Edge Label Placement in Layered Graph Drawing

Christoph Daniel Schulze, Nis Wechselberg, and Reinhard von Hanxleden

Deptartment of Computer Science, Kiel University, Kiel, Germany
{cds,nbw,rvh}@informatik.uni-kiel.de

Abstract. Many visual languages based on node-link diagrams use edge labels. We describe different strategies of placing edge labels in the context of the layered approach to graph drawing and investigate ways of encoding edge direction in labels. We also report on the results of experiments conducted to investigate the effectiveness of the strategies.

1 Introduction

Visual programming languages based on *node-link diagrams*, such as Sequentially Constructive Charts (SCCharts) [13] (a synchronous state charts dialect, see Figure 1), have become mainstream in several industries. Many share a number of similarities: first, being based on a notion of either data flow (data is produced, processed, and consumed by *nodes* and transmitted between them through *links* or *edges*) or control flow (nodes represent *states* that can be active or not, with *transitions* transferring control between them); second, deriving some of their semantics through textual labels; and third, requiring users to spend a considerable amount of time on laying out their diagrams [7] for them to properly readable [9], giving rise to automatic layout algorithms [12].

A popular layout approach for flow-based diagrams is the layered approach introduced by Sugiyama et al. [11], which tends to emphasize data or control flow by making the majority of edges point in the same direction. The original description of the layered approach did not mention edge labels. Not taking them into account, however, will lead to layouts with too little space available for their placement, resulting in overlaps with other diagram elements—something that may well cause users to refrain from using automatic layout in the first place. This paper is about making labels first-class citizens during automatic layout.

Due to its prevalence in flow-based diagrams, we will limit our discussions to horizontal layout directions. While the question of how well-suited the methods are to vertical layouts is interesting, it is outside the scope of this paper due to space constraints.

Contributions. We show different ways of placing labels within the layered approach, including the selection of layers to place labels in and the side of their edge to place them on. We also investigate ways of encoding an edge's direction

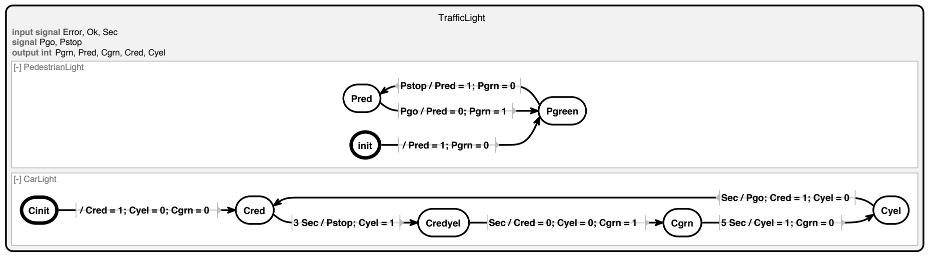

TrafficLight
input signal Error, Ok, Sec
signal Pgo, Pstop
output int Pgrn, Pred, Cgrn, Cred, Cyel
[-] PedestrianLight
Pred
Pstop / Pred = 1; Pgrn = 0
Pgo / Pred = 0; Pgrn = 1
Pgreen
init
/ Pred = 1; Pgrn = 0
[-] CarLight
Cinit
/ Cred = 1; Cyel = 0; Cgrn = 0
Cred
Sec / Pgo; Cred = 1; Cyel = 0
3 Sec / Pstop; Cyel = 1
Credyel
Sec / Cred = 0; Cyel = 0; Cgrn = 1
Cgrn
5 Sec / Cyel = 1; Cgrn = 0
Cyel

Figure 1: An SCChart laid out with the methods we propose in this paper. This drawing uses a horizontal layout with edges routed as splines.

through label placement or additional decorations, intended to be of particular help in use cases where only parts of a diagram can be displayed on screen. We summarize results of a controlled experiment as to the effectiveness of the proposed strategies.

Related Work. Label placement in general has a long history in cartography. In a classic paper [4], Imhof lays down six principles for good map labeling, which Kakoulis and Tollis [5] apply to edge labeling as the following three rules:

1. No overlaps between labels and other diagram elements.
2. It should be clear which diagram element a label belongs to. Imhof calls this "clear graphic association".
3. Among all acceptable positions, a label should be placed in the best possible.

Kakoulis and Tollis also provide a definition of the *edge label placement problem*, which is about placing edge labels in diagrams whose elements have already been placed. Existing algorithms, of which Kakoulis and Tollis provide an overview [12, Chapter 15], usually either run the risk of violating rules 1 or 2 or may resort to hiding or at least scaling down labels to avoid violations—both undesirable for visual programming languages.

In this paper we consider label placement a part of automatic layout, thereby ensuring that there will always be enough space available to satisfy rules 1 and 2. Klau and Mutzel [6] do this for the topology-shape-metrics approach to graph drawing, although their results do not always seem to satisfy rule 2. The *Graphviz dot*[1] algorithm, an implementation of the layered approach, handles edge labels by introducing dummy nodes [2], an approach we follow as well. However, they do not describe any strategies regarding where edge labels end up with regard to their edge. Castell et al. [1] place labels on edges, which is also one of our label placement strategies. However, they do not discuss graphical design considerations and do not evaluate whether doing so may have a negative impact on the ability of users to read the resulting drawing.

[1] `http://www.graphviz.org/`

There have been more radical proposals, most notably by Wong et al. [15] who replace an edge by its label. That approach would not work with long edges or orthogonal edge routing, but our on-edge label placement strategy to be introduced in Section 3 can be seen as a less extreme version of this technique.

Different methods have been proposed to indicate edge direction, such as using curvature, or color and thickness gradients from an edge's tail to its head [16,3]. These will cease to work in use cases where users only see a small part of a larger diagram and changes in such features are subtle. Animating edges or rendering them as sequences of arrows [3], may work well, but increase visual clutter and require the rendering of edges to be changed, which may be impossible if it carries semantical meaning (as, for example, it does in *LabVIEW* by *National Instruments*). Our methods do not require such design changes.

Outline. We describe label placement techniques in Sections 2 and 3, respectively, before introducing directional decorators in Section 4. We evaluate the techniques in Section 5 and conclude in Section 6.

An extended version of this paper that includes detailed descriptions of the experiments we report on is available as a technical report [10].

2 Layer Selection

The layered approach is split into five phases. *Cycle breaking* reverses edges in the input graph to make the graph acyclic, to be restored again once the algorithm has finished. *Layer assignment* partitions the set of nodes into a sequence of *layers* such that edges only point to layers further down the sequence. Edges that span multiple layers are broken by introducing *dummy nodes* such that edges always connect nodes in adjacent layers. *Crossing reduction* orders the nodes in each layer to reduce the number of edge crossings. *Node placement* computes vertical node coordinates and thereby determines the height of the diagram. *Edge routing* finally computes bend points for the edges according to the preferred edge routing style. For flow-based diagrams, this will usually be orthogonal edge routing or spline edge routing. The methods to be described in this paper work with both.

The aim of integrating edge label placement into the layout algorithm is to reserve enough space for the edge labels to be placed without overlaps and with clear graphic association. Similar to Graphviz dot, we break each edge that has labels by introducing a *label dummy node* to represent them. We compute the size of the dummy node such that all edge labels fit into it, stacked upon each other with a configurable amount of space between them, plus spacing to be left between the labels and their edge. Once edge routing has finished, label dummies are replaced by the labels they represent.

Label dummies need to be inserted before the layer assignment step to ensure that each dummy is assigned to a layer (which might end up existing only because of the label dummy). After layer assignment, we can move each label dummy to a layer of our choice, if necessary. That choice is obvious if the edge is so short that there is only one layer to choose from. If the edge is longer, however—such as the edge from `cyel` to `cred` in Figure 1—we need a strategy that defines what constitutes the best choice.

If the edge spans layers L_1 through L_n, two simple strategies are obvious. The *median strategy* places the label dummy in layer $L_{\lfloor n/2 \rfloor}$, while the *end layer strategy* places it either in layer L_1 (*source layer strategy*) or L_n (*target layer strategy*).

The most appropriate strategy depends on the visual language. In SCCharts, for example, edges represent transitions from a source to a target state that are eligible to be taken based on some condition, which is part of each transition's label. If edge labels are placed using the median strategy, a user might have to search a large area of an SCChart in order to understand a single transition. In this case, the source layer strategy may be more helpful.

An optimization goal might be to assign labels to layers such that the drawing's width is minimized. While taking layer widths into account seems easy enough, it is complicated by the fact that the widths may be changed by the assignment itself. Finding good algorithms to solve this problem is the subject of ongoing research and transcends the scope of this paper.

3 Label Side Selection

An edge label can be placed above, below, or even on the edge it belongs to, and we can think of different strategies to make a decision.

3.1 Same-Side Strategy

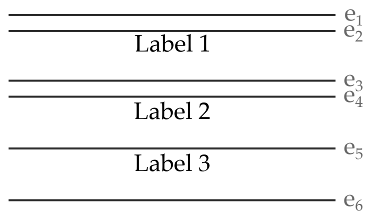
The *same-side strategy* places all labels either above or below their edge, as shown in Figure 2a. The simplest strategy to implement, it may also be the easiest for users to understand due to its consistency.

For clear graphic association to be achieved, it does require the spacing between a label and its edge to be noticeably smaller than the spacing between the label and other edges, in accordance with the Gestalt principle of *perceptual grouping* [14]. Donald Norman would call this "knowledge in the world" [8] in that users do not require further information to decipher the diagram.

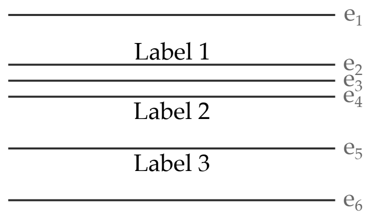
If spacings are chosen badly, the same-side strategy can still work if the user is aware of it. Norman, however, claims that such additional information required to understand the world—what he calls "knowledge in the head"—should be avoided when possible, making properly chosen spacings the preferred method.

3.2 Directional Strategy

The same-side strategy works well in terms of clear graphic association, but does not encode the direction an edge is heading towards. Since a label may be far removed from the end points of its edge (which, depending on the graph's size and the way it is displayed, may not even be on screen), any clue as to the edge



a: Same-side strategy.



b: Directional strategy.

Figure 2: Two different label side selection strategies. Both place labels closer to their edge than to other edges. Note that this is only an excerpt of a graph, which explains the absence of edge arrows or nodes.

direction may help a user navigate the diagram. The *directional strategy* aims to do just that by placing labels above edges running rightwards and below edges running leftwards.

Figure 2b shows an example of this strategy in action. Knowing about this placement method lets us deduce that e_2 is headed rightwards while e_4 and $e5$ are going off to the left. If spacings are chosen well, this additional piece of knowledge is not required for clear graphic association, but offers clues to advanced users of a visual language who know about the convention. If spacings are chosen badly, however, the directional strategy ceases to work due to the ambiguity it would produce.

Of course, this strategy requires knowledge in the head, which we will improve upon in Section 4.

3.3 On-Edge Strategy

We have thus far focused on placing labels next to their edge, which is the standard edge labeling strategy in the vast majority of graphical modeling tools, such as *Papyrus* (Eclipse Foundation) or *Simulink* (MathWorks). There is a case to be made, however, for placing them *on* their edge.

When placing labels next to their edge, one of our main concerns has to be clear graphic association. Wong et al. [15] respond to that challenge by replacing the edge with its label. We will not follow their proposal, for several reasons. First, for the approach to work without introducing distortion or very different font sizes, the length of an edge would have to be a function of the text it is labeled with—a prerequisite not compatible with the layered approach. And second, the orthogonal edge routing style (or any routing style that employs bend points, for that matter) would degrade label legibility even further.

On-edge label placement achieves optimal graphic association without completely replacing edges by their label. If the layout direction is horizontal, we may also reduce the diagram's height slightly because there is no edge-label spacing anymore, and since each label sits on its edge the space between the

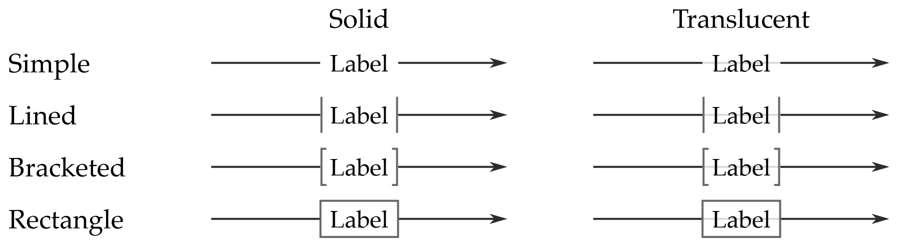


Figure 3: Four examples on-edge label designs.

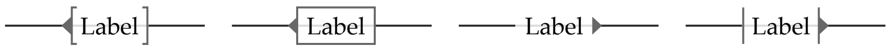


Figure 4: Labels can be decorated with arrows to point at where the edge is heading. While this example only shows on-edge labels, such decorations can of course also be added to labels placed next to their edge.

label and unrelated edges can be much smaller than it could otherwise. For on-edge label placement to work, the graphical representation of edge labels has to be designed accordingly. Labels must have either a solid background or at least cause the background to be sufficiently faded for the edge not to interfere with the text's legibility. This requirement is easy to meet, and many designs for on-edge labels are possible, which may even reflect different edge semantics. Figure 3 shows four simple examples of on-edge label representations. Castell et al. [1] use a simple solid design when drawing statecharts, but do not discuss their motivation for doing so. Interrupting the edge, however, may cause users to have a harder time following it through the diagram.

4 Directional Decorators

The directional label side selection strategy had the advantage of encoding information about the direction of an edge, but suffered from both potential graphic association problems as well as knowledge in the head for its proper interpretation. An alternative is to communicate through the label's design instead of its placement. Figure 4 shows examples of on-edge labels decorated with an additional arrow which points towards the edge's head. Such decorations work with any label side selection strategy, thus allowing the same-side strategy to communicate the same amount of information as the directional strategy while being slightly clearer in terms of graphic association.

An interesting problem concerns the implementation of directional decorators. Whether the arrow should point leftwards or rightwards is subject to the diagram's layout, which implies that the viewing framework needs to support changes to the visualization after automatic layout has run. How this can be done depends on the viewing framework and is outside the scope of this paper.

5 Evaluation

We conducted a controlled within-participants experiment with 48 participants in order to answer two research questions (a detailed account of the experiment is available in the expanded technical report [10]).

First, are users better at inferring edge directions with directional label placement or with on-edge label placement with directional decorators? We showed users random images with lines labeled using one of the two strategies, intended to simulate seeing excerpts of larger diagrams. We found that they had a significantly faster response time and significantly lower error rate with on-edge labels.

Second, does on-edge label placement have a negative impact on the ability of users to follow edges through a diagram? We showed users graphs that had a start node highlighted and asked how many nodes were reachable from that node in two steps. Among three conditions (same-side, directional, and on-edge) we could not find significant differences in response time or error rate.

In a concluding interview we asked participants to rank four label placement strategies (same-side, directional, on-edge without and with directional decorators). The latter was ranked significantly higher than the other three strategies, among which we did not find significant differences. The directional strategy was often described as being confusing. Some participants mentioned that the value of on-edge label placement with directional decorators increases with a diagram's size, noting that the additional arrows can add visual clutter to small diagrams.

6 Conclusion

We presented different placement strategies for placing edge labels in flow-based diagrams. On-edge label placement yields clearest graphical association, and usually slightly smaller diagrams. With directional decorators added it was largely preferred by users. Some did complain about the fact that on-edge labels interrupt their edges, but we did not find significant performance differences in a task that required participants to follow edges through a diagram.

Future Work Some visual languages tend to produce rather long edge labels that make the assignment of labels to layers a crucial influence on the width of drawings. Finding a heuristic that yields assignments that produce smaller drawings seems necessary.

Clear graphic association of on-edge labels may be impaired if labels span multiple lines of text. This issue should be investigated further.

Acknowledgements. The authors would like to thank Nadine Yarar and Andreas Mühling, and Emmanuel Manalo for helpful suggestions while designing the experiment, and Helen Purchase for her excellent book on the subject.

This work has been supported in part by the German Science Foundation, as part of the Compact Graph Drawing with Port Constraints (ComDraPor,DFG HA 4407/8-1) project.

References

1. R. Castelló, R. Mili, and I. G. Tollis. An algorithmic framework for visualizing Statecharts. In *GD 2000: Proceedings of the 8th International Symposium on Graph Drawing*, volume 1984 of *LNCS*, pages 43–44. Springer-Verlag, 2001.
2. E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *Software Engineering*, 19(3):214–230, 1993.
3. D. Holten, P. Isenberg, J. J. van Wijk, and J.-D. Fekete. An extended evaluation of the readability of tapered, animated, and textured directed-edge representations in node-link graphs. In *2011 IEEE Pacific Visualization Symposium*, pages 195–202, mar 2011.
4. E. Imhof. Positioning names on maps. *The American Cartographer*, 2(2):128–144, 1975.
5. K. G. Kakoulis and I. G. Tollis. An algorithm for labelling edges of hierachical drawings. In G. Di Battista, editor, *Graph Drawing (Proceedings GD '97)*, LNCS, pages 169–180. Springer-Verlag, 1997.
6. G. W. Klau and P. Mutzel. Combining graph labeling and compaction. In *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, volume 1731 of *LNCS*, pages 27–37. Springer, 1999.
7. L. K. Klauske and C. Dziobek. Improving modeling usability: Automated layout generation for Simulink. In *Proceedings of the MathWorks Automotive Conference (MAC'10)*, 2010.
8. D. A. Norman. *The Design of Everyday Things*. Basic Books, 1988.
9. M. Petre. Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM*, 38(6):33–44, June 1995.
10. C. D. Schulze, N. Wechselberg, and R. von Hanxleden. Edge label placement in layered graph drawing. Technical Report 1802, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Feb. 2018. ISSN 2192-6247.
11. 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, Feb. 1981.
12. R. Tamassia, editor. *Handbook of Graph Drawing and Visualization*. CRC Press, 2013.
13. R. von Hanxleden, M. Mendler, J. Aguado, B. Duderstadt, I. Fuhrmann, C. Motika, S. Mercer, O. O'Brien, and P. Roop. Sequentially Constructive Concurrency—A conservative extension of the synchronous model of computation. *ACM Transactions on Embedded Computing Systems, Special Issue on Applications of Concurrency to System Design*, 13(4s):144:1–144:26, July 2014.
14. J. Wagemans, J. H. Elder, M. Kubovy, S. E. Palmer, M. A. Peterson, M. Singh, and R. von der Heydt. A century of Gestalt psychology in visual perception: I. Perceptual grouping and figureground organization. *Psychological Bulletin*, 138(6):1172–1217, Nov. 2012.
15. P. C. Wong, P. Mackey, K. Perrine, J. Eagan, H. Foote, and J. Thomas. Dynamic visualization of graphs with extended labels. In *INFOVIS '05: Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, page 10, Washington, DC, USA, 2005. IEEE Computer Society.
16. K. Xu, C. Rooney, P. Passmore, D.-H. Ham, and P. H. Nguyen. A user study on curved edges in graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2449–2456, Dec 2012.