INSTITUT FÜR INFORMATIK

Wrapping Layered Graphs

Ulf Rüegg and Reinhard von Hanxleden

Bericht Nr. 1803
February 2018
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Department of Computer Science
Kiel University
Olshausenstr. 40
24098 Kiel, Germany

Wrapping Layered Graphs

Ulf Rüegg and Reinhard von Hanxleden

Report No. 1803
February 2018
ISSN 2192-6247

E-mail: {uru,rvh}@informatik.uni-kiel.de

An abridged version of this work is published at the
10th International Conference on the Theory and Application of Diagrams,
Edinburgh, Scotland, June 2018.

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

Abstract

We present additions to the widely-used layout method for directed acyclic graphs of Sugiyama et al. [16] that allow to better utilize a prescribed drawing area. The method itself partitions the graph's nodes into *layers*. When drawing from top to bottom, the number of layers directly impacts the *height* of a resulting drawing and is bound from below by the graph's longest path. As a consequence, the drawings of certain graphs are significantly taller than wide, making it hard to properly display them on a medium such as a computer screen without scaling the graph's elements down to illegibility. We address this with the *Wrapping Layered Graphs Problem (WLGP)*, which seeks for *cut indices* that split a given layering into *chunks* that are drawn side-by-side with a preferably small number of edges wrapping backwards. Our experience and a quantitative evaluation indicate that the proposed wrapping allows an improved presentation of narrow graphs, which occur frequently in practice and of which the internal compiler representation SCG is one example.

Contents

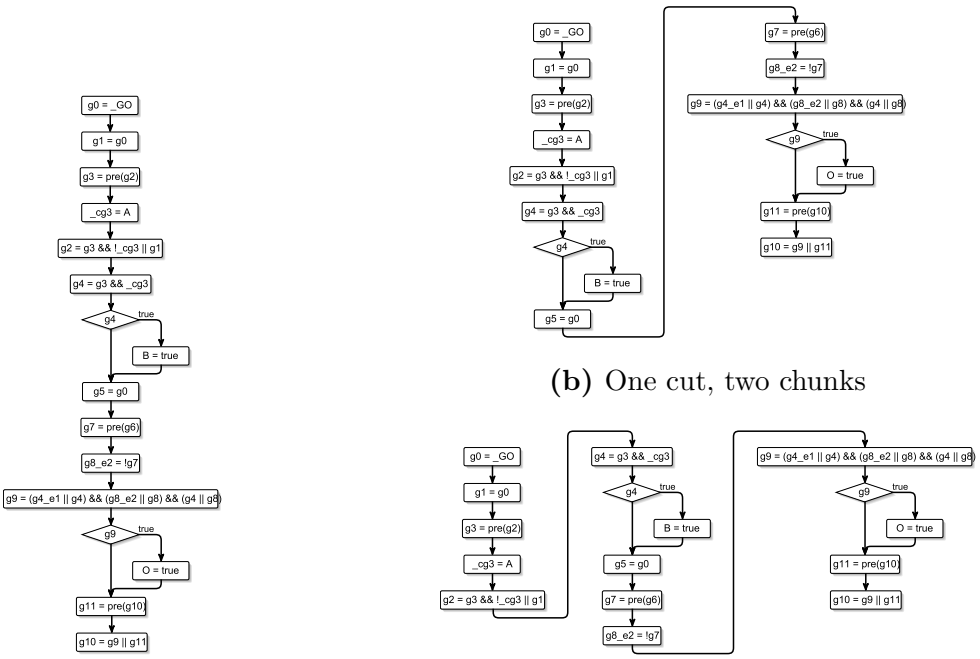
1 Introduction

The *layer-based layout approach* is a successful method to lay out directed acyclic graphs. In its original form it has been presented by Sugiyama et al. in 1981 [16] and since then has been refined for many use cases. Healy and Nikolov give a comprehensive survey of the state of the literature in 2013 [8]. The approach's central idea is to assign nodes to indexed *layers* such that edges point from layers with lower index to layers with higher index. The overall layout task is split into three consecutive phases: *layering*, *crossing minimization*, and *coordinate assignment*. To allow cyclic graphs, an additional initial *cycle breaking* phase can be added to make the graph acyclic, and a final *edge routing* phase can be added to support different edge routing styles, e. g. orthogonal or using splines.

If a graph is laid out from top to bottom, the number of used layers directly impacts the height of the resulting drawing and is bound from below by a graph's longest path. Thus, an inherent problem of the layering process is that graphs with long longest paths may yield drawings which are significantly taller than wide, resulting in unfortunate aspect ratios. Certain graph types occurring in practice encourage such drawings by nature. See, for instance, Figure 1.1a, which shows the drawing of a sequentialized *Sequentially Constructive Graph (SCG)* [17], a type of control flow diagram, where each node represents an execution step of a program. The larger the program represented by the SCG, the taller the resulting drawing will become. Consequently, at some point either only a small part of the drawing can be displayed on a monitor, or the graph's drawing has to be scaled down to illegibility. However, in particular the labels within the nodes require a presentation in which they can be read by a user.

Contributions. We present methods that integrate seamlessly with the layer-based approach and aim at splitting a graph's layering into multiple *chunks* which then can be drawn side by side as demonstrated in Figures 1.1b and 1.1c. We refer to this process as *wrapping* and call the points where a layering is split *cuts*. The input to our method is a prescribed drawing area which determines the number of required chunks and the concrete cut positions such that a resulting drawing uses the available drawing area to its full potential. Note that while the edges connecting one chunk with another chunk point upwards, something that the original approach seeks to avoid, the chunks are well-separated and the overall flow of the diagram remains clearly visible. Our methods are implemented in the Eclipse Layout Kernel (ELK) open-source project[1].

[1] `https://www.eclipse.org/elk/`



(a) No cuts, one chunk

(b) One cut, two chunks

(c) Two cuts, three chunks

Figure 1.1. A sequentialized SCG drawn with our method and different numbers of cuts. The aspect ratios of the drawings vary significantly, each well-suited for a certain prescribed drawing area.

Outline. The paper is structured as follows. Next, we discuss related work and the terminology we use (Chapter 2). Chapter 3 then first presents heuristics that permit a single edge to point backwards between a subsequent pair of chunks (Section 3.1). Building on these heuristics, Section 3.2 discusses a method that allows arbitrary numbers of edges to be cut, while trying to keep these numbers low to preserve legibility. Chapter 4 presents final observations.

Related Work. Traditional layering methods minimize the number of used layers or the overall edge length and have no further control over the width and height [4, 6]. In subsequent work, several authors aimed at either restricting the width [3, 7, 12, 1], i. e. the maximum number of nodes in any layer, or at reducing the height [10, 13, 5], or at influencing both, e. g. with the goal of a certain aspect ratio [7, 11]. None of these can overcome the bound on the height imposed by a graph's longest path. Rüegg et al. propose to combine the cycle breaking and the layering phase [14]. Minimizing a linear combination of the overall edge length and the number of upward pointing edges, they are able to alter the aspect ratio of a final drawing using different weights for the two individual terms. However, since it is not immediately clear how to select the weights to obtain a particular aspect ratio and since the resulting drawings may look entirely different from drawings created with traditional methods, they suggest to investigate *bluntly cutting* the layering, which is the motivation for this work.

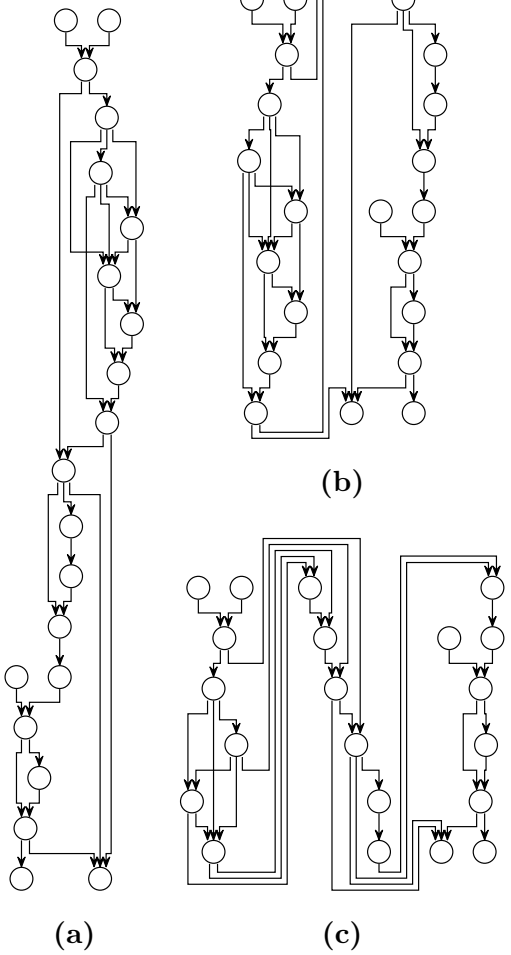


Figure 1.2. Three drawings of the same graph using our method, each time with a different number of cuts.

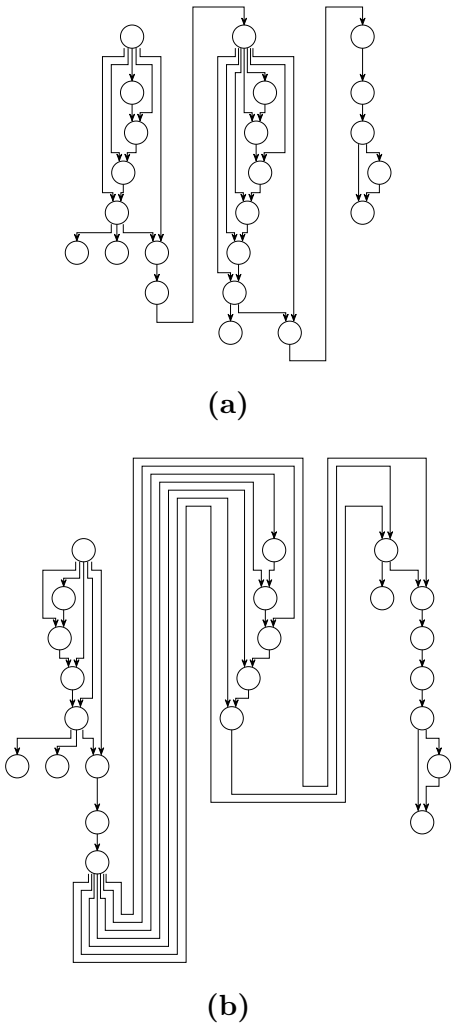


Figure 1.3. Selecting unfortunate cut indices can result in poor and cluttered drawings. In (b) the two cut indices of (a) were increased and decreased by one, respectively.

2 Preliminaries

A *layering* L of a directed acyclic graph $G = (V, E)$ is a partitioning of the graph's nodes into distinct *layers* $L = (L_1, \ldots, L_m)$. Let $L(v)$, $v \in V$, denote the index of v's layer. In the context of Sugiyama et al.'s method, it is required that $\forall (u, v) \in E$: $L(v) - L(u) \geq 1$ holds. Furthermore, from the crossing minimization phase on a *proper* layering is required, in which all edges must be *short*: $L(v) - L(u) = 1$, $(u, v) \in E$. A proper layering can be constructed by splitting all edges that span multiple layers, introducing a *dummy node* in each spanned layer.

Let a graph $G = (V, E)$ and a corresponding layering L be given. A *cut index* c is an index out of $[2, \ldots, m]$ where $m \geq 2$ is assumed since otherwise splitting a layering is not required. A cut index c may be *valid*. The exact definition of valid depends on the use case and will be defined more precisely later on. If c is not valid, it is *forbidden*. An edge $e = (u, v) \in E$ is *spanning* an index i with $1 < i \leq m$ if $u \in L_j, j < i$ and $v \in L_k, k \geq i$. An ordered set of monotonically increasing valid cut indices $C = (c_1, \ldots, c_n)$ partitions a layering L into $n + 1$ *chunks*:

$$S_1, \ldots, S_{n+1} = (L_1, \ldots, L_{c_1 - 1}), (L_{c_1}, \ldots), \ldots, (\ldots, L_{c_n - 1}), (L_{c_n}, \ldots, L_m).$$

Let $m' = \max_{1 \leq i \leq n+1} |S_i|$. A new layering $L' = (L'_1, \ldots, L'_{m'})$ can be constructed by subsequently adding the nodes of each S_i to the layers of L', each time starting at L'_1. An edge that spans a cut index is said to be *cut* in the new layering.

To compute reasonable cut indices, it is necessary to be aware of a graph's dimensions. In the literature, the *height* of a layering is often defined as the number of used layers, and the *width* as the maximum number of nodes in any layer [7], where nodes are assumed to have a unit dimension. For the width the contribution of dummy nodes can either be considered or neglected. In our context, a node v has a width $w(v)$ and a height $h(v)$. The methods described in this paper shall therefore use more precise estimations. For this we first define the dimensions of each individual layer L_i as:

$$w(L_i) = \sum_{v \in L_i} (w(v) + s) \qquad h(L_i) = \max_{v \in L_i} (h(v) + s).$$

The constant s is a prescribed spacing value to be preserved between pairs of nodes. Since we approximate, we are fine with summing up one s too much. The overall width and height of layering L can then be estimated as:

$$w(L) = \max_{L_i \in L} w(L_i) \qquad h(L) = \sum_{L_i \in L} h(L_i).$$

While this is more accurate than considering every node to have the same dimension, the quality of the estimations still depends on the complexity of the used graph instances.

In particular, the definitions neglect the impact of edges that are to be routed in-between layers, something that is all the more relevant the denser graphs get. Furthermore, remember that the goal is to draw the computed chunks next to each other with backward wrapping edges in-between. Generally, the subsequent steps of the layer-based approach will ensure that those edges are drawn straight in a final drawing. As a consequence, the nodes within the new layering's layers are not as close to each other as assumed by the width estimates defined above. To address this, further width and height estimates are defined as follows that consider the computed chunks. The dimensions of the individual chunks $S_1, \ldots, S_{n+1}$ are:

$$w(S_i) = \max_{L_j \in S_i} w(L_j), \qquad h(S_i) = \sum_{L_j \in S_i} h(L_j).$$

The chunk-based estimates of the new layering L' then are:

$$w_S(L') = \sum_{1 \leq i \leq n+1} w(S_i), \qquad h_S(L') = \max_{1 \leq i \leq n+1} h(S_i).$$

Note, however, that this time $h_S(L')$ intuitively feels inferior to $h(L')$: $h_S(L')$ may underestimate the height of the individual layers since the different chunks share the same set of layers (a tall node would thus increase the height of all chunks, not only of the chunk it belongs to as assumed by $h_S(L')$). Thus, the combination of $w_S(L')$ and $h(L')$ would be expected to be the most accurate one. For ease of presentation and since an informal experiment with the test graphs used here did not show significant differences in the results, the following explanations use the chunk-based estimates only. Nevertheless, it may be worthwhile to explore the impact of different estimates in greater detail in the future. Now to the problem we seek to solve:

Problem (Wrapping Layered Graphs (WLGP))**.** Given graph $G = (V, E)$ with layering L and a prescribed drawing area, the problem is to find a set of valid cut indices C such that for the induced layering L'

1. the prescribed drawing area is used as good as possible, and
2. the number of edges that point into the "wrong" direction is kept small.

We assess the first point using the *max scale* measure [14]. It states for a given prescribed drawing area $\mathcal{R} = (r_w, r_h)$ the largest scale factor s that can be used to display a graph's drawing within $\mathcal{R}$. In our case it is computed as: $s = \min\{{}^{r_w}/_{w_S(L')}, {}^{r_h}/_{h_S(L')}\}$. When comparing two drawings, D_1 and D_2, of the same graph with max scale values s and s', the *max scale ratio* is $r = {}^{s}/_{s'}$, and the drawing with the larger max scale value is considered to be superior. Thus, D_1 is superior if $r > 1$. The measure does not allow to compare max scale values originating from different graphs. Further note that point 2) of the problem does not relate to the number of cuts but to the number of edges that span a certain cut index.

3 Wrapping Layered Graphs

We first consider a variant of WLGP that is restricted to wrapping a single edge per cut and solely optimizes point 1), since it is far easier to implement within the layer-based approach. It is referred to as WLGPse and discussed in the next section. Solving the unrestricted WLGP requires more intricate adaptations of the layer-based approach but works for an arbitrary number of wrapped edges and is discussed in Section 3.2.

3.1 Single-Edge

A cuts is *valid* for WLGPse if it is in-between a pair of layers that is connected by exactly one edge: $|\{e \in E : e \text{ spans } c\}| \leq 1$. The wrapping can then be realized using a single processing element after the crossing minimization phase, constructing the new layering according to the computed cut indices as outlined during the preliminaries. The order of the nodes within the layers can be derived straightforwardly and has largely been determined by the crossing minimization. Wherever a backward wrapping edge is to be inserted, the processor creates a chain of long edge dummies.

Next, two straightforward and fast heuristics to solve WLGPse are presented and evaluated based on a set of graphs from practice afterwards. The first heuristic tries to come close to a desired aspect ratio of the final drawing, hence it is referred to as *aspect ratio-driven (ARD)*. The second one tries, at the cost of further work, to maximize the max scale value, hence it is referred to as *max scale-driven (MSD)*.

ARD. Let a graph G and a layering L be given. The desired aspect ratio of the new layering L' is $a = {}^{w_S(L')}/_{h_S(L')}$. The number of chunks z the old layering has to be split into in order to achieve a can be estimated as follows, where one assumes $w_S(L') \approx z \cdot w(L)$ and $h_S(L') \approx {}^{h(L)}/_{z}$. Given z, a corresponding set of potential cut indices C can be defined as well:

$$z = round\left(\sqrt{\frac{a \cdot h(L)}{w(L)}}\right), \qquad C = \left\{\left\lceil \frac{|L|}{z+1} \cdot i \right\rceil + 1,\ 1 \leq i \leq z\right\}.$$

z can be bound from above as there cannot be more chunks than layers: $z = \min\{z, |L| - 1\}$. Since ${}^{|L|}/_{z+1} \geq 1$, no two elements of C are equal.

MSD. Aiming at a specific aspect ratio does not necessarily result in a drawing that uses a prescribed drawing area to its full potential. The aforementioned ARD heuristic simply splits the graph such that the resulting layering roughly matches the aspect

ratio of the prescribed drawing area. The MSD heuristic takes a different approach by directly addressing the max scale measure. To better understand its idea, remember the definition of max scale: $\min\{{}^{r_w}/_{w}, {}^{r_h}/_{h}\}$, where r_w and r_h are the width and height of a prescribed drawing area, and w and h are the width and height of a produced drawing. Clearly, to get a large max scale value, small values of w and h are advantageous. Following this, MSD's goal is to choose z cut indices in a manner such that the prescribed drawing area's aspect ratio is matched roughly and such that either w or h is minimized.

Let a graph G and layering L be given. The tentative height of the layering L' to be constructed is the maximum of sums (each sum represents the height of a single chunk). Given z, a set of cut indices that minimizes this maximum can be calculated efficiently. A list ch of length $m = |L|$ stores at each index i the cumulative height of all layers at smaller indices than i. Thus, $ch(m)$ holds the overall height. Every chunk should consist of layers whose heights roughly sum up to ${}^{ch(m)}/_{z+1}$. The algorithm then finds the cut indices by iterating ch from start to end and collecting each index at which the desired sum is exceeded. Since MSD must know z to do so, it uses the z calculated by ARD as an initial guess. Still, MSD uses a measure to assess the quality of the calculated cut indices that is independent of z, it can thus try different values for z, i. e. slightly increase it or decrease it. During the upcoming evaluations this is denoted by MSD-k, where k indicates the amount of freedom. For instance, $k = 1$ and $z = 3$ would allow MSD to try layerings with two, three, and four chunks.

We did not consider the tentative width of L' so far: improvidently minimizing the sum of maximums (where the maximums are the widths of the individual layers) is very likely to result in unbalanced cut indices; a "good" set of cut indices, w.r.t. the minimal width, would tend to put the narrowest nodes in chunks on their own and put as many nodes as possible in chunks width very wide nodes. Nevertheless, our evaluations suggest that the width case should not be neglected completely and should thus be investigated in future work.

Validification. Remember that it is not allowed to cut at every index. As a consequence, the forbidden indices in C have to be altered. We refer to this process as *validification*. Two interchangeable validification strategies are evaluated here: The first strategy increments every forbidden cut index until it is valid. To preserve the original distances between consecutive cut indices, the remaining cut indices are increased accordingly. This can result in indices larger than the number of original layers. Such indices are simply discarded. The method is referred to as Greedy. The second strategy proceeds in a more sophisticated fashion. For each forbidden index it seeks the closest valid index position. For this, it iterates all possible index positions from 2 to $|L|$ and keeps track of the last valid index. When a forbidden cut index is encountered, the iteration is continued until either a valid (larger) index is found or the distance to the next valid index is larger than the distance to the last valid index. If both distances are the same, we arbitrarily select the smaller index. The intention behind this method is to distribute the cut indices more evenly and to avoid having to discard indices. It is referred to as Lookback.

3.1.1 Evaluation

We evaluated the two heuristics, ARD and MSD, using 135 sequentialized SCGs[1] with 10 to 227 nodes, 41.1 on average, and 11 to 253 edges, 45.5 on average. All of these graphs permit valid cut indices. The desired separation value was set to 20 pixels. For every graph the two heuristics were used to compute five different drawings with a particular drawing area in mind, and four questions were to be answered:

Q1 How do the max scale values of drawings created with ARD and MSD compare to drawings where no wrapping was applied?

Q2 Which validification method performs better?

Q3 How close are the estimations of width and height to the width and height of the final drawing?

Q4 What is a reasonable value of k for MSD-k?

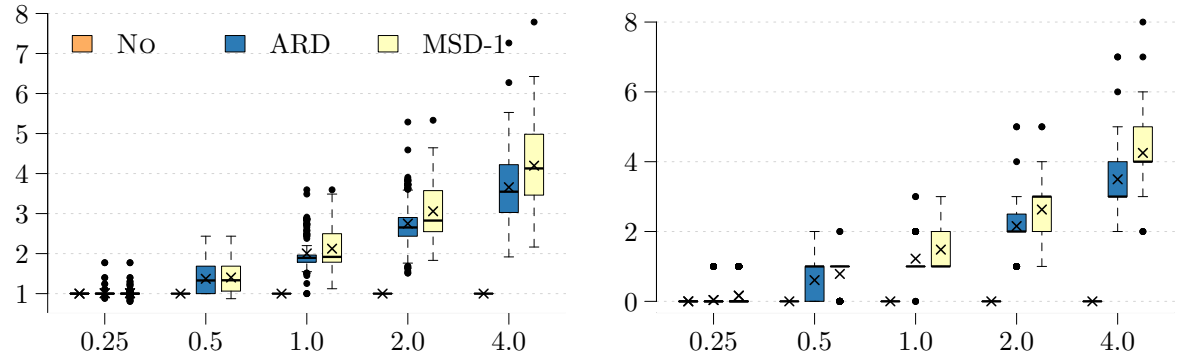
The concrete dimensions of the drawing areas do not matter for the evaluation, which is why the plots simply show the target aspect ratios. We use boxplots[2] to present the results since they give a better overview of the distribution of the data than mere averages. Remember that for the max scale values only comparisons between various methods for the *same* target drawing area are possible and that larger values are deemed superior.

Regarding the general quality of the heuristics' results (Q1) consider Figure 3.1, which shows max scale ratios with relation to no wrapping. It can be seen that both heuristics are advantageous for target aspect ratios of 1.0 to 4.0, with MSD-1 producing drawings that can be scaled more than four times larger on average for 4.0 than when no wrapping is applied. Also, starting with a target aspect ratio of 1.0, neither heuristic results in worse max scale values than no wrapping. For 0.25 the heuristics compute a single cut for 5 (ARD) and 22 (MSD) graphs without any significant improvement in max scale, indicating that the traditional layerings already fit the drawing area best; something one would expect. For the tested graphs MSD-1 produces on average slightly larger max scale values than ARD, which stem from an on average larger number of cuts.

We do not show plots for the remaining three questions but only summarize our findings. Regarding Q2, we found that the LOOKBACK method is superior. Applied after both ARD and MSD, it consistently produces equal or better max scale values on average for the final drawings, particularly for larger aspect ratios. Following this observation, the plots in Section 3.2.4 are produced with the LOOKBACK method. The tested SCGs usually result in a layering with no more than one original node per layer, thus the width and height estimations should be rather accurate (Q3). This turned out to be the case with an average deviation of 12.8 % of the estimated width from the final width and 3.7 % for the height. Regarding Q4, we found that for target aspect ratios larger than or equal to 0.5, setting k to 1 instead of 0 produces slightly better max scale

[1] `https://git.rtsys.informatik.uni-kiel.de/projects/KIELER/repos/models-public/`

[2] Created with R 3.2.3; dots indicate outliers, crosses indicate mean values.



(a) Max scale ratio w.r.t. No

(b) Number of cuts

Figure 3.1. Comparing results for final drawings either created with a wrapping heuristic, ARD or MSD-1, or without wrapping (No). (a) and (b) show different measures as stated in their captions (y axis) plotted against varying target aspect ratios (x axis).

values, while further increasing k does not improve the results. We observed that an increased k yields a worse max scale value for certain graph instances. This is explained by the fact that the validification method is executed after the heuristic, i. e. MSD-1 finds better cut indices than MSD-0 but the validified indices of MSD-0 allow for final drawings with a larger max scale value.

3.2 Multi-Edge

The wrapping presented in the previous section addresses a very restricted scenario in which only one backward wrapping edge per cut is allowed. This restriction allows good estimations on the width and height of the final drawing and allows an easy implementation. However, the desire to leverage the space of a given presentation medium to its full potential remains as well for graphs for which it is not always possible to wrap only a single edge per cut. This is, for instance, the case for denser graphs where the space between a pair of layers is usually populated by many edges. Cutting many edges results in a large number of backward wrapping edges, which in turn results in poor and cluttered final drawings. See Figures 1.2 and 1.3 for examples. Good cuts are therefore between pairs of layers that have a small number of edges in-between them.

3.2.1 Edge-Aware Cut Improvement

Let L denote a given layering computed with a traditional layering method. We determine an initial set of cut indices C using ARD or MSD and afterwards apply a greedy improvement strategy (ECI) to avoid cutting too many edges. ECI tries to slightly alter the given indices in order to reduce the number of cut edges. Let I denote the set of possible indices, i. e. $I = \{2, \ldots, |L|\}$, and let $spans(i)$ denote the number of edges spanning index $i \in I$.

The method uses a cost function *cost* to rate the quality of an index as a cut. Here, the following function is used:

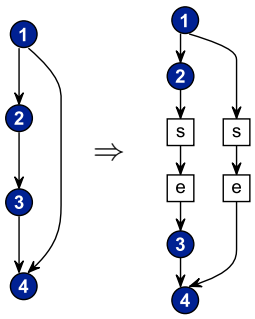
$$cost(i) = spans(i) + \delta(i)^e, \qquad \text{where } \delta(i) = \min_{c \in C} \ |i - c|$$

is the minimal distance of an index i to any of the cut indices of C. The preservation of the existing cuts can be weighted larger by increasing e, where the fact that e is an exponent already strongly limits their freedom. To give an example, for $e = 2$ an index with a distance of two is deemed better than a given cut index that would save more than four edge reversals. Using the cost function, ECI calculates the cost of each index and greedily picks the index with lowest cost. Afterwards the cut index in C that is closest to the picked index is removed, thus resulting in different cost values for the next index to pick. The algorithm runs in $O(|C| \cdot |L|)$ time.

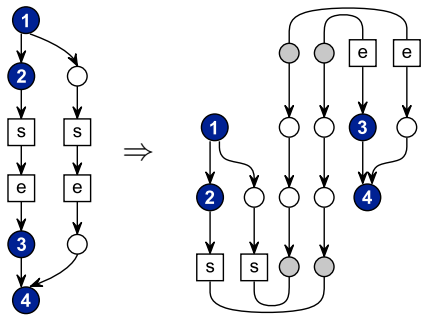
3.2.2 Technical Integration

Next, we explain how the multi-edge wrapping procedure can be integrated into the layer-based approach without changing an existing implementation. It consists of three ameliorations, which are also illustrated in Figure 3.2.

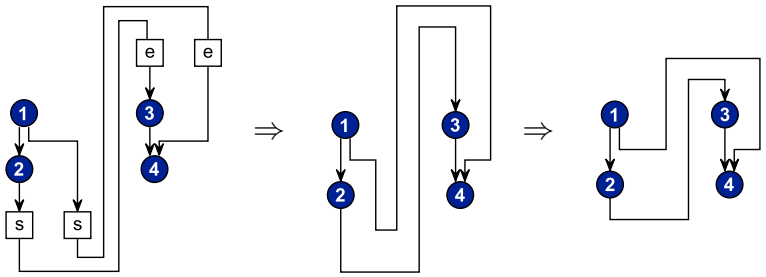
First, immediately after the layering phase, before the layering is turned into a proper layering, dummy nodes are inserted wherever a cut is planned (Figure 3.2a). More precisely, for a planned cut between two layers i and j, two new layers are inserted and every edge that either connects or spans i and j is split by adding two *breaking point* dummies in the new layers. The dummies are called `s` and `e` for start and end, and while one can surely come up with an implementation that manages without them, they are helpful as clear markers in both documentation and code. After inserting `s` and `e`, the existing implementation takes care of making the layering proper by adding dummy nodes for long edges, and it executes the crossing minimization phase. There, the breaking point dummies can be understood as long edge dummies and therefore do not negatively influence the number of edge crossings. Second, the new layering L' can be formed as depicted in Figure 3.2b. The `e` dummy nodes of every pair of breaking point dummies in a certain layer can be moved to the very first layer of the new layering (in given order). Any successive nodes are moved to the next layer and so forth. The edge that connects the two dummies now runs from the very first layer to one of the later layers of the new layering. Since such a layering would be infeasible, dummy nodes must be introduced in the same way as they are introduced for long edges after the layering phase. Note that if the chains of long edge dummies are introduced in the reversed in-layer order than the breaking point dummies, no new edge crossings have to be introduced. Further note that the edges that connect the gray circled dummy nodes to `s` and `e` are normally prohibited as they connect nodes that are located in the same layer. Our implementation already comprises methods that handle such cases, originally designed by Schulze et al. to handle *inverted ports* [15]. Alternatively, the gray dummy nodes could be dropped and `s` and `e` could be connected to the adjacent white dummy nodes. This would then require additional edge routing code that makes



(a) Dummy nodes for breaking points (`s` and `e`) are introduced after the layering phase based on calculated cut indices.



(b) After crossing minimization, the breaking point dummies are used to alter the layering. In a top to bottom layout the chunks defined by the cut indices are placed as columns next to each other. Chains of dummy nodes are inserted for the edges between `s` and `e`.



(c) After the edge routing phase and after merging the dummy nodes of long edges, the `s` and `e` dummy nodes remain. They are removed and the three involved edges are simply joined to form the final edge path, which can be improved further using one-dimensional compaction techniques (right drawing).

Figure 3.2. Illustrating the wrapping procedure. (a), (b), and (c) are three new intermediate steps to be executed at different points during the overall execution of the layer-based layout.

sure the corresponding edges are routed around the `s` and `e` nodes. Third, after edge routing and after removing long edge dummies, the scenario depicted in the left drawing of Figure 3.2c is given. The paths of the three edges incident to each of the `s` and `e` node pairs can simply be joined to form the final path of the backward wrapping edge. At this point certain edges may take superfluous detours, e. g. the edge connecting node `1` and node `4` in the center drawing of Figure 3.2c. While the detour to the right of node `2` is necessary for certain graph instances to prevent edge crossings, it introduces two unnecessary edge bends and increases the edge's overall length in this particular case. However, a simple post-processing of the edge paths, e. g. using one-dimensional compaction techniques [9], allows to improve matters to obtain edge paths as seen in the right drawing of Figure 3.2c.

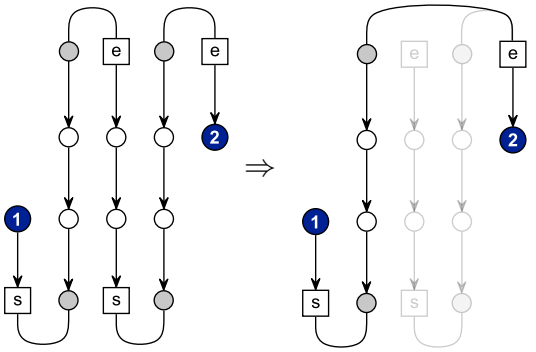


Figure 3.3. Edges spanning multiple cut indices result in back and forth edge paths. One pair of s and e is superfluous and can be removed.

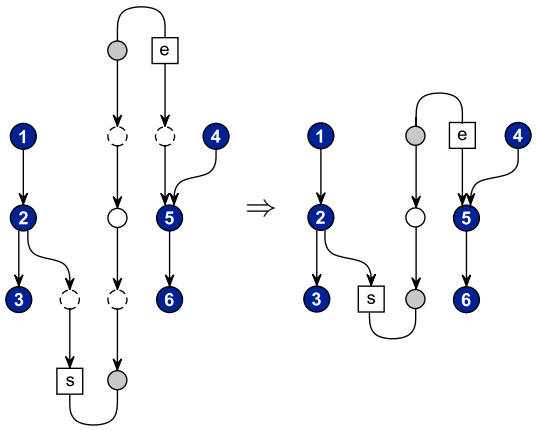


Figure 3.4. The overall edge length can be reduced by replacing superfluous dummy nodes.

3.2.3 Edge Path Improvements

Besides the small edge detour just mentioned, there are two other types of unfortunate edge paths that result from the very rigid procedure discussed above. Two improvement strategies, which have been applied to Figure 1.2, are explained next.

Edges Spanning Multiple Cuts. An edge can span multiple cut indices resulting in a back and forth edge path, as depicted on the left side of Figure 3.3. This stems from the fact that splitting a single edge (u, v) at multiple cut indices results in subsequent pairs of breaking point dummies s and e, which are then connected by long edge dummies. Let d^* denote a chain of zero or more dummy nodes. The sequence of nodes connecting u and v can be written as: $u, d_1^*, \mathsf{s}_1, d_2^*, \mathsf{e}_1, d_3^*, \mathsf{s}_2, \ldots, d_{m-2}^*, \mathsf{s}_n, d_{m-1}^*, \mathsf{e}_n, d_m^*, v$. As illustrated in Figure 3.3, this sequence can be reduced by re-connecting e_n to the last dummy of d_2^* and dropping all dummies in-between: $u, d_1^*, \mathsf{s}_1, d_2^*, \mathsf{e}_n, d_m^*, v$.

As a consequence, every cut edge is routed back to the initial layer immediately after it is cut. The edge then skips a number of chunks horizontally, before it is routed to its target node. While this significantly reduces the length of the edges and avoids back and forth edge routes, which feel very unnatural to a human and are tedious to follow, it can happen that new edge crossings are introduced that are otherwise avoided by the intertwining edge routes.

Overly Long Backward Edges. A cut edge is first routed to the last layer, then back to the very first layer, and finally to the target node. This can result in unnecessarily long edge paths, as depicted on the left side of Figure 3.4. This problem occurs simply due to the very rigid procedure of wrapping back the edges by splitting between the two breaking point dummies and inserting a full chain of long edge dummies. If the source and target node are not located at the beginning and end of a chunk, this automatically

produces overly long edge paths. Both the `s` node and the `e` node are next to a long edge dummy (dashed outline) above and below them, respectively. Consider only the `s` node for the moment. If the pair p of the `s` node and the gray dummy to its right are adjacent within their common layer, and if the same is true for the two neighboring dashed long edge dummies, the two long edge dummies can simply be replaced by p. The effect can be seen on the right side of Figure 3.4. Note that if one iterates the nodes within the layers from right to left, any "nested" `s` nodes are processed first, making room for other `s` nodes that were blocked beforehand. The `e` node in Figure 3.4 can be treated analogously, except that this time it is advantageous to iterate the layer from left to right.

3.2.4 Evaluation

The multi-edge wrapping is evaluated using a subset of 146 graphs of the well-known North (AT&T) graphs, which have a small aspect ratio when drawn from top to bottom [14].[3] The graphs are referred to as *ATTar* in what follows. We assigned a width and height of 30 pixels to the otherwise dimensionless nodes and used a desired separation value of 20. The different configurations of the layout are denoted as follows: no wrapping is performed (No), wrapping is performed but neither the cuts are improved nor the edge length is optimized (Wr), wrapping is performed and cuts are improved (Wri), and finally wrapping is performed and all improvements are executed (Wrii). For all configurations that include wrapping, we set the score function's exponent to 2.0 and used MSD-1 as basis, there was no significant difference when using ARD for the tested graphs.

During the evaluation of the single-edge wrapping we mainly used the max scale value to assess drawing quality. This time, however, cutting many edges can results in very cluttered drawings with an increased edge length. Also, optimizing long chains of breaking point dummy nodes may yield additional edge crossings that can make it more difficult to follow specific edge paths. Therefore, in addition to the max scale measure, the following plots include results for the number of cut edges, the average length per edge, and the average number of crossings per edge. The following questions were to be answered:

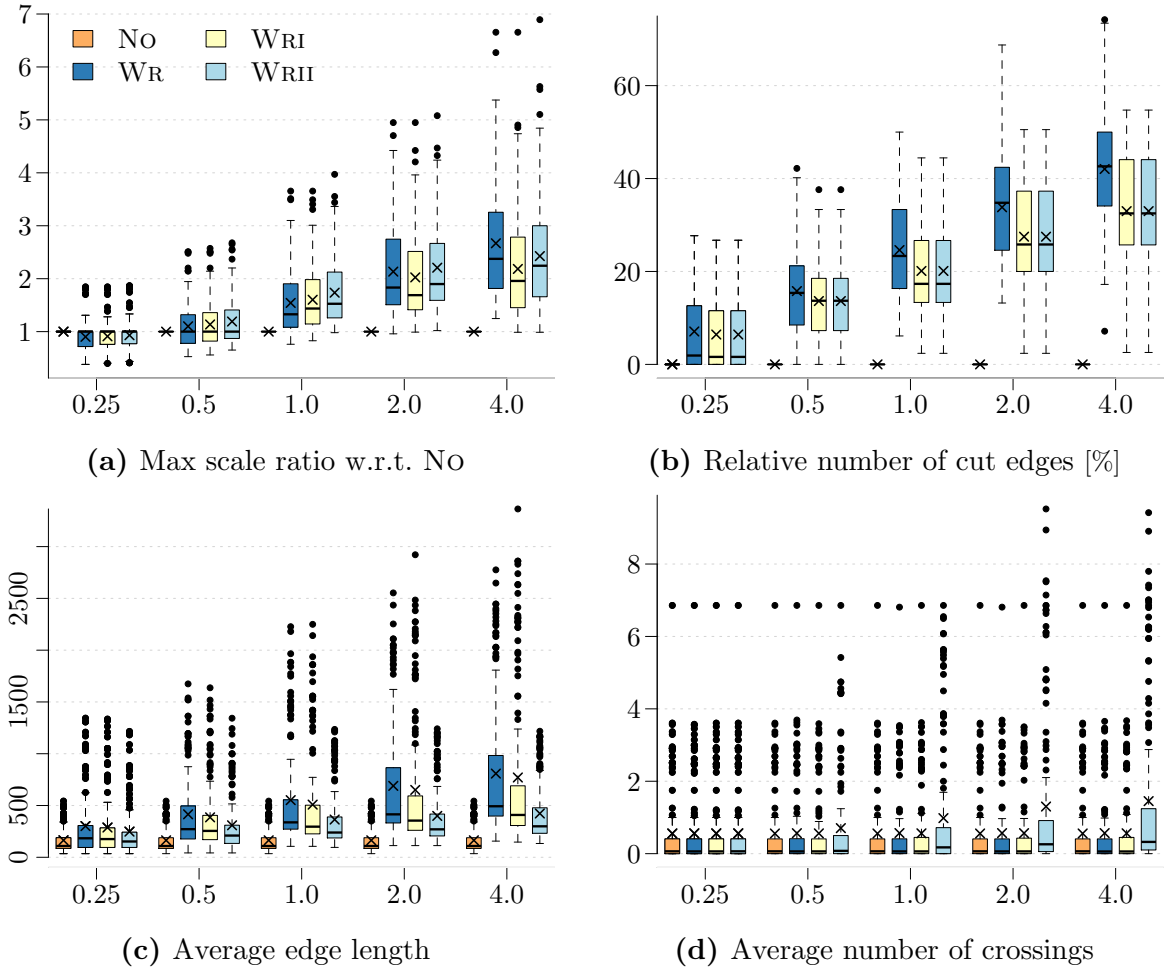
Q1 Does the wrapping improve the max scale measure?

Q2 What is the effect of ECI on max scale and on the number of cut edges? Is the overall edge length reduced by the proposed optimizations?

Q3 How close are the width and height approximations to the width and height of the final drawing?

Figure 3.5 presents the results. When aiming for drawing areas with aspect ratios equal to or above 1.0, executing either of the three wrapping configurations improves

[3] We excluded `g.29.15` from the results since it is significantly denser than the other graphs and worsens the presentation.



(a) Max scale ratio w.r.t. No

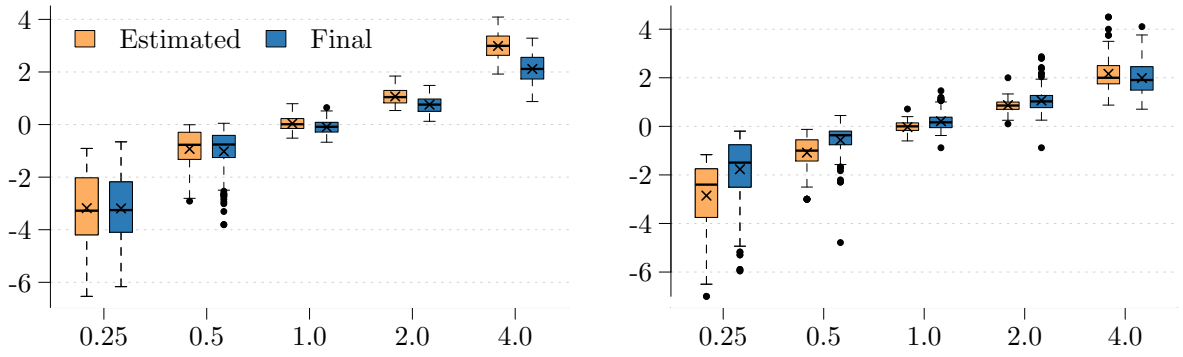
(b) Relative number of cut edges [%]

(c) Average edge length

(d) Average number of crossings

Figure 3.5. Results of wrapping ATTar graphs with varying improvements. Target aspect ratios (x axis) are plotted against the measures stated in the captions (y axis).

the max scale values compared to No (Q1); drawings created with Wrii can on average be scaled more than twice as large as No for a target aspect ratio of 4.0. For aspect ratios smaller than 1.0 no significant improvement can be observed. In fact, for 0.25 No yields the largest max scale values on average; for 72 graphs no wrapping was performed, and for 51 graphs wrapping was performed but resulted in worse max scale values than No. This indicates that one has to be careful when applying the wrapping procedure in ranges where a traditional layering is already close to the prescribed drawing area, which is not surprising since the backward wrapping edges occupy additional space. Moreover, wrapping yields an increased number of cut edges and an increased average edge length, both correlating with the target aspect ratio: the more cuts are required to reach the prescribed drawing area, the more edges must be cut, naturally increasing the total edge length. Figures 3.5b and 3.5c show that the number of cut edges can be decreased using ECI and that the edge length can be improved using the two proposed edge path optimizations (Q2). The increased number of crossings for Wrii, as seen in



(a) MSD-1, for SCGs

(b) MSD-1 and ECI, for ATTar

Figure 3.6. Comparison of estimated and final normalized aspect ratios.

Figure 3.5d, is expected, and the slight variations visible in the outliers of the other three methods can be explained by the heuristic nature of the employed *layer-sweep strategy* to minimize crossings [15].

When using max scale as quality measure, it is not important that the heuristics work with estimates of width and height that are as close as possible to the final pixels of a drawing, but that the relation between the estimates resembles their relation within the final drawing. Thus, to address Q3, we plotted the *normalized aspect ratio* of the estimated dimensions and of the dimensions of the final drawing in Figure 3.6. The normalized aspect ratio is defined for an aspect ratio a as $a - 1$ if $a \geq 1$ and as $1 - 1/a$ otherwise. It can be seen that the estimates are generally close to the final results. It remains to be explored how much the quality of the estimates impacts the results and whether better estimations can actually produce better results. It is likely that better estimations require further knowledge about the remaining layer-based phases to be executed after wrapping: the coordinate assignment phase and the edge routing phase.

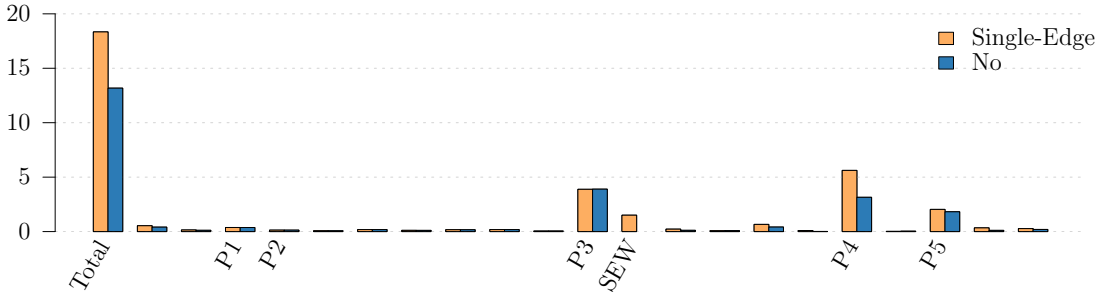
4 Discussion

We presented a methodical process to cut a traditional layering into multiple chunks and draw them side by side with the goal to maximize the scale factor with which a drawing can be displayed in a prescribed area. It works best with sparse and "elongated" graphs and graphs that regularly have "bottlenecks". Our method can easily be explained to and understood by a user. Apart from the cut points, a drawing looks similar to a traditional drawing without any cuts applied. Furthermore, the inherent flow of directed graphs is well-preserved by the wrapping process; the "later" the node is situated within the flow, the "later" the chunk the node ends up in. All of the cut edges that are pointing backwards are routed in well-defined areas of the drawing, in-between pairs of chunks, and can therefore easily be identified and followed by a user.

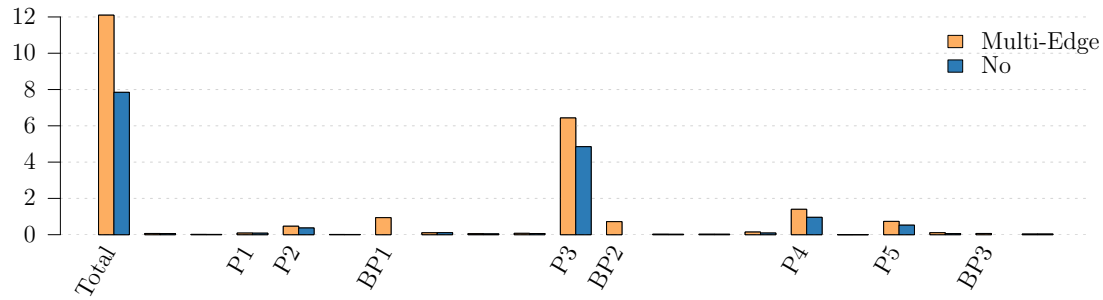
Concerning execution time, which has not been discussed so far, the wrapping process adds a significant number of dummy nodes (`s` and `e`, and long edge dummies) to the graph, of which mainly the long edge dummies created after crossing minimization generate additional work. It thus depends on the complexity of the subsequent coordinate assignment and edge routing phases how much the overall execution time increases. To get a rough idea of the overhead, we executed our two wrapping variants for one graph each on a laptop with an Intel i7 2GHz CPU and 8GB memory: the sequentialized SCG scg_seq.503.00 with 503 nodes and 574 edges, and the North graph g.95.1 with 95 nodes and 132 edges. For each graph, we ran ELK Layered[1] 20 times and plotted the fastest run in Figure 4.1. It can be seen that the overhead keeps within limits: the single-edge execution (SE) took about 18ms instead of 13ms, the multi-edge execution (ME) took about 12ms instead of 8ms. For the SE case, coordinate assignment took 5.6ms instead of 3.1ms. This is as to be expected since the used coordinate assignment algorithm runs in time linear to the number of nodes at that point, which, with increasing numbers of cuts, almost equals twice the number of original nodes. For the ME case, the crossing minimization contributes a significant amount to the total execution time and itself increases from 4.8ms to 6.4ms due to the added `s` and `e` dummies. The lower the execution time of an individual processor was, the larger was the variance between executions. This is also the reason for the slight increase in the execution time of the layer assignment phase during ME, which works on the same graph in either case.

Alternatively, one could compute node coordinates and edge routes for each chunk separately, entirely avoiding the long edge dummies, and stitch the chunks together afterwards. However, this alters the layer-based approach's line of action in a way that may not be realizable in every existing implementation.

[1] Using a variation of Brandes and Köpf's coordinate assignment algorithm [2] and orthogonal edge routes. When wrapping, MSD-1 is used, Wrii for the multi-edge case, and a drawing area with an aspect ratio of 4.0 is targeted.



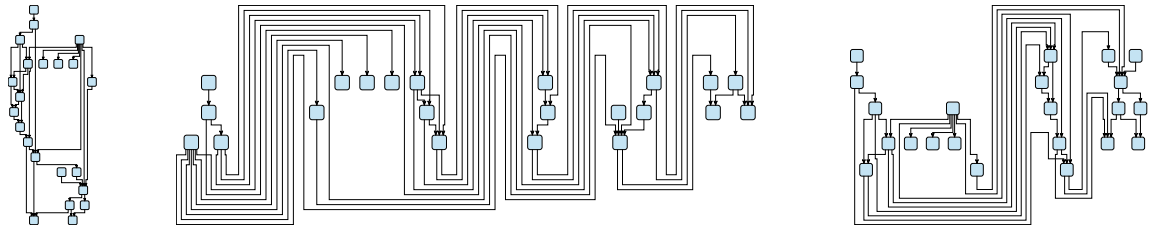
(a) scg_seq.503.00, 11 cuts applied



(b) g.95.1, 8 cuts applied

Figure 4.1. Execution time profiles of ELK Layered for (a) single-edge wrapping and (b) multi-edge wrapping compared to no wrapping (No). The barplots show the execution time in ms (y axis) of each layout processor (x axis) and the combined execution time in the very left pair of bars. Only the interesting layout processors are labeled: P1–P5 are the five phases of the layer-based approach, SEW is the sole processor for single-edge wrapping, and BP1–BP3 are the three processors for multi-edge wrapping. The total number of layout processors in (a) and (b) is different since they are assembled dynamically based on the graph's requirements [15].

As mentioned, our method becomes less effective when many edges have to be cut, e. g. as it is the case for denser graphs. A problematic example can be seen in Figure 4.2. While the cut improvement significantly enhances the appearance of drawing (b), resulting in drawing (c), neither of the two is necessarily a clear improvement over the traditional layering (a). We believe that this can be addressed with alternative presentation styles. To give an example, a set of backward edges in-between a pair of chunks could be *bundled* if it does not involve any edge crossings: the order of the edges leaving at the first chunk is exactly the same order with which the edges enter the second chunk. It is therefore not necessary to follow the edge paths in detail. Likewise, the edge paths could be omitted completely and the connection between the bottom and the top part of the cut edges could be realized using labeling numbers.



(a) No **(b)** Wr **(c)** Wrii

Figure 4.2. Different drawings of ATTar graph g.22.76, targeting a drawing area with an aspect ratio of 4.0 in cases (b) and (c). The drawings are scaled to a common height.

Bibliography

[1] Radoslav Andreev, Patrick Healy, and Nikola S. Nikolov. Applying ant colony optimization metaheuristic to the DAG layering problem. In *Proceedings of the Parallel and Distributed Processing Symposium (IPDPS'07)*, pages 1–9, 2007.

[2] Ulrik Brandes and Boris Köpf. Fast and simple horizontal coordinate assignment. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, volume 2265 of *LNCS*, pages 33–36. Springer, 2002.

[3] Edward G. Coffman. and Ronald L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1(3):200–213, 1972.

[4] Peter Eades and Kozo Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4):424–437, 1990.

[5] Carsten Friedrich and Falk 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.

[6] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A technique for drawing directed graphs. *Software Engineering*, 19(3):214–230, 1993.

[7] Patrick Healy and Nikola S. Nikolov. How to layer a directed acyclic graph. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, volume 2265 of *LNCS*, pages 16–30. Springer, 2002.

[8] Patrick Healy and Nikola S. Nikolov. Hierarchical drawing algorithms. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, pages 409–453. CRC Press, 2013.

[9] Thomas Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[10] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, June 1995.

[11] Lev Nachmanson, George Robertson, and Bongshin Lee. Drawing graphs with GLEE. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Graph Drawing*, volume 4875 of *LNCS*, pages 389–394. Springer Berlin Heidelberg, 2008.

[12] Nikola S. Nikolov, Alexandre Tarassov, and Jürgen Branke. In search for efficient heuristics for minimum-width graph layering with consideration of dummy nodes. *Journal of Experimental Algorithmics*, 10, 2005.

[13] Stephen C. North and Gordon 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.

[14] Ulf Rüegg, Thorsten Ehlers, Miro Spönemann, and Reinhard von Hanxleden. Generalized layerings for arbitrary and fixed drawing areas. *Journal of Graph Algorithms and Applications*, 21(5):823–856, 2017.

[15] Christoph Daniel Schulze, Miro Spönemann, and Reinhard 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.

[16] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, February 1981.

[17] Reinhard von Hanxleden, Björn Duderstadt, Christian Motika, Steven Smyth, Michael Mendler, Joaquín Aguado, Stephen Mercer, and Owen O'Brien. SCCharts: Sequentially Constructive Statecharts for safety-critical applications. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*, Edinburgh, UK, June 2014. ACM.