operation as necessary, utilizing
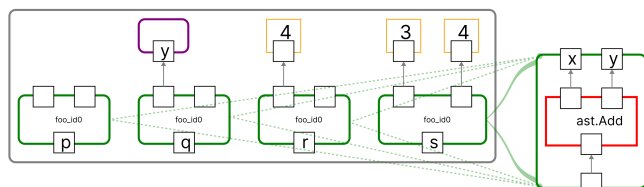
Figure 5: Duplicate structures within one group are drawn with edges from the collapsed copies to the expanded form. In this application, we choose lightweight dashed edges to depict this relationship. Users of our layout can choose to suppress these edges and/or the expansion of duplicate siblings as layout parameters..

the same separation technique as described in the Flexible RT algorithm [20]. Subsequently, the parent node adjusts to the average displacement of all child nodes, minimizing the total displacement of child nodes.

Note that this separation is performed for child subtrees that are oriented in the same direction (bottom or right). In some scenarios, as shown in Figure 4 *step3* (left), the children oriented in the left-right direction may overlap with those in the top-down direction. Because the contours of these groups can form non-convex polygons, resolving their overlap becomes a problem of finding the Minimum Translation Vector (MTV) to separate overlapping non-convex geometric shapes, a challenge commonly addressed in collision detection within computer graphics [9]. Although optimal solutions typically require complex algorithms, we find a simplified approach with linear running time to be good enough in our application: We perform the same separation operation as in *step2* but execute it twice: initially, we separate the polygons by moving them along the y-axis, and then repeat the process along the x-axis. We compare the two results and take the approach that results in the shortest total distance moved as the final choice. The parent node then adjusts to the average displacement of all child nodes. For instance, in Figure 4 *step3*, the two subtrees of node 1 are effectively separated by moving along the x-axis.

We provide two methods for drawing duplicate members of the same group. They can be individually expanded or shown as a single subgroup with additional edges. Figure 5 illustrates the latter case where all duplicate groups are connected to the expanded group with dashed lines, eliminating the need for drawing each expansion as required in the focus+context layout. This reduction of drawn elements is another benefit of the overview+detail approach.

4 CASE STUDIES

We discuss the use of our layout for debugging and understanding compound networks generated during automated extraction of computations from source code. These case studies arose from deployed use of our layout by the author who was not involved in the design of the layout or the implementation but was involved in providing feedback towards a visualization that used the layout as its main view as well as in generating the datasets we used. We provide background for the use and visualization and then discuss how it was used in practice.

4.1 Layout Application Background

Our layout was used in the context of project involving automated extraction of computational models from source code and other documents. These models can then be analyzed either computationally or manually, combined with other models, and transpiled into new, more performant code. The models are stored in a custom abstract format, known as a Function Network, that can be cast as a compound network.

The author who performed these case studies is part of the extraction team. They were interested in the visualization to help them
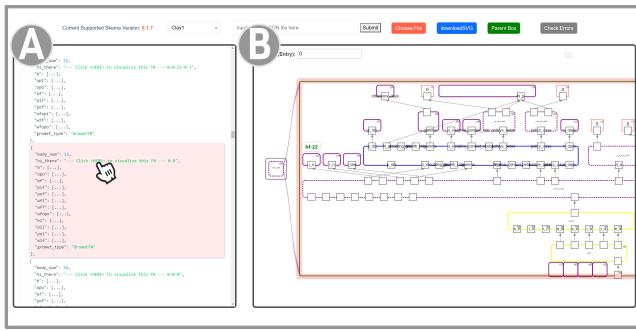
Figure 6: The interface used in the case study has two linked views. View A shows the raw JSON representing the Function Network. View B is the main view of our compound networks.
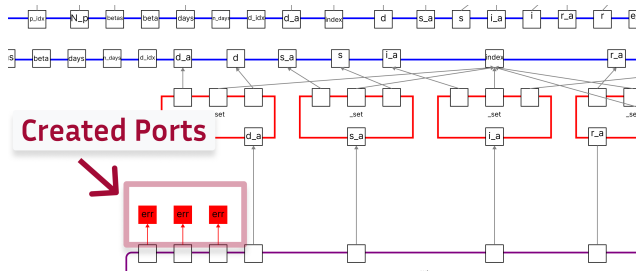

Figure 7: Undefined ports in the Function Network often signify bugs. We add them if referenced in red to make them more salient.

understand the Function Networks to help them create and debug steps further in the pipeline. As both the extraction and subsequent steps were in development, their use also led to identification of bugs in the extraction code.

The Function Networks encapsulate concepts like expressions, functions, loops, and modules as a generic 'box' that can contain other boxes. Each box can have multiple input and output ports which represent values and can have names corresponding to variables extracted from the source. Links between ports indicate movement of values. In Function Networks, arrows point to a value's source rather than its destination. This abstraction is used by other analyses in the pipeline.

To match the illustrations in the project's documentation, we use categorical color to differentiate concepts like expressions and functions. We paired our layout with an auxiliary view showing the raw JSON representing the Function Network (Figure 6). These linked views enable seeking a particular box and expand the visualization such that the internals of that box are visible.

A common bug in the Function Network extraction is referencing an undefined port. In this case, we created the port in our layout but drew it in solid red to indicate it was our creation (Figure 7).

4.2 Use of Our Layout

These case studies were extracted from brief notes the performing author wrote in a single month's summary report from the entire project. The first author followed up with the performing author for more detail. Both the report and follow up occurred during design of the multi-view visualizaiton, before publication was discussed.

Using Our Layout for Debugging. The author reported two instances in which our layout helped with debugging. In the first, they were able to determine a downstream error in running their code was caused by an upstream error in defining the Function Network. They noticed the solid red ports indicating missing definitions in the Function Network. Due to the placement in the layout,

they were able to narrow down what part of the extraction process it was associated with. They said, "With the way the ports are laid out, it's easy to tell what kind of wires these are," referring to the implicit purpose of the links.

In the second instance, the author debugged their own code which makes inferecnes about the computation performed by the Function Network. They had made an assumption about the way the Function Network represented vectors. Once they checked the visualization, they understood their error, as well as the correct representation, and fixed their code. They particularly noted the "cascading" style of the layout helped them: "If you see it, you'll see a lot of those set operations going on. It's kind of that cascading, from the [internal name] visualization."

Using Our Layout for Understanding. The author also used our layout for understanding how Function Networks represent functions as parameters to other functions. The author was aware that other team members had added new specifications regarding function parameters. They used the layout with several test cases to understand the new specifications, often tweaking the code and examining the changes. Prior to the introduction of the layout, they would rely on documentation alone, which had a few hand-drawn examples of small, isolated uses. They said, "It allowed me again, to get a better understanding of the structure and pretty quickly be able to update the code on my side."

5 Conclusion and Future Work

We have presented an overview+detail layout for compound graphs and demonstrated its utility through case studies in debugging compound graphs representing computation. The layout specifically prioritizes displaying both the structure between higher level groups as well as the detail within groups, placing the two depictions, expanded and collapsed, near each other in the overall layout.

The impetus of our layout design was aiding in tracing through compound graphs representing processes that can include nested sub-processes, such as those found in biological processes, chemical equations, and computational data flow. Thus, it is designed primarily for directed networks. Specifically, we apply a convention of placing inputs on one side and outputs on the other through our layout's explicit ports.

We recommend this layout be used when the compound structure has several levels of nesting. In shallow networks or applications where higher levels are not as important, the limitations of focus+context and/or semantic zooming approaches in showing the higher-level behavior would have little effect on the analysis. We do not claim that our method is a superior substitute for previous methods, but rather an alternative choice in scenarios where understanding multiple levels simultaneously is crucial. Ultimately, empirical studies would need to be performed to more definitively and precisely understand the suitability of these methods to various compound graph tasks.

We note our tree layout method restricts the layout direction to right and down, which may still result in some long edges in some cases when the parent group is large and many subgroups are expanded. We made this compromise in our layout design for better computational efficiency and a clearer overall direction (from top-left to bottom-right). Alternatives placement algorithms, including ones that conisder all directions, may further improve the overview+detail layout approach. Another limitation of our approach, similar to the traditional RT algorithm, is that it may produce elongated layouts. To address this, we allow users to pan and zoom within our application, as shown in Figure 6. Future work could explore methods to partially collapse elongated nodes or investigate creating area-adaptive layouts similar to Misue's approach [13], but with plural layout directions and varied node sizes.

REFERENCES

[1] J. Abello, F. van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):669–676, 2006. doi: 10.1109/TVCG.2006.120 2

[2] D. Archambault, T. Munzner, and D. Auber. Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):900–913, 2008. doi: 10.1109/TVCG.2008.34 2

[3] D. Archambault, T. Munzner, and D. Auber. Tugging graphs faster: Efficiently modifying path-preserving hierarchies for browsing paths. *IEEE Transactions on Visualization and Computer Graphics*, 17(3):276–289, 2011. doi: 10.1109/TVCG.2010.60 2

[4] P. W. Atkins, J. De Paula, and J. Keeler. *Atkins' physical chemistry*. Oxford university press, 2023. 2

[5] M. Balzer and O. Deussen. Level-of-detail visualization of clustered graph layouts. In *2007 6th International Asia-Pacific Symposium on Visualization*, pp. 133–140. IEEE, 2007. doi: 10.1109/APVIS.2007.329288 2

[6] A. Cockburn, A. Karlson, and B. B. Bederson. A review of overview+ detail, zooming, and focus+ context interfaces. *ACM Computing Surveys (CSUR)*, 41(1):1–31, 2009. doi: 10.1145/1456650.1456652 2

[7] P. Eades, Q.-W. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In *Graph Drawing: Symposium on Graph Drawing, Berkeley, California, USA, September 18–20, 1996*, pp. 113–128. Springer, 1997. doi: 10.1007/3-540-62495-3_42 1

[8] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphviz—open source graph drawing tools. In *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9*, pp. 483–484. Springer, 2002. doi: 10.1007/3-540-45848-4_57 2

[9] C. Ericson. *Real-time collision detection*. Crc Press, 2004. doi: 10.1201/b14581 3

[10] M. Forster. Applying crossing reduction strategies to layered compound graphs. In *International Symposium on Graph Drawing*, pp. 276–284. Springer, 2002. doi: 10.5555/647554.729577 1

[11] E. R. Gansner, Y. Koren, and S. C. North. Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):457–468, 2005. doi: 10.1109/TVCG.2005.66 2

[12] M. Gosak, R. Marković, J. Dolenšek, M. S. Rupnik, M. Marhl, A. Stožer, and M. Perc. Network science of biological systems at different scales: A review. *Physics of life reviews*, 24:118–135, 2018. doi: 10.1016/j.plrev.2017.11.003 1

[13] K. Misue. Area-adaptive drawing of rooted trees. In *2024 IEEE 17th Pacific Visualization Conference (PacificVis)*, pp. 152–161, 2024. doi: 10.1109/PacificVis60374.2024.00025 4

[14] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, 1995. doi: 10.1006/jvlc.1995.1010 2

[15] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Transactions on software Engineering*, (2):223–228, 1981. doi: 10.1109/TSE.1981.234519 2

[16] G. Sander. Layout of compound directed graphs. In *Technical Report A/03/96, Universität des Saarlandes, FB 14 Informatik, 66041 Saarbrücken*, 1996. 1

[17] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(2):162–188, 1996. doi: 10.1145/230562.230577 2

[18] K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892, 1991. doi: 10.1109/21.108304 1

[19] 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. doi: 10.1109/TSMC.1981.4308636 2

[20] A. van Der Ploeg. Drawing non-layered tidy trees in linear time. *Software: Practice and Experience*, 44(12):1467–1484, 2014. doi: 10.1002/spe.2213 1, 3

[21] F. Van Ham and J. J. Van Wijk. Interactive visualization of small world graphs. In *IEEE Symposium on Information Visualization*, pp. 199–206. IEEE, 2004. doi: 10.1109/INFVIS.2004.43 2

[22] C. Vehlow, F. Beck, and D. Weiskopf. Visualizing group structures in graphs: A survey. In *Computer Graphics Forum*, vol. 36, pp. 201–225. Wiley Online Library, 2017. doi: 10.1111/cgf.12872 1

[23] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mane, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):1–12, 2017. doi: 10.1109/TVCG.2017.2744878 1, 2

[24] H.-Y. Wu, M. Nöllenburg, and I. Viola. Graph models for biological pathway visualization: State of the art and future challenges. *arXiv preprint arXiv:2110.04808*, 2021. 1

[25] V. Yoghourdjian, T. Dwyer, G. Gange, S. Kieffer, K. Klein, and K. Marriott. High-quality ultra-compact grid layout of grouped networks. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):339–348, 2015. doi: 10.1109/TVCG.2015.2467251 2