A New Radial Space-Filling Visualization Approach for Planar st-Graphs

Ilir Jusufi* Andreas Kerren† Yuanmao Wang

ISOVIS Group, DFM, Linnaeus University, Växjö, Sweden

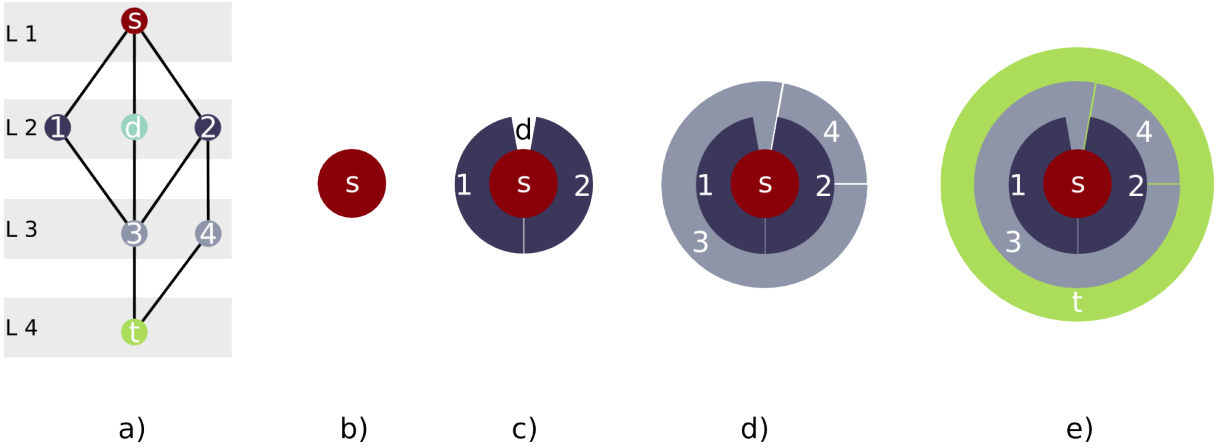

Figure 1: The node-link diagram of the input graph is shown on the left (after the initial three steps of our layout algorithm have been performed). The consecutive diagrams (b-e) show the step-by-step computation of the space-filling layout process. The source is marked with the label s, the target/sink with label t, whereas the node labeled with d is a so-called dummy node. This node is used to compute the empty space as seen in Figures 1(c) and 1(d). Note that the direction of the edges is implicitly given by the four layers (top-down).

Abstract

Planar st-graphs are used in a number of different application fields in the sciences, but also in industry. So far, mainly node-link-based layouts have been used to visualize such graphs especially in the Graph Drawing community. One drawback of these standard layouts is their high consumption of space. In Information Visualization, there exist visualization techniques for graphs which achieve considerable space savings, such as matrix-based approaches. In this work, we present a novel space-filling representation to visualize planar st-graphs.

Keywords: Network visualization, space-filling representations, st-graphs, graph layout.

Index Terms: G.2.2 [Discrete Mathematics]: Graph Theory—Graph Algorithms; H.5.2 [Information Interfaces and Presentation]: User Interfaces; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques;

1 Introduction

A planar st-graph is a directed acyclic graph that has exactly one source node s, one sink node t and admits a planar drawing, i.e., no two edges intersect [3]. These graphs are used in a variety of areas including motion planning, computational geometry, graph theory, and very-large-scale integration (VLSI) design [9, 4].

In this paper, we present a new radial space-filling approach to visualize planar st-graphs. Since these graphs have exactly one single source, one single sink, and no edge crossings, it is possible to adapt traditional space-filling layouts like treemap [2] or sunburst [7] approaches for a space-saving representation of st-graphs. We took inspiration from the work of Papamanthou and Tollis [5] who used wide rectangles to represent intermediate nodes of st-graphs, while using different colors to denote the source and sink nodes. A prototype has been implemented to demonstrate our ideas. The layout algorithm is briefly outlined in the next section followed by a discussion of the prototype implementation and preliminary results in Section 3.

*e-mail: ilir.jusufi@lnu.se

†e-mail: andreas.kerren@lnu.se

2 The Layout Strategy

The first three steps of the algorithm are well known from traditional layering algorithms introduced by the Graph Drawing community [8]. At first, *longest path layering* is used to place the nodes into corresponding layers. Second, so-called *dummy nodes* are introduced if an edge connects two nodes that are not in neighbor layers. The third step is responsible for *removing the edge crossings*. Figure 1(a) shows a small graph that has been layered.

These steps, although not introduced by us, form the basis for our radial space-filling approach. In the following, we present two phases for drawing the st-graph. In the first phase, we traverse the layers top-down: starting from the first layer (the source node) and compute all shapes and positions of each node. The final drawing is done in a second bottom-up traversal. The reason for this strategy is to avoid the drawing of complex shapes by drawing circles and sectors only. The second traversal could be avoided altogether by drawing the nodes in the computation phase by using a z-index (if the programming language permits it) to ensure that the nodes from top layers are placed in the foreground.

The first phase of the layout algorithm is explained in the following with the help of the sample graph shown in Figure 1(a). We start at the top layer $L1$, i.e., the source node. If there is only one node in the current layer, and there are no transitive edges passing through, the node is drawn as a circle with circumference depending on the layer number the node belongs to. In $L1$, the source node fulfills this condition and is drawn as shown in Figure 1(b). If there are more than one node in the corresponding layer, then they are

Poster Abstract, IEEE VisWeek 2012, Seattle, WA, USA, 2012.

drawn proportionally as sectors on top of the parent nodes. In case there is a dummy node, i.e., a transitive edge is present, a small percentage of the circular ring is omitted to show the connection between the previous layer and the next one. The rest of the circular space is divided proportionally as shown in Figure 1(c). The next step is more complicated: node 3 has the nodes 1, 2 and *s* as parents (keep in mind that node *d* is a dummy node and is only used to calculate a gap), i.e., the corresponding segment in the new layout has to "touch" all of them. However because node 2 has two children (3 and 4), the available space for the children should be divided equally on the area touching node 2. Therefore as depicted in Figure 1(d), we compute the position of node 4 on top of its parent taking half of the perimeter of the parent's sector. The rest of the space is reserved for node 3 as it touches the rest of the parents. Finally, the position and size of the sink node *t* is computed, and the graph can be finally drawn as shown in Figure 1(e). All positions are specified by calculating the start and stop coordinates of the circle sectors.

3 Implementation and Results

A prototype implementation has been developed using *Java* together with the *Processing* [6] graphic library. It uses *GraphML* [1] as input file format of the st-graphs. As our current tool is a proof-of-concept implementation, we have not implemented any standard crossing reduction algorithm or planarity check [3] yet. This task is manually done at the moment by using drag and drop interactions on the node-link representation of the loaded input st-graph shown in Figure 2. After the removal of all edge crossings, we are able to run our algorithm.

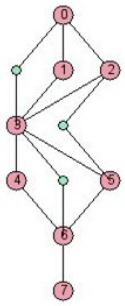

Figure 2: View for eliminating edge crossings (EEC-view). Users are able to drag and drop nodes within the corresponding layers. The small nodes without labels are automatically generated dummy nodes.

To demonstrate the final layout results in this poster paper, we built a st-graph by using the *yEd* [10] tool shown in Figure 3. Subsequently, the graph has been exported to GraphML format and loaded into our prototype implementation. The nodes will be placed in layers and dummy nodes will be created automatically. As mentioned before, we have to manually ensure that there are no edge crossings by interacting with the node-link representation. Finally, the algorithm creates a space-filling radial layout for planar st-graphs as shown in Figure 4. Brushing is used to show the mapping between the new space-filling layout and the EEC-view.

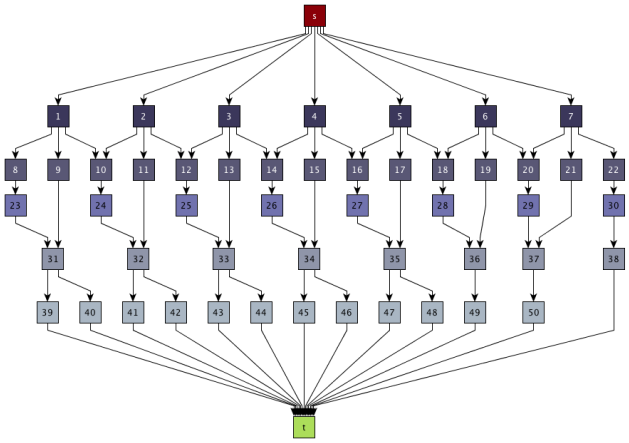

Figure 3: Manually specified planar st-graph consisting of 52 nodes and 76 edges (drawn by using *yEd*).

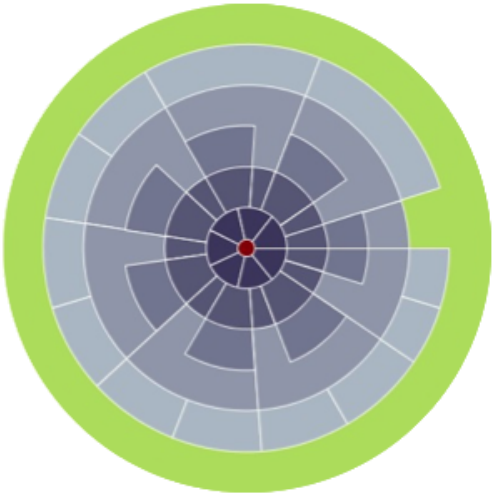

Figure 4: Screenshot of the layout produced by our algorithm using the input st-graph shown in Figure 3. Node labels are shown via interactive tooltips.

References

[1] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall. Graphml progress report (structural layer proposal). In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proceedings of the 9th International Symposium on Graph Drawing (GD '01)*, volume 2265 of *LNCS*, pages 501–512. Springer, 2002.

[2] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd conference on Visualization '91*, VIS '91, pages 284–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[3] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Proceedings of the International Symposium on Theory of Graphs '66*, pages 215–232. Gordon and Breach, New York, 1966.

[4] T. Nishizeki and M. S. Rahman. *Planar Graph Drawing*. Lecture Notes Series on Computing – Vol. 12. World Scientific, 2004.

[5] C. Papamanthou and I. G. Tollis. Applications of parameterized *st*-orientations in graph drawing algorithms. In *Graph Drawing*, pages 355–367, 2005.

[6] C. Reas, B. Fry, and J. Maeda. *Processing: A Programming Handbook for Visual Designers and Artists*. The MIT Press, 2007.

[7] J. Stasko. An evaluation of space-filling information visualizations for depicting hierarchical structures. *Int. J. Hum.-Comput. Stud.*, 53(5):663–694, Nov. 2000.

[8] 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.

[9] R. Tamassia and J. S. Vitter. Parallel transitive closure and point location in planar structures. *SIAM J. Comput*, 20:708–725, 1991.

[10] yWorks. *yEd Graph Editor*, last accessed: 2012-08-02. `http://www.yworks.com/en/products_yed_about.html`.