Using One-Dimensional Compaction for Smaller Graph Drawings

Ulf Rüegg, Christoph Daniel Schulze,
Daniel Grevismühl, and Reinhard von Hanxleden

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

Abstract. We use the technique of one-dimensional compaction as part of two new methods tackling problems in the context of automatic diagram layout: First, a post-processing of the layer-based layout algorithm, also known as Sugiyama layout, and second a placement algorithm for connected components with external extensions.
We apply our methods to dataflow diagrams from practical applications and find that the first method significantly reduces the width of left-to-right drawn diagrams. The second method allows to properly arrange disconnected graphs that have hierarchy-crossing edges.

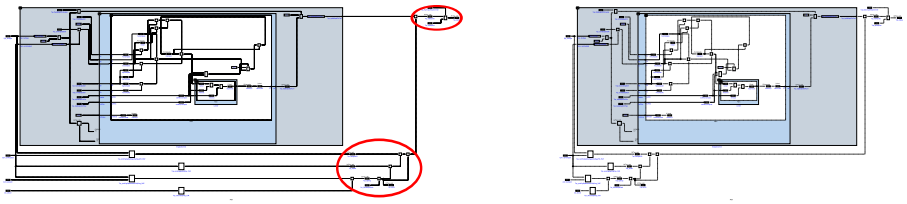
1 Introduction

Automatically drawing graph-based visual models has gained more and more acceptance over the past years, with industrial tools starting to incorporate automatic layout facilities, be it semi-automatic or fully-automatic, to support model-driven engineering or interactive browsing of models [2]. Example tools are *LabVIEW* (National Instruments), *EHANDBOOK* (ETAS), *Simulink* (The MathWorks, Inc.), and *Ptolemy* (UC Berkeley).

For applications where *hierarchical dataflow diagrams* are used, the layout techniques have continuously been improved to handle most of the peculiarities of this type of diagram [13]. Still, further improvements are required regarding the compactness of the resulting drawings [6]. In this paper we show how the simple technique of *one-dimensional compaction* can be used to significantly improve the compactness of dataflow diagrams drawn with state-of-the-art methods (see Fig. 1 for a result). While we motivate our contributions from the perspective of dataflow diagrams, they are not restricted to this type of diagram. The presented methods are implemented as part of the open-source Eclipse Layout Kernel (ELK)[1].

One-dimensional compaction is a well-known technique to minimize the area occupied by a set of objects in the plane. As opposed to the NP-hard two-dimensional compaction problem, it can be solved efficiently in time $O(n \log n)$, n being the number of objects [8]. Given a set of rectangles $\mathcal{R}$, where every rectangle is of the form $r = (r_x, r_y, r_w, r_h)$, one seeks for a set of rectangles

[1] http://www.eclipse.org/elk



(a) **(b)**

Fig. 1. Illustration of our first contribution. (a) An automatically drawn dataflow diagram with the layer-based layout methods by Schulze et al. [13]. Circled nodes are pushed to the right due to the method's nature. (b) The same diagram after our post-processing. The diagram's width is reduced by about 16% and the average edge length is reduced by over 50%.

$\mathcal{R}'$ by changing x-coordinates only such that no pair of rectangles overlaps, the relative positioning is preserved, and the overall width w is minimized, with $w = |\ \max_{r' \in \mathcal{R}'}\ (r'_x + r'_w)\ -\ \min_{r' \in \mathcal{R}'}\ r'_x\ |$.

In the remainder of this paper we outline our contributions. Further implementation details, how to address the peculiarities of dataflow diagrams, and possible strategies for improvement, can be found in an accompanying technical report [12].

2 Layer-Based Drawings

In 1981, Sugiyama et al. described the structure of a successful methodology to draw directed graphs in the plane [14]. It is known under various names such as Sugiyama-style layout, hierarchical layout, and layer-based layout [7]. Essentially, it consists of five consecutive phases: (1) *cycle breaking* makes cyclic graphs acyclic by reversing edges, (2) *layering* assigns nodes to indexed layers such that edges always connect layers of lower index to higher index, (3) *crossing minimization* aims at reducing the number of edge crossings, (4) *node coordinate assignment* determines explicit y-coordinates for nodes, and (5) *edge routing* determines paths for edges and assigns x-coordinates to nodes. Most literature in this context assumes that nodes are of the same size. However, this is not the case with most practical applications, and it has been observed that the compactness of diagrams suffers in the presence of significant size differences [3,6]. Existing methods to tackle this issue either result in unpleasant drawings or increase the complexity of subsequent steps of the approach [9,10,3]. A common idea is to assign large nodes to multiple layers, for instance, by splitting them into multiple small chunks. The crossing minimization phase then has to keep edges from crossing nodes, and the node coordinate assignment has to assert that all chunks receive the same y-coordinate. Nonetheless, the problem becomes more and more imminent with diagram exploring approaches where nodes sizes may differ by factors of 10 or even 100 [2,6].

Recently, Schulze et al. presented several extensions to the layer-based approach to handle dataflow diagrams with *ports* (explicit attachment point of edges on a node's perimeter) and *orthogonally* routed edges [13], see Fig. 1. Working with these kinds of diagrams, we observed that scenarios where wide nodes (shaded background) prevent more compact placements are quite common. The problem illustrated in Fig. 1a is that the layer-based approach assigns nodes rigidly to layers and no pair of connected nodes may be placed in the same layer, thus pushing the set of small nodes in the lower right to the right. Here, one-dimensional compaction allows to reduce the diagram's overall width by breaking the rigid layering and pushing everything as far as possible to the left. During the process, vertical segments of orthogonally routed edges may be regarded as rectangles with either zero or very small width. Since the compaction procedure can be applied to the final drawing, after the traditional layer-based approach has finished completely, no additional complexity is added to any of the layer-based phases.

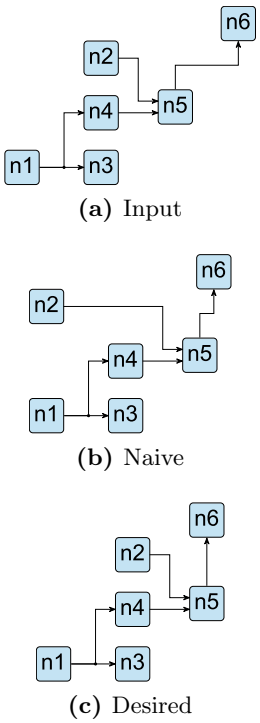


Fig. 2. Example of applying one-dimensional compaction to an input graph (a) with different objectives of minimal width and minimal edge length in (b) and (c).

Diagrams as the one seen in Fig. 1 can be formalized as *directed hypergraphs*, which are pairs $HG = (V, H)$. V is a set of nodes and $H \subseteq (P(V) \times P(V))$ a set of hyperedges that are connected to nodes via one of the nodes' *ports*. Schulze et al. represent each hyperedge $h = (S, T) \in H$ by a set of edges, i. e. for every pair $s \in S$ and $t \in T$ a directed edge $e = (s, t)$ is introduced. This allows to use known layer-based methods without the requirement to specifically address hyperedges. Both nodes and edges can carry labels that contribute to their bounding boxes. Additionally, a drawing must adhere to certain spacings between nodes and edges. After applying standard layer-based techniques, one-dimensional compaction can be applied to HG by transforming it into a set of rectangles $\mathcal{R}$: 1) the bounding box of every node $v \in V$ is added as a rectangle to $\mathcal{R}$. 2) For every vertical segment of an edge $e \in H$, we add a rectangle with corresponding height and unit width to $\mathcal{R}$. To guarantee enough room for edge labels they can be added to the set of rectangles as well. Still, to get satisfying results in practice, several subtleties have to be addressed. First, prescribed spacing values between diagram elements have to be maintained. This can be done by either enlarging the rectangles in $\mathcal{R}$ or by adding minimum separation constraints to the edges of the constraint graph. Second, with the extensions by Schulze et al. [13], edges are allowed to connect to the northern and southern border of a node. Consider

Table 1. Results of applying one-dimensional compaction to layer-based drawings of dataflow diagrams. LR stands for left compaction followed by right compaction, and EL stands for compaction aiming for short edges. $\bar{n}$ and $\bar{e}$ denote the average number of nodes and edges. $\bar{w}$ denotes the average width after compaction in percent of the original width, $\bar{el}$ the average edge length. Standard deviations are given in brackets.

	$\bar{n}$	$\bar{e}$		$\bar{w}$(%)	$\bar{el}$(%)
EHANDBOOK	25.4 [15.0]	30.8 [18.3]	LR	83.7 [11.9]	78.2 [15.4]
			EL	85.3 [11.2]	76.6 [16.2]
Ptolemy	15.7 [7.4]	19.6 [11.5]	LR	93.6 [8.2]	88.3 [13.8]
			EL	94.3 [7.4]	87.1 [13.3]

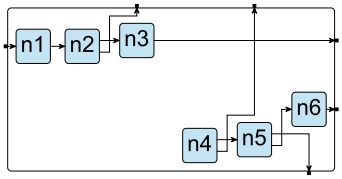
the edge $e = (\mathsf{n5}, \mathsf{n6})$ in Fig. 2. Such vertical segments are *grouped* with the corresponding node during compaction, which prevents them from detaching. Third, edge lengths are not considered: compare the position of n2 in Fig. 2b and c). While this problem has been discussed in the literature [8], we suggest two simple solutions specifically tailored for graph layout: a) Having compacted to the left, fix the positions of nodes that have no outgoing edge in the original graph, and execute another compaction pass to the right. This preserves minimum width with some edge length reduction. b) Add the edges of HG to the constraint graph and use the adapted network simplex algorithm presented by Gansner et al. [4] to find a placement with minimum edge length. More details on all three points can be found in the accompanying technical report [12].

Our main goal is to improve on diagrams that occur in practice. Our evaluation set consists of 69 diagrams from the commercial interactive model browsing solution EHANDBOOK[2] and a subset of 529 diagrams shipping with the academic Ptolemy project [11]. Both diagram types are hierarchical: nodes can contain further nodes, i. e. sub-diagrams. We extracted such sub-diagrams and evaluate them separately, which is feasible since the layout algorithm considers every sub-diagram separately anyway. The results of applying our method can be seen in Tab. 1. We measured values for both compaction strategies previously mentioned: subsequent left-right compaction with node locking (LR) and minimizing edge length (EL). The average width of the EHANDBOOK and Ptolemy drawings decreased by about 16% and 6%, the edge lengths decreased by 22% and 12%. No significant difference can be observed between the two compaction strategies. Still, since edges that can obviously be shortened are immediately noticed by users (cf. Fig. 2), we suggest to use compaction with edge length minimization. Executed on an Intel i7 2GHz CPU and 8GB memory laptop, both methods finish in well under 10ms for up to 100 nodes. For up to 1000 nodes EL's execution time increases significantly, which is expected since the network simplex algorithm is used. Still, it finishes in under 0.6s. Therefore all setups are fast enough for applications that involve user interaction.

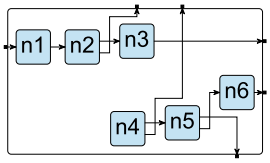
[2] `http://www.etas.com/de/products/ehandbook.php`

3 Connected Components With External Extensions

When a diagram consists of multiple sub-graphs that are not connected among each other, see Fig. 3 for a simple example, the problem arises to place the sub-graphs in the plane such that little space is used. Each sub-graph can be approximated by its bounding box and the problem can be formulated as a rectangle packing problem. However, such problems are often NP-complete [8] and rectangles may be poor approximations. Freivalds et al. and Goehlsdorf et al. discuss relevant related work and present heuristics for the problem based on a *polyomino* representation, which approximates every sub-graph using squares on a grid [1,5]. The approaches work well for flat diagrams. With dataflow diagrams, a node can contain a sub-graph and nodes of the sub-graph can be connected to nodes on other hierarchy levels via so-called *external ports* on the hierarchical node's perimeter. When placing the sub-graphs in the plane these edges have to be considered. They are not allowed to cross other sub-graphs. This cannot be prevented using the previously mentioned methods. Furthermore, sub-graphs should be placed such that the overall length of external edges is as small as possible.



(a) Before compaction



(b) After compaction

Fig. 3. Placing a diagram's sub-graphs must assert that no external edge crosses a subgraph.

To better approximate a sub-graph, we construct its *rectilinear convex hull* and split it into a set of rectangles. Both can be done in $O(n \log n)$ time using a scanline method, where n is the number of points used to represent the area covered by a sub-graph in the first case, and the number of corners of the rectilinear convex hull in the second case.

Let $\mathcal{C}$ be a set of *components*, see also Fig. 4. Each component $c_i \in \mathcal{C}$ is a tuple $c_i = (\mathcal{R}_i, \mathcal{E}_i)$, where $\mathcal{R}_i$ is a non-empty set of *rectangles* and $\mathcal{E}_i$ is a (possibly empty) set of *external extensions*. Rectangles are 4-tuples (see Sec. 1). The k-th rectangle of c_i is r_i^k. We assume that all rectangles of the same component somewhere touch alongside their border. An external extension $e_i^l = (d_i^l, \delta_i^l, \epsilon_i^l)$ of a component c_i is a triple of a direction $d_i^l \in \{n, e, s, w\}$, an offset δ_i^l relative to r_i^0, and a width ϵ_i^l. The offset and the width describe an extension clockwise, i. e. for a south extension, the offset is its right-most point and the width points to the left. Intuitively it represents a line or a strip attached to the border of a rectangle which extends infinitely into the specified direction. See Fig. 4 for an illustration. We say an extension (d, δ, ϵ) is *horizontal* if $d \in \{w, e\}$ and *vertical* if $d \in \{n, s\}$. A set of components $\mathcal{C}$ is considered *proper* if no pair of components overlaps and no external extension overlaps a component.

Using compaction to minimize the area of a set of components requires a proper set of components to start with. We use the *cell packing* algorithm [12] that turns a (possibly improper) set of components into a proper one by calculating sensible x and y-coordinates for all rectangles.

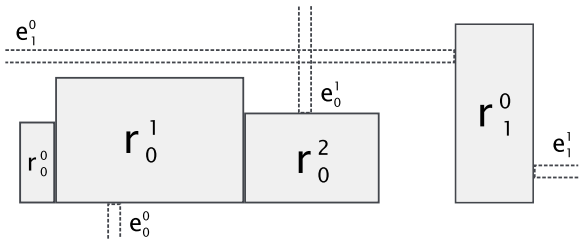


Fig. 4. The diagram shows two components c_0 and c_1. c_0 consists of three rectangles and two external extensions and c_1 consists of a single rectangle and two extensions. The external extensions e_1^0 and e_0^1 are allowed to overlap since one is vertical and the other one is horizontal. They are not, however, allowed to overlap with any of the rectangles.

For compacting layer-based drawings as described in Sec. 2, it is sufficient to compact along the x-dimension only. This time, however, it is necessary to compact in both dimensions, which is possible by continuously applying one-dimensional compaction in alternating dimensions and directions until no further, or little, progress is made. For a given set of components $\mathcal{C}$ we construct a grouped constraint graph (cf. [12]). Each component is represented by a group and the component's rectangles are added to it. The external extensions are converted into finite rectangles: each external extension is cut at the point where it intersects with the bounding box surrounding all components of $\mathcal{C}$. After each compaction pass these lengths have to be adjusted to prevent components from permuting.

Obviously, horizontal and vertical extensions that represent rectangles cannot be present at the same time during one-dimensional compaction since the set of rectangles may not be valid. Remember that external extensions are allowed to overlap with each other but the representing rectangles are not allowed to overlap. Still, it is important that the horizontal extensions are considered during vertical compaction, to prevent nodes from overlapping with external extensions; the same is true for vertical extensions during horizontal compaction. The independent application of horizontal and vertical compaction allows to use two different sets of rectangles depending on the compaction direction: $\mathcal{H} = \mathcal{R} \cup \{(d, \delta, \epsilon) \in \mathcal{E} : d \in \{n, s\}\}$ for horizontal compaction and $\mathcal{V} = \mathcal{R} \cup \{(d, \delta, \epsilon) \in \mathcal{E} : d \in \{e, w\}\}$ for vertical compaction.

4 Final Remarks

In this paper we show how one-dimensional compaction can be applied to two problems from the field of automatic diagram layout, more specifically, layer-based drawings and placement of disconnected graphs. We tested our methods with dataflow diagrams from practice and found that the width of layer-based drawings can significantly be reduced and that they allow disconnected graphs with hierarchy-crossing edges that are part of hierarchical graphs to be placed.

Acknowledgements. This work was supported by the German Research Foundation under the project *Compact Graph Drawing with Port Constraints* (ComDraPor, DFG HA 4407/8-1).

References

1. K. Freivalds, U. Dogrusoz, and P. Kikusts. Disconnected graph layout and the polyomino packing approach. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 378–391. Springer Berlin Heidelberg, 2002.
2. P. Frey, R. von Hanxleden, C. Krüger, U. Rüegg, C. Schneider, and M. Spönemann. Efficient exploration of complex data flow models. In *Proceedings of Modellierung 2014*, Vienna, Austria, Mar. 2014.
3. C. Friedrich and F. Schreiber. Flexible layering in hierarchical drawings with nodes of arbitrary size. In *Proceedings of the 27th Australasian Conference on Computer Science (ACSC'04)*, pages 369–376. Australian Computer Society, Inc., 2004.
4. 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.
5. D. Goehlsdorf, M. Kaufmann, and M. Siebenhaller. Placing connected components of disconnected graphs. In *6th International Asia-Pacific Symposium on Visualization, 2007*, pages 101–108, Feb 2007.
6. C. Gutwenger, R. von Hanxleden, P. Mutzel, U. Rüegg, and M. Spönemann. Examining the compactness of automatic layout algorithms for practical diagrams. In *Proceedings of the Workshop on Graph Visualization in Practice (GraphViP'14)*, Melbourne, Australia, July 2014.
7. P. Healy and N. S. Nikolov. Hierarchical drawing algorithms. In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*, pages 409–453. CRC Press, 2013.
8. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, Inc., New York, NY, USA, 1990.
9. K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, June 1995.
10. S. C. North and G. Woodhull. Online hierarchical graph drawing. In *Revised Papers of the 9th International Symposium on Graph Drawing*, volume 2265 of *LNCS*, pages 232–246. Springer, 2002.
11. C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.
12. U. Rüegg, C. D. Schulze, D. Grevismühl, and R. von Hanxleden. Using one-dimensional compaction for smaller graph drawings. Technical Report 1601, Kiel University, Department of Computer Science, Apr. 2016. ISSN 2192-6247.
13. C. D. Schulze, M. Spönemann, and R. von Hanxleden. Drawing layered graphs with port constraints. *Journal of Visual Languages and Computing, Special Issue on Diagram Aesthetics and Layout*, 25(2):89–106, 2014.
14. 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.