Examining the Compactness of Automatically Generated Layouts for Practical Diagrams

Carsten Gutwenger[1], Ulf Rüegg[2], Miro Spönemann[2],
Reinhard von Hanxleden[2], and Petra Mutzel[1]

[1] Technische Universität Dortmund
{carsten.gutwenger,petra.mutzel}@tu-dortmund.de
[2] Christian-Albrechts-Universität zu Kiel
{rvh,uru,msp}@informatik.uni-kiel.de

Abstract. Graph drawing algorithms have important practical applications, e. g. *layer-based* algorithms for *data flow diagram* layout in embedded software design and *planarization-based* algorithms to layout UML diagrams in software engineering.
Most current drawing methods focus on the optimization of aesthetic criteria such as the number of edge crossings and bends. The aspects of compactness and aspect ratio are often treated with lower priority, but in practice these are important as well. We present computational experiments showing that compactness can become a problem, especially for large and nested diagrams. Furthermore, we discuss possible new research directions.

1 Introduction

In the context of graph-based diagrams, automatic layout algorithms are a common tool to free developers from the time-consuming task of manually arranging nodes and links on a canvas. Different kinds of diagrams may have domain-specific requirements on the arrangement of the elements. For data flow diagrams, *layer-based* methods have proven themselves to work well [18], whereas for UML class diagrams *planarization-based* methods yield more suitable results [6].

With diagrams used in practice, however, we found that the drawings created with these methods often lack *compactness*. Consider Fig. 1, a data flow diagram from an industrial application [8] that models the control unit of a car engine. The original size of the diagram is $25\,179 \times 11\,035$ pixels. Using modern monitors with a resolution of $1\,920 \times 1\,080$ pixels, this means that 134 monitors would be required to display the diagram with its intended size (neglecting the correct aspect ratio). In other words, less than 1 % of the diagram can be shown on a standard monitor with the original 1:1 scale. Even though for most tasks that are conducted by engineers only parts of the diagram are relevant and sophisticated filtering methods can help to reduce the presented information, this example demonstrates that readability can be severely compromised by poor usage of screen area.

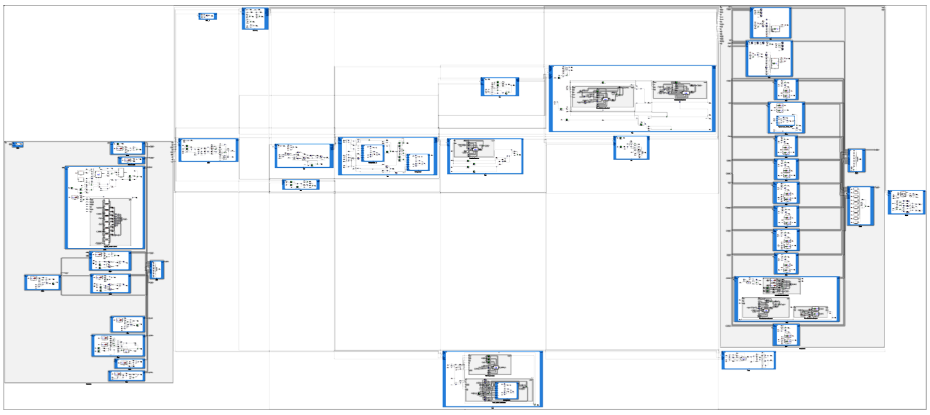


Fig. 1. A practical data flow diagram modeling the control unit of a car engine. The diagram is scaled to less than 2.5 % of its original size. More than 90 % of the diagram's area is unused.

Contributions. In this paper, we review existing research on the compactness of graph drawings, present results of experimental evaluations with practical diagrams, and discuss new ideas for future research. Summarizing the results of our experiments, we made the following observations:

(O1) *Aspect Ratio.* Layer-based algorithms yield bad aspect ratios for certain diagrams, especially for diagrams with long directed paths.

(O2) *Whitespace of Compound Diagrams.* For diagrams containing compound nodes, i. e. nodes that contain nested subgraphs, the amount of whitespace increases with the number of compound nodes when using layer-based algorithms.

(O3) *Whitespace and Planarization Methods.* While planarization-based methods produce drawings with few edge crossings, for certain diagrams the amount of whitespace can increase drastically.

Outline. We define different aspects of compactness in Sect. 2 and discuss existing work on improving the compactness of diagrams in Sect. 3. Sect. 4 presents two experiments that quantify the fraction of diagrams that lack certain criteria of compactness. We conclude with possible future research directions in Sect. 5.

2 Definition of Compactness

We define the width w of a diagram as the difference between the largest and the smallest x-coordinate of any element in the drawing; the height h is defined analogously using y-coordinates. Traditionally, compactness has been understood as the goal to minimize the area $w \cdot h$, or to minimize w or h independently (cf. the optimization goals for orthogonal compaction discussed in [15, Chapter 4]).

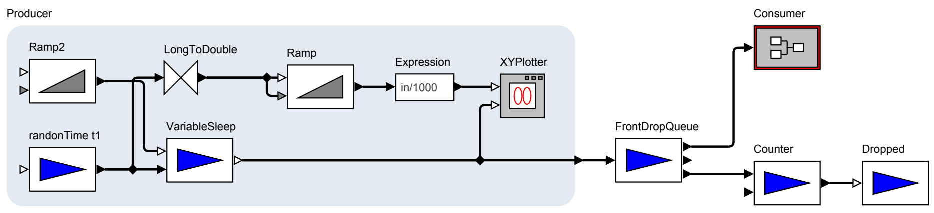


Fig. 2. A Ptolemy data flow diagram drawn with the methods of Schulze et al. [18]. Producer and Consumer are *compound actors*, where only the content of Producer is visible. All other nodes are *atomic actors*.

A difficulty with this definition is that it does not allow an absolute rating of a drawing, since it is not known how much area is required for a good drawing of a particular graph. Here we consider two other aspects of compactness that are easier to assess, namely the aspect ratio and the whitespace ratio.

Aspect ratio. The aspect ratio of a drawing is $r = w/h$. An ideal aspect ratio matches the used display medium. For monitors this depends on the resolution, and is typically between 1.33 and 1.78. Often diagrams are printed to a sheet of paper, which has an aspect ratio of 1.41 (ISO 216 format). As drawings with aspect ratios outside, yet close to, these values are fine, we assume in the following that all aspect ratios r in the range $1 \leq r \leq 2$ are acceptable.

Whitespace. The whitespace of a drawing is the area that remains unused. We measure the whitespace by literally painting the rectangular bounding box of all nodes onto a black-and-white canvas. Every node's bounding box includes possible labels and ports that are located outside the actual node. Additionally, layout algorithms can be configured with a parameter s controlling the minimal spacing between any two nodes. We extend the bounding box in all directions by the value s. The whitespace value is the ratio of the number of pixels that are left white to the total number of pixels in the canvas.

Edges are neglected as we feel that including them in the black-and-white drawing would bias the result: long edges would lead to larger areas that are drawn in black and thus reduce the whitespace, contradicting the aesthetic criterion of minimizing edge length. For a similar reason, we omit compound nodes in the whitespace evaluation, since their size is adapted to the bounding box of their nested subgraph.

We are not aware of any studies evaluating how much whitespace is necessary or helpful to conceive a diagram optimally. Certainly some whitespace is necessary in order to emphasize certain structures such as grouping of nodes. In the future a quantification would be desirable. Furthermore, our results encourage a metric that takes more factors into account, such as edges. Following our definition, graphs with higher edge density typically yield higher amounts of whitespace, hence the comparison of results for different graphs is problematic.

3 Previous Work on Compactness

Layer-based algorithms. Layer-based methods are the standard approach for drawing directed graphs. It consists of the following steps:

1. *Cycle removal:* Eliminate cycles by reversing a small subset of edges.
2. *Layer assignment:* Assign all nodes to *layers* such that edges point from layers of lower index to layers of higher index.
3. *Crossing reduction:* Find an ordering of the nodes in each layer with the aim of reducing the overall number of crossings.
4. *Coordinate assignment:* Determine node coordinates and replace dummy nodes by bend points.

A very common optimization goal for layer assignment is to minimize the sum of layer differences of source and target nodes for all edges. While this can be solved efficiently [10], it does not solve the compactness problems described above. Compactness has been addressed with regard to the number of layers and the maximal number of nodes per layer [11,17,1], which is often called the *width* of the layering. Further approaches target a given aspect-ratio [16] or distributed large nodes over several layers [9].

Planarization-based algorithms. The *topology-shape-metrics* approach consists of the following three phases:

1. *Planarization:* Remove edges until the graph is planar, then reinsert each removed edge with a minimal number of crossings.
2. *Orthogonalization:* Compute a bend-minimal edge routing that consists of horizontal and vertical line segments.
3. *Compaction:* Find a valid assignment of node and bend point coordinates such that the total length of line segments is minimized.

The simplest approach for one-dimensional compaction is based on longest paths and guarantees a minimal width or height of the drawing, but does not guarantee that the sum of the edge lengths is minimal. The latter can be achieved using flow-based compaction. However, both methods require that the orthogonal representation has only rectangular faces; this can be achieved by adding additional edges (*dissection*) but limits the amount of freedom for compaction. An alternative approach is based on turn-regularity [4], which does not require rectangular faces. An experimental comparison of these methods is presented in [13]. Due to their nature of only considering one dimension, none of these methods considers the aspect ratio of the resulting drawing. For two-dimensional compaction, the drawing is iteratively compacted in x- and then in y-direction, until no more improvement can be achieved. However, the resulting solutions can be far away from an optimal solution for the two-dimensional compaction problem. On the other hand, Klau and Mutzel [14] describe an ILP-based optimal algorithm for two-dimensional compaction, which can minimize the sum of the edge lengths or the sum of width and height of the drawing. Moreover, Eiglsperger and Kaufmann [7] present a linear-time compaction heuristic.

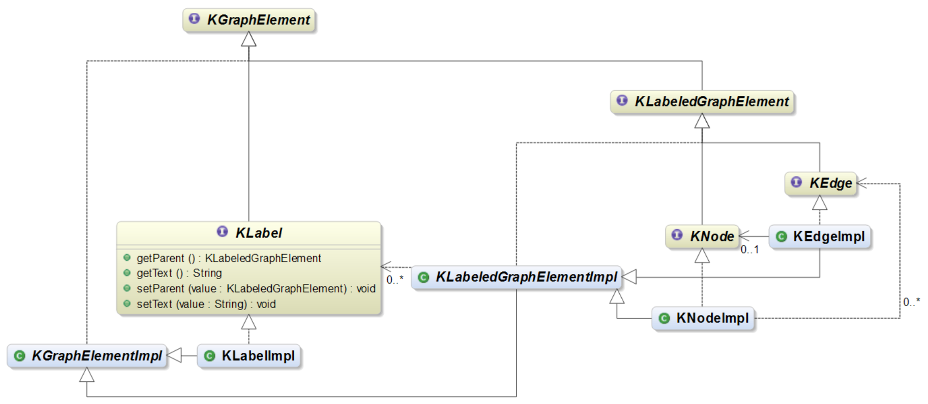


Fig. 3. A UML class diagram drawn with the approach of Eiglsperger et al. [6]. While the methods of the KLabel interface are visible, they are hidden for all other interfaces and classes.

4 Experiments

We present experiments based on two classes of diagrams used in practice and a set of graphs that are often used for evaluating graph drawing methods. We analyze the aspect ratio and the amount of whitespace of the created drawings, showing that our observations discussed in Sect. 1 hold for a considerable portion of diagrams.

Ptolemy data flow diagrams. Ptolemy is an open source project that ships with a number of example data flow models for testing and demonstration.[3] The diagrams can be nested with *compound actors*, which are composed of further actors to describe subsystems; see Fig. 2 for an example where the Producer compound actor is *expanded*, i. e. its internals are made visible, and the Consumer compound actor is *collapsed*. All other actors are *atomic* actors, i. e. do not contain children.

For our evaluations we use three variations of these models. First, we *flatten* all compound actors, i. e. all contained atomic actors are recursively moved to the highest hierarchy level and the compound actors are eliminated. Second, we display all atomic actors within their parent compound actors. Third, we start with collapsed compound actors and successively expand them, obtaining different views for each diagram. To illustrate this, Fig. 2 shows one expanded compound actor, Producer. Expanding the Consumer actor would yield another view where the content of both compound actors is visible. Overall we evaluated 330 flattened diagrams, 199 diagrams with full hierarchy containing at least 2 compound nodes, and 11 successively expanded diagrams.

[3] `http://ptolemy.eecs.berkeley.edu`

Table 1. Percentages of Ptolemy diagrams drawn with certain aspect ratios. For each listed variation, an extra row contains the results for graphs with more than 20 nodes.

Aspect Ratio Intervals		$(0, 1)$	$[1, 2]$	$(2, \infty)$
Ptolemy	Flattened Graphs	24 %	36 %	40 %
	$>$ 20 Nodes	41 %	37 %	22 %
	Compound Graphs	26 %	44 %	30 %
	$>$ 20 Nodes	32 %	40 %	28 %
North Graphs	Klay Layered	52 %	30 %	18 %
	$>$ 20 Nodes	63 %	23 %	14 %
	Graphviz Dot	60 %	23 %	17 %
	$>$ 20 Nodes	70 %	16 %	14 %

Class diagrams. KIELER is an open source project written in Java with over 1.4 million lines of code.[4] We created eight class diagrams of several subprojects, e. g. KIML, KLighD, and SCL. The diagrams contain 20 to 55 nodes and 27 to 75 edges. We use two variations of the class diagrams, one where the details of classes and interfaces are visible and one where they are hidden, see Fig. 3.

North graphs (a. k. a. AT&T graphs). The library is widely used in the graph drawing community and serves as an application-independent benchmark [5]. We assigned a size of $(30, 30)$ to each node for the whitespace evaluation. We evaluated 1276 graphs with 10 to 100 nodes and 9 to 241 edges.

We processed the North graphs with the *Dot* algorithm of the Graphviz library [10] and the *KLay Layered* algorithm of KIELER [18]. The data flow diagrams have been drawn using KLay Layered only, which is specialized for handling ports. The *Planarization* algorithm of the OGDF library [6] has been used for the class diagrams.

4.1 Results

(O1) Aspect ratio. Table 1 gives an overview of the resulting aspect ratios of drawn diagrams; it can be seen that for every class of graphs less than 50 % of the drawings have an acceptable aspect ratio.

The results for the flattened Ptolemy graphs and the North graphs are also shown in more detail in Fig. 4. Each bar accumulates the number of graphs with an aspect ratio within an interval of size 0.5, specified by the values left and right of the respective bar. The four marked bars depict the desired interval $1 \leq r \leq 2$ of the aspect ratio r, and the right-most interval contains all graphs with $r > 9.5$. Only about 36 % (Ptolemy graphs) and 30 % (North graphs) have an aspect ratio in the desired interval. While the drawings with aspect ratio below 1 (presuming left-to-right layout) can be improved with alternative layer assignment methods discussed in Sect. 3, no current methods can handle the remaining 40 % and 18 % with an aspect ratio above 2 because the minimal number of layers is determined by the longest paths of these graphs (see Sect. 5).

[4] `http://www.ohloh.net/p/kieler`

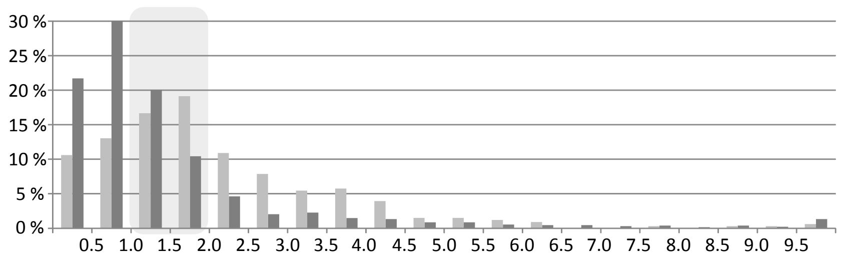


Fig. 4. Detailed aspect ratios of flattened Ptolemy graphs (light-gray) and North graphs (dark-gray) drawn with the KLay Layered (cf. Table 1 rows 1 and 5).

Table 2. Percentage of whitespace per total area of a drawn diagram and the increase incurred at each expansion. Figures are the average of 11 nested diagrams with 9 or more compound actors.

Expanded Compound Nodes	1	2	3	4	5	6	7	8	9
Avg. Whitespace	71.8	80.4	85.1	87.5	87.8	89.9	91.1	91.2	91.8
Avg. Increase	-	8.6	4.7	2.4	0.3	2.1	1.2	0.2	0.5

(O2) Whitespace of compound diagrams. Table 2 shows the average amount of whitespace of the successively expanded Ptolemy diagrams. It can be seen that the amount of whitespace grows continuously with an increasing number of expanded compound nodes. Diagrams with more than nine compound nodes reach up to 97 % whitespace. Better methods have to be found to reduce the unused space for diagrams containing compound nodes (such as in Fig. 1).

(O3) Whitespace and planarization methods. The amount of whitespace of the eight class diagrams after applying the planarization-based layout can be seen in Table 3. The same average value of about 72 % can be observed for both types of diagrams, the ones with hidden and with visible details. We found that the specified spacing around nodes has no significant impact on the relative amount of whitespace. The results in Table 3 originate from a spacing value of 50. Spacing values of 20 or 90 yield similar whitespace ratios of $70 - 73$ %.

We applied the force-based *FDP* algorithm of Graphviz in addition to the planarization-based method for a comparison. The algorithm was configured to produce drawings as compact as possible while remaining legible. This setup yields 18 % whitespace on average. While we believe that the planarization-based drawings are more comprehensible because they produce fewer crossings, we conclude that there are layout algorithms producing drawings with less than a third of the whitespace when relaxing the goal to minimize edge crossings. In terms of area, the examined class diagrams require an average of 5.3 monitors when using planarization-based method, whereas all but three diagrams would fit on a single monitor using force-directed methods (0.7 on average).

Table 3. Amount of whitespace per class diagram. The "plain" row refers to classes shown without any methods, and "detailed" to classes with visible methods. The first two rows are results of a planarization-based layout algorithm, the last row results from the FDP algorithm of Graphviz with overlap mode set to `COMPACT`. For all results a spacing of 50 was specified.

	batik	diagrams	kexpr	kgraph	kiml	klighd	sc	sccharts	**Avg.**
plain [%]	77.5	70.2	69.5	68.6	79.7	76.3	65.0	63.4	**71.3**
detailed [%]	75.7	73.5	70.1	71.3	76.8	66.6	71.2	65.2	**71.3**
force-plain [%]	9.6	28.4	28.0	18.6	14.3	17.5	16.6	11.2	**18.0**

5 Open Problems

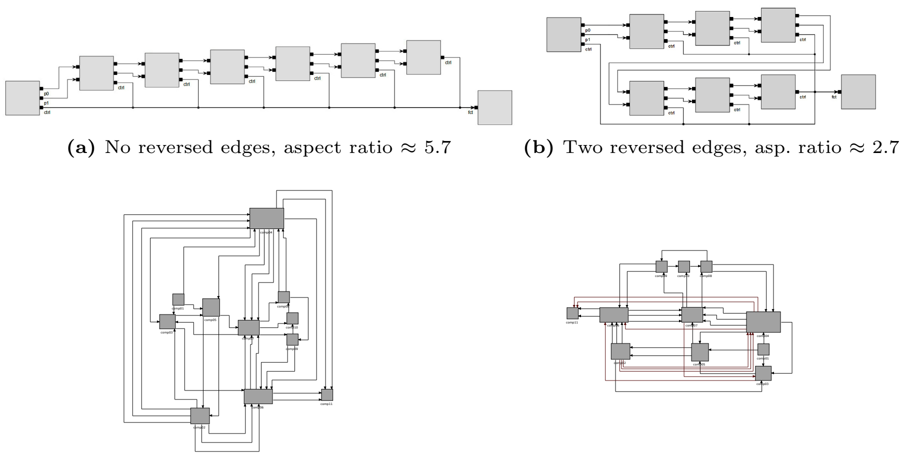
We present four directions for future research: minimization of longest paths and improved layer assignment for the layer based approach, and partial orthogonalization and graph decomposition for the topology-shape-metrics approach.

1. Longest paths. The minimal number of layers for an acyclic graph is determined by its longest path, since all nodes of a path have to be assigned to different layers. Hence the longest path limits the freedom of layer assignment algorithms regarding their goal of finding a layering that allows a compact drawing. This fact has been neglected in previous approaches to cycle elimination, which only address the number of reversed edges [12]. We propose to generalize the edge reversal phase such that it targets two optimization goals: Minimize the number of reversed edges and minimize the length of the longest path.

An important question is how to balance the two optimization goals. Ideally, this balancing should be controllable with a parameter, allowing to adapt the drawings to the needs of individual applications.

A possible heuristic for the generalized edge reversal problem is to separate the two goals: first make the graph acyclic using a standard method [12], then shorten the longest path by reversing additional edges. For acyclic graphs a longest path can be found in linear time. As a first experiment, we simply reverse the middle edge of each longest path, if that does not introduce new cycles. An example is shown in Fig. 5(a) and (b). We evaluated this approach using 180 Ptolemy diagrams and found that the aspect ratio of 76 diagrams with original aspect ratios above 2 decreased on average by a factor of 0.59. For none of the tested diagrams the aspect ratio is reduced below 1. The results include improvements from 8.1 to 3.1 and from 6.0 to 1.9.

2. Layer assignment. In the example in Fig. 5(b) it can be observed that reducing the number of layers can lead to a higher number of edge crossings. It is an open problem how to address this in the layer assignment phase. Furthermore, the aspect ratio of the layering shall be optimized, which can be defined as follows. Given a layer assignment $V_1, \ldots, V_k$ with k layers and $n_{\max} = \max\{|V_i| : 1 \leq i \leq k\}$, the predicted aspect ratio is $k/n_{\max}$ for left-to-right layout and



(a) No reversed edges, aspect ratio ≈ 5.7 **(b)** Two reversed edges, asp. ratio ≈ 2.7

(c) Topology-shape-metrics, 4 crossings **(d)** Partial orthogonalization, 24 crossings

Fig. 5. Improving the compactness of drawings. *Layer-based:* (a) and (b) depict how by reversing additional edges, the longest path in an acyclic graph can be shortened, allowing a better aspect ratio. *Planarization-based:* Two orthogonal drawings of the same graph. The drawing (c) has been produced by removing 8 edges from the graph, applying a topology-shape-metrics layout, and reinserting the missing edges manually. By this we obtain 20 additional edge crossings compared to (d), but only 58% of the drawing area.

$n_{\max}/k$ for top-to-bottom layout. For drawings with large nodes it may be appropriate to consider the nodes' dimensions in this prediction. Another possible extension is to solve the edge reversal problem as part of the layer assignment process. This would allow more effective choices of which edges to reverse such that compact drawings are possible.

3. Partial orthogonalization. We propose to temporarily remove a subset of the edges from the topology-shape-metrics approach and reinsert them in a post-processing step using existing routing methods such as the algorithm of Wybrow et al. [19]. An example is shown in Fig. 5(b) and (c).

– *Edge removal during planarization.* Edges with many crossings introduce many dummy nodes, potentially increasing the size of the drawing. Instead of inserting such edges during planarization, they could be removed.
– *Edge removal during orthogonalization.* Edges with many bend points require additional space in the compaction phase. A possible heuristic is to compute an orthogonal representation, remove edges with too many bend points, and then repeat the process iteratively.

4. Graph decomposition. Decomposing graphs and processing the components with force-based drawing methods is known from multi-level approaches [3]. Archambault et al. extended this idea to also consider other layout methods such as tree layouts and circular layouts [2]. The topology-shape-metrics method, however, has not been considered for graph decomposition yet. By putting together separately processed components, more detailed control on the area and aspect ratio of the composed drawing could be obtained.

– *Trees.* We remove subgraphs which are free trees from the input graph, so we get one core graph plus a collection of tree subgraphs. There are many options for drawing trees, and this freedom can be exploited to optimize compactness, e. g. the face into which to embed a tree can be freely chosen.
– *Biconnected components.* Another possibility is to use the decomposition of the graph into its biconnected components. We can use a similar approach as for trees, identifying a large core graph and cutting out some subgraphs which are only connected at cut vertices. A more complex variant would be to use this approach recursively, i. e. for each removed subgraph we can again cut out subgraphs connected at cut vertices.

References

1. Andreev, R., Healy, P., Nikolov, N.S.: Applying ant colony optimization metaheuristic to the DAG layering problem. In: Proceedings of the Parallel and Distributed Processing Symposium (IPDPS'07). pp. 1–9 (2007)
2. Archambault, D., Munzner, T., Auber, D.: Topolayout: Multilevel graph layout by topological features. IEEE Transactions on Visualization and Computer Graphics 13(2), 305–317 (2007)
3. Bartel, G., Gutwenger, C., Klein, K., Mutzel, P.: An experimental evaluation of multilevel layout methods. In: Proceedings of the 18th International Symposium on Graph Drawing (GD'10). LNCS, vol. 6502, pp. 80–91. Springer (2011)
4. Bridgeman, S.S., Di Battista, G., Didimo, W., Liotta, G., Tamassia, R., Vismara, L.: Turn-regularity and optimal area drawings of orthogonal representations. Computational Geometry 16(1), 53–93 (2000)
5. Di Battista, G., Garg, A., Liotta, G., Parise, A., Tamassia, R., Tassinari, E., Vargiu, F., Vismara, L.: Drawing directed acyclic graphs: An experimental study. In: Proceedings of the Symposium on Graph Drawing (GD'96), LNCS, vol. 1190, pp. 76–91. Springer (1997)
6. Eiglsperger, M., Gutwenger, C., Kaufmann, M., Kupke, J., Jünger, M., Leipert, S., Klein, K., Mutzel, P., Siebenhaller, M.: Automatic layout of UML class diagrams in orthogonal style. Information Visualization 3(3), 189–208 (2004)
7. Eiglsperger, M., Kaufmann, M.: Fast compaction for orthogonal drawings with vertices of prescribed size. In: Proceedings of the 9th International Symposium on Graph Drawing (GD'01). LNCS, vol. 2265, pp. 124–138. Springer (2002)
8. Frey, P., von Hanxleden, R., Krüger, C., Rüegg, U., Schneider, C., Spönemann, M.: Efficient exploration of complex data flow models. In: Proceedings of Modellierung 2014. Vienna, Austria (Mar 2014)
9. Friedrich, C., Schreiber, F.: Flexible layering in hierarchical drawings with nodes of arbitrary size. In: Proceedings of the 27th Australasian Conference on Computer Science (ACSC'04). pp. 369–376. Australian Computer Society, Inc. (2004)
10. Gansner, E.R., Koutsofios, E., North, S.C., Vo, K.P.: A technique for drawing directed graphs. Software Engineering 19(3), 214–230 (1993)

11. Healy, P., Nikolov, N.S.: How to layer a directed acyclic graph. In: Proceedings of the 9th International Symposium on Graph Drawing (GD'01). LNCS, vol. 2265, pp. 563–566. Springer (2002)
12. Healy, P., Nikolov, N.S.: Hierarchical drawing algorithms. In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization, pp. 409–453. CRC Press (2013)
13. Klau, G.W., Klein, K., Mutzel, P.: An experimental comparison of orthogonal compaction algorithms. In: Proceedings of the 8th International Symposium on Graph Drawing (GD'00). LNCS, vol. 1984, pp. 37–51. Springer (2001)
14. Klau, G.W., Mutzel, P.: Optimal compaction of orthogonal grid drawings. In: Proceedings of the 7th International IPCO Conference on Integer Programming and Combinatorial Optimization. LNCS, vol. 1610, pp. 304–319. Springer-Verlag (1999)
15. Klau, G.W.: A Combinatorial Approach to Orthogonal Placement Problems. Ph.D. thesis, Universität des Saarlandes (Sep 2001)
16. Nachmanson, L., Robertson, G., Lee, B.: Drawing graphs with GLEE. In: Hong, S.H., Nishizeki, T., Quan, W. (eds.) Graph Drawing, Lecture Notes in Computer Science, vol. 4875, pp. 389–394. Springer Berlin Heidelberg (2008)
17. Nikolov, N.S., Tarassov, A., Branke, J.: In search for efficient heuristics for minimum-width graph layering with consideration of dummy nodes. Journal of Experimental Algorithmics 10 (2005)
18. Schulze, C.D., Spönemann, M., von Hanxleden, R.: Drawing layered graphs with port constraints. Journal of Visual Languages and Computing, Special Issue on Diagram Aesthetics and Layout 25(2), 89–106 (2014)
19. Wybrow, M., Marriott, K., Stuckey, P.J.: Orthogonal connector routing. In: Proceedings of the 17th International Symposium on Graph Drawing (GD'09). LNCS, vol. 5849, pp. 219–231. Springer (2010)