Preserving Order during Crossing Minimization in Sugiyama Layouts

Sören Domrös[a] and Reinhard Von Hanxleden[b]
Department of Computer Science, Kiel University, Kiel, Germany
{sdo, rvh}@informatik.uni-kiel.de

Keywords: Sugiyama Layout, Layered Drawings, User Intentions, Graph Order.

Abstract: The Sugiyama algorithm, also known as the layered algorithm or hierarchical algorithm, is an established algorithm to produce crossing-minimal drawings of graphs. It does not, however, consider an initial order of the vertices and edges. We show how ordering real vertices, dummy vertices, and edge ports before crossing minimization may preserve the initial order given by the graph without compromising, on average, the quality of the drawing regarding edge crossings. Even for solutions in which the initial graph order produces more crossings than necessary or the vertex and edge order is conflicting, the proposed approach can produce better crossing-minimal drawings than the traditional approach.

1 INTRODUCTION

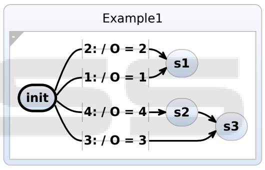
Edge crossings are the most important syntactic aesthetic criterion for node-link diagrams (Purchase, 1997). However, the desire for few edge crossings should not hinder us in synthesizing, automatically and in real-time, a diagram that abides the Nothing is Obviously Non-Optimal (NONO) principle (Kieffer et al., 2016). The SCChart (von Hanxleden et al., 2014) in Figure 1b has no edge crossings, but the drawing is obviously non-optimal when considering the transition order specified in the textual source in Figure 1a. Specifically, the order of edges in the drawing is not consistent with the order of the corresponding transitions in the graph.

In general, graph drawing algorithms consider graphs to consist of unordered sets of vertices and edges. This is also the case for the Sugiyama algorithm (Sugiyama et al., 1981), also known as the layered or hierarchical algorithm, that is used to produce the drawing in Figure 1b. However, in practice we often want to consider some ordering, e. g. the textual order defined in some input file, e. g. in a textual SCChart depicted in Figure 1a. This paper presents an approach to produce drawings where the edges and vertices are ordered in the graph model whenever that is possible without increasing the number of edge crossings, see Figure 1c.

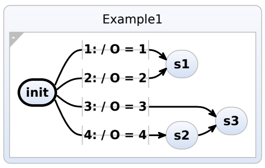
Preserving the textual order in the diagram is part of secondary notation (Petre, 1995) since the visual

```
scchart Example1 {
  output int O
  initial state init // v1
  do O = 1 go to s1 // e11
  do O = 2 go to s1 // e12
  do O = 3 go to s3 // e13
  do O = 4 go to s2 // e14

  state s1 // v2

  state s2 // v3
  go to s3 // e31

  state s3 // v4
}
```

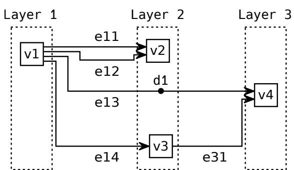
(a) Textual SCChart input file



(b) SCChart synthesized from textual input with Sugiyama layout not considering order



(c) SCChart considering order (this work)



(d) The underlying graph

Figure 1: An SCChart with an underlying ordered graph $G = (V, E)$ with $V = \langle v_1, v_2, v_3, v_4\rangle$ and $E = \langle\langle e_{11}, e_{12}, e_{13}, e_{14}\rangle, \langle\rangle, \langle e_{31}\rangle, \langle\rangle\rangle$.

complies with the semantics. Furthermore, since vertices and edges are ordered as in the graph model, we expect that the layout stability, and with it the preservation of the mental map (Eades et al., 1991; Misue et al., 1995), is improved, since small changes in the

[a] https://orcid.org/0000-0002-8011-8484
[b] https://orcid.org/0000-0001-5691-1215

Domrös, S. and Von Hanxleden, R.
Preserving Order during Crossing Minimization in Sugiyama Layouts.
DOI: 10.5220/0010833800003124
In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022) - Volume 3: IVAPP*, pages 156-163
ISBN: 978-989-758-555-5; ISSN: 2184-4321

graph model do not cause large changes in the drawing.

1.1 Contribution & Outline

This paper presents an approach to introduce the concept of graph order to the Sugiyama algorithm. Specifically, the contributions are the following:

- we define the concept of vertex and edge graph order (Section 3.1);
- we adapt crossing minimization for proper layered *ordered* graphs (Section 3.2);
- we extend the proposed solution to include *partially* ordered graphs (Section 3.3);
- we extend the solution to include backward edges (Section 3.4);
- we propose an order metric that can be used to further improve crossing minimization (Section 3.5).

The resulting algorithm configurations are discussed and evaluated in Section 4. Section 5 presents related work and Section 6 concludes this paper.

An extended version of this paper has appeared as a technical report (Domrös and von Hanxleden, 2021). Specifically, that report presents how to deal with dangling source vertices and discusses various insights in more detail.

2 LAYERED ALGORITHM

We define a graph $G = (V, E)$, *vertices* $V = \{v_1, \ldots, v_n\}$, *edges* $E \subseteq P \times P$, and *ports* P that the edges are anchored at, where $\mathsf{P}(v)$ is the subset of ports that belong to a vertex v. Conversely, $\mathsf{v}(p) \in V$ denotes the vertex that port p is anchored at. For an edge $e = (p, q)$, $\mathsf{p_s}(e) = p \in \mathsf{P}(v)$ describes the *source port* and $\mathsf{p_t}(e) = q \in \mathsf{P}(w)$ describes the *target port*. For simplicity, we also write $e = (v, w)$ for source vertex v and target vertex w as short form of $e = (p, q)$, $\mathsf{v}(p) = v$, and $\mathsf{v}(q) = w$ if we do not care about the ports. For port p, $\mathsf{type}(p) \in \{\mathsf{src}, \mathsf{tgt}\}$ indicates whether p is a source or target port and $\mathsf{e}(p) \in E$ returns the edge of the port p.

The algorithm places vertices in vertical layers, as seen in Figure 1d, and only routes edges between two layers, i. e. *in-layer edges* are forbidden. The algorithm is divided into five phases: cycle breaking, layer assignment, crossing minimization, vertex placement, and edge routing (Sugiyama et al., 1981). The first two phases transform the digraph into a proper layered digraph. In a *layered* graph $G = (V, E, L)$ the set of vertices V is partitioned into m mutually exclusive ordered subsets that represent their layering $L = (L_1, \ldots, L_m)$, with $L_i = \langle v_{L_{i1}}, \ldots, v_{L_{ir}} \rangle$ for a layer of size r. $\mathsf{L}(v) = i$ denotes the layer i of a vertex $v \in V$. A graph is *proper layered* iff for all edges $e = (v, w) \in E$, $\mathsf{L}(w) = \mathsf{L}(v) + 1$ holds. Since this is generally not possible for digraphs, *dummy vertices* and *dummy edges* are added to replace *long edges* that span multiple layers. In Figure 1d, the edge from v_1 to v_4 is a long edge with one dummy vertex d_1 in layer 2. We distinguish between *real vertices* and dummy vertices.

We call vertices with no incoming edges *sources* and vertices with no outgoing edges *sinks*, and define the functions $\mathsf{indegree} : V \to \mathbb{N}$ and $\mathsf{outdegree} : V \to \mathbb{N}$ that return the number of incoming and outgoing edges of a vertex.

Cycle breaking transforms a given graph into an acyclic one. This problem is commonly known as the minimum feedback arc set problem and is NP-hard (Karp, 1972). We call the edges that are reversed in this process *backward edges*. In the following algorithm, they are handled as normal edges. During the edge routing phase, they are reversed to their original direction.

The layer assignment phase creates a proper layered graph by introducing dummy vertices and dummy edges.

Crossing minimization uses the proper layered digraph and orders all vertices in their layers and ports on their vertices such that minimal edge crossings are created, as seen simplified in Algorithm 1.

Algorithm 1: crossingMinimization (original).

```
Input: A proper layered graph G = (V, E, L)
Output: A proper layered ordered graph
r = randomSeed // A fixed random seed
t = 7 // Thoroughness
sweepForward = sweepDirection(r)
bestOrder = null
for i = 0; i < t; i = i + 1 do
    G = randomizeLayers(G, r, sweepForward)
    do
        foreach L_i ∈ L do
            minimizeCrossings(L_i)
    while improved(G);
    if crossings(G) < crossings(bestOrder) then
        bestOrder = G
    sweepForward = ¬sweepForward
return bestOrder
```

Since the crossing minimization problem is NP-hard and remains NP-hard on bipartite graphs (Garey and Johnson, 1983), a heuristic that includes ports (Spönemann et al., 2010) is used. Crossing minimization consists of several runs to prevent local minima bounded by the thoroughness value t. Seeded random values are used to guarantee the same diagram for the same graph. For the first run it is randomly decided whether the layers are traversed beginning with the first or the last layer. We call this the *sweep direction* and distinguish between a forward sweep and a backward sweep. Moreover, the random seed is used to reorder all independent vertices and ports, i. e. all sources or sinks and their ports, via randomizeLayers. In this algorithm, edges are ordered via the ports they are anchored at. Since edges only connect ports in neighboring layers, ordering the ports is enough and edge order is defined by them.

Random permutation of the first or last layer is applied to prevent local minima. If the random initial order yields a local minimum it can be resolved by using a higher thoroughness value or by permuting the first layer. The thoroughness value of 7 proved to be sufficient to prevent local minima even for large graphs in its implementation in the Eclipse Layout Kernel (ELK)[1].

We call the current layer the *fixed layer* and the next layer (in case of a forward sweep L_{i+1}, else L_{i-1}) the *free layer*. At this point we consider the layers ordered and use a crossing minimization strategy, such as the barycenter heuristic (Spönemann et al., 2010), to order the vertices and ports in the free layer and continue to do so with the next layer while sweeping forward and backward until no improvement can be found. For each run the resulting edge crossings are counted efficiently by using the order of their ports, as described by (Barth et al., 2004). The run that yields the smallest number of crossings defines the order of the vertices in each layer and the order of the ports on each vertex.

To evaluate our approach we use the Barycenter method proposed by Sugiyama et al. to minimize the crossings. However, any approach that does not change the vertex order if it is crossing minimal would work here, such as the median heuristic (Eades and Wormald, 1986) or any approach that sweeps through the layers and counts crossings to compare the result.

3 GRAPH ORDER CROSSING MINIMIZATION

As explained earlier, a key difference between the standard layered approach and our proposal is that we consider vertices and edges to be ordered. The next section will formalize this order. This serves as grounding for the subsequent sections, which explain how to produce drawings that aim to reflect that order whenever this is possible without compromising other aesthetic criteria, specifically the number of edge crossings.

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

3.1 Graph Order

Definition 1 (Ordered Graph)**.** *We define an ordered graph as $G = (V,E)$ were $V = \langle v_1,\dots,v_n\rangle$ is the ordered set of vertices and $E = \langle\langle e_{11},\dots,e_{1k_1}\rangle,\dots,\langle e_{n1},\dots,e_{nk_n}\rangle\rangle$ is the ordered set of ordered sets of k_i outgoing edges for each vertex v_i. E implicitly defines an ordered set of outgoing and incoming ports P at which each edge is anchored.*

An example of an ordered graph that follows Definition 1 can be seen in Figure 1. A proper layered ordered graph $G = (V,E,L)$ is defined analogously. We define $\mathsf{o} : V \cup E \cup P \to \mathbb{Z}$ (see Definition 2) as the function that assigns a *graph order value* to vertices, edges, and ports.

Definition 2 (Graph Order o)**.** *$\mathsf{o}(v) = n$ if $v \in V$ is the nth vertex in the graph. Analogously, $\mathsf{o}(e) = n$ if $e \in E$ is the nth edge in the graph and $\mathsf{o}(p) = \mathsf{o}(\mathsf{e}(p))$ for port $p \in P$.*

The graph order specifies orders for vertices and edges, as expressed by o. Ideally, this graph order is also reflected in the drawing of the graph, as is the aim of this work. However, this is not always possible, at least not simultaneously for both vertices and edges, as they may sometimes induce conflicting orderings, as illustrated in the example in Figure 2. Actually one might argue that such cases could and should be avoided, e. g. when writing a textual SCCharts specification, but we still want to be able to handle such cases. We, therefore, distinguish the graph order on vertices and edges from the *drawing order* defined by a vertex order $\prec_v$: $V \times V$ and a port order $\prec_p$: $P \times P$. We also introduce a flag $\mathsf{prioEdgeOrder} = \neg\mathsf{prioVertexOrder}$ to express whether vertex or edge order is prioritized.

Definition 3 (Vertex Order $\prec_v$ for ordered graphs)**.** *For $v,w \in L_i$ for some layer $L_i \in L$, we define $\prec_v$ such that $v \prec_v w$ holds iff one of the following cases applies:*

1. $\mathsf{prioVertexOrder} \wedge \mathsf{o}(v) < \mathsf{o}(w)$. *I. e. vertices are ordered by their graph order. For example in Figure 1, we have $v_2 \prec_v v_3$*
2. $\mathsf{prioEdgeOrder} \wedge \mathsf{p_s}(\mathsf{getFirstEdge}(v)) \prec_p \mathsf{p_s}(\mathsf{getFirstEdge}(w))$ *where* getFirstEdge *returns the edge on the first port of the vertex. The*

(a) Prioritize edge order over vertex order (prioEdgeOrder)

(b) Prioritize vertex order over edge order (prioVertexOrder)

Figure 2: We can either prioritize edge or vertex order. This may yield different drawings depending on the graph order.

first edge is on the incoming port that was deemed smallest by $\prec_p$.
I. e. vertices are ordered by their incoming edges and not by the graph order.

Definition 4 (Port Order $\prec_p$). *For ports a and b attached to the same vertex v,* $\mathsf{v}(a) = v = \mathsf{v}(b)$*, we define* $\prec_p$ *such that* $a \prec_p b$ *holds iff one the following cases applies:*

1. $\mathsf{type}(a) = \mathsf{type}(b) = \mathsf{src} \wedge \mathsf{v}(\mathsf{p_t}(\mathsf{e}(a))) = \mathsf{v}(\mathsf{p_t}(\mathsf{e}(b))) \wedge \mathsf{o}(a) < \mathsf{o}(b)$.
I. e. means outgoing ports that connect to the same target vertex are ordered by the graph order of their edges.
2. $\mathsf{type}(a) = \mathsf{type}(b) = \mathsf{src} \wedge \mathsf{v}(\mathsf{p_t}(\mathsf{e}(a))) = w \neq u = \mathsf{v}(\mathsf{p_t}(\mathsf{e}(b))) \wedge \mathsf{o}(\mathsf{getMinEdge}(v,w)) < \mathsf{o}(\mathsf{getMinEdge}(v,u))$ *where* $\mathsf{getMinEdge} : V \times V \to E$ *returns the edge with the minimum graph order* o *between two vertices.*
I. e. outgoing edges that do not connect to the same target vertex are ordered by the minimal edge order of their target.
E. g. this reduces unnecessary edge crossings in Figure 3 since placing e_{43} *below* e_{42} *would always produce a crossing and bundles edges with the same target.*
3. $\mathsf{type}(a) = \mathsf{type}(b) = \mathsf{tgt} \wedge \mathsf{p_s}(\mathsf{e}(a)) \prec_p \mathsf{p_s}(\mathsf{e}(b))$.
I. e. incoming ports are sorted as the corresponding source port of their edge. This is needed to prevent unnecessary crossings since the source ports are already correctly ordered by $\prec_p$.

3.2 Proper Layered, Ordered Graphs

Our goal is to order all ports and vertices before crossing minimization. We change crossingMinimization in Algorithm 1 such that the *bestOrder* is initialized with the input graph G, the first run starts with a forward sweep, and the second run with a backward sweep, as seen in Algorithm 2. For these first two runs randomizeLayers is not executed since the graph is already ordered.

Algorithm 2: crossingMinimization (new).

Input: A proper layered graph $G = (V, E, L)$
Output: A proper layered ordered graph
r = randomSeed // A fixed random seed
$t = 7$ // Thoroughness
sweepForward = true
foreach $v \in V$ **do**
  sort($\mathsf{P}(v), \prec_p$)
foreach $L_i \in L$ **do**
  sort($L_i, \prec_v$)
bestOrder $= G$
for $i = 0; i < t; i = i + 1$ **do**
  if $i > 1$ **then**
    G = randomizeLayers($G, r, sweepForward$)
  do
    foreach $L_i \in L$ **do**
      minimizeCrossings(L_i)
  while improved(G);
  if crossings(G) < crossings(*bestOrder*) **then**
    bestOrder $= G$
  sweepForward $= \neg$*sweepForward*
return *bestOrder*

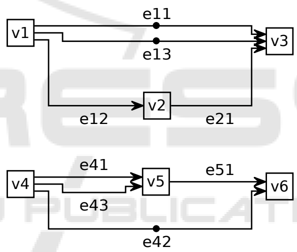


Figure 3: Two graphs with long edges and multiple edges to the same target. Edges with the same target are grouped together to reduce potential edge crossings, (dummy vertices marked as black circles).

3.3 Partially Ordered Graphs

When long edges are introduced, the graph definition changes and dummy vertices and edges are added that have no graph order. A properly layered, partially ordered graph G is defined as the tuple (V', E', L') with $V' = (V, V_D)$ with $V_D = \{d_1, \ldots, d_n\}$ as the set of dummy vertices, $E' = (E_L, E_D)$ with E_L the ordered set of edges in which long edges (v, w) are replaced by shortened long edges (v, d_i) and $E_D = \{e_{d_1}, \ldots, e_{d_n}\}$ the set of dummy edges. As before, P' contains the ports of the corresponding edges of E'. $L' = (L'_1, \ldots, L'_n)$ with $L'_i = (L_i, L_{D_i})$ consists of the ordered part L_i and the set of dummy vertices L_{D_i} in that layer. Dummy vertices and edges are originally not part of the graph and have, therefore, no derived graph order. We extend o such that $\mathsf{o}(e_{d_i}) = \mathsf{o}(e_{kj})$

for a dummy edge e_{d_i} if e_{kj} is the original long edge the dummy edge was created for. Note that a dummy vertex has no defined graph order value.

We have to change cases 1 and 2 of Definition 4 to also handle long edges. Instead of comparing the target vertex, the *long edge target vertex*, which corresponds to the real vertex the edge eventually connects to, of each port is compared.

Definition 3 also has to be changed. The condition in case 1 changes to $(\mathsf{prioEdgeOrder} \vee c \in V_D \vee u \in V_D) \wedge \mathsf{v}(\mathsf{p_s}(\mathsf{getFirstEdge}(\mathsf{v}))) \prec_v \mathsf{v}(\mathsf{p_s}(\mathsf{getFirstEdge}(w)))$. Dummy vertices are compared to other vertices using the incoming edges.

3.4 Backward Edges

Backward edges are in most cases already handled by the algorithm. For a consistent drawing style, we want to place backward edges below normal ones and change Definition 4 case 1 such that this is the case, and change getMinEdge in case 2 such that the graph order of backward edges is not considered here since they originate from a different vertex.

3.5 A Graph Order Metric

The defined relations $\prec_v$ and $\prec_p$ serve as a metric to decide how good a graph is ordered. This metric can be used as a secondary criterion during crossing minimization. To do this, line 16 in Algorithm 2 is changed by adding the port or vertex order violations multiplied by a weight w_v and w_p, so that the condition becomes:

$$\begin{aligned} & w_p \cdot \mathsf{portOrderViolations}(G) \\ & + w_v \cdot \mathsf{vertexOrderViolations}(G) + \mathsf{crossings}(G) \\ < \; & w_p \cdot \mathsf{portOrderViolations}(bestOrder) \\ & + w_v \cdot \mathsf{vertexOrderViolations}(bestOrder) \\ & + \mathsf{crossings}(bestOrder) \end{aligned}$$

where portOrderViolations and vertexOrderViolations count the number of port and vertex order violations for a partially ordered graph. These weights express how many order violations are as important as an edge crossing. E. g. $w_v = w_p = 0.1$ means that 10 vertex or port order violations are as important as an edge crossing.

4 EVALUATION

We compare nine different algorithm configurations, as seen in Table 1. Note that N_V and N_E produce the same graph but are differently evaluated regarding their violations of the graph order, as described in Section 4.2.

Table 1: N (unordered), V (prioVertexOrder), E (prioEdgeOrder). Overview and encoding of the evaluated algorithmic alternatives. Columns differ in whether vertices or edges are prioritized, rows differ in the weights assigned to vertex/port order violations relative to edge crossings, which carry a weight of 1.

w_p	w_v	N	V	E
0	0	$\mathrm{N}_V, \mathrm{N}_E$	V	E
>0	0		$\mathrm{V}_{w_p,0}$	$\mathrm{E}_{w_p,0}$
0	>0		V_{0,w_v}	V_{0,w_v}
>0	>0		V_{w_p,w_v}	E_{w_p,w_v}

We consider 54 SCCharts that were developed by humans. SCCharts models may consist of concurrent more than one concurrent region with states and edges between them. Each region has its own graph. For each model we, therefore, might solve several graph drawing problems. The chosen models have two to 72 vertices per region and up to 310 vertices per model with an average of 44 vertices per model (including dummy vertices). There are from one to 16 vertices per layer. The edge density to adjacent layers is two to 73 with an average of 9. The average vertex degree is between zero and seven.

For all graphs the ordering step is done in a fraction of a millisecond and is significantly quicker and less complex than crossing minimization in general. The layout direction is set the RIGHT and the dummy vertices are sorted above normal vertices (dummyVerticesAbove). How a value for w_v and w_p is chosen is described in the following.

4.1 Weighted Ordering with w_v and w_p

Table 2 illustrates the effects of varying w_v and w_p on edge crossings and on the number of fully ordered drawings of the 54 graphs. Increasing the weight of $\prec_v$ and $\prec_p$ during crossing minimization tends to increase the number of correctly ordered graphs at the cost of edge crossings, but it cannot always find an ordering with minimal order violations. The reasons for this is that the barycenter heuristic (or any other commonly used approach) used during crossing minimization does not focus on the order but on crossing minimization. If no run yields the ordered solution, it cannot be chosen, even though it would be chosen if it occurred, based on the weights w_v and w_p.

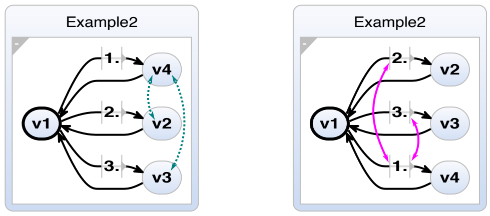
4.2 Quantitative Evaluation

The number of drawing order violations for the different approaches can be seen in Table 2. N_V serves

Table 2: Graph order violations for the metrics $\prec_v$ and $\prec_p$ for w_v and w_p set to 0.001, 0.01, 0.1, 0.5, 1, 10, and 100 for their respective approaches. Lines for $\mathrm{V}_{w_p,0}$ and $\mathrm{E}_{w_p,0}$ are omitted since they did not change with varying w_p. *Fully ordered drawings* describes the number of models that have no order violations in any part of their model. *Crossings* describes the total number of edge crossings in all regions of all 54 models with the corresponding algorithm.

	$\prec_v$	$\prec_p$	Fully ordered drawings	Crossings
N_V	250	695	1	26
V	143	91	23	32
$\mathrm{V}_{0.001,0}$	143	91	23	32
...	...	...	...	...
$\mathrm{V}_{100,0}$	143	91	23	32
$\mathrm{V}_{0,0.001}$	91	173	24	28
$\mathrm{V}_{0,0.01}$	91	173	24	28
$\mathrm{V}_{0,0.1}$	91	173	24	28
$\mathrm{V}_{0,0.5}$	44	172	24	28
$\mathrm{V}_{0,1}$	81	184	25	39
$\mathrm{V}_{0,10}$	64	159	26	82
$\mathrm{V}_{0,100}$	72	169	26	112
$\mathrm{V}_{0.001,0.001}$	122	96	24	28
$\mathrm{V}_{0.01,0.01}$	122	96	24	28
$\mathrm{V}_{0.1,0.1}$	122	96	24	28
$\mathrm{V}_{0.5,0.5}$	95	78	26	36
$\mathrm{V}_{1,1}$	102	48	29	93
$\mathrm{V}_{10,10}$	129	12	29	149
$\mathrm{V}_{100,100}$	129	12	29	149
N_E	45	695	1	26
E	27	91	31	32
$\mathrm{E}_{0.001,0}$	27	91	31	32
...	...	...	...	...
$\mathrm{E}_{100,0}$	27	91	31	32
$\mathrm{E}_{0,0.001}$	14	93	32	28
$\mathrm{E}_{0,0.01}$	14	93	32	28
$\mathrm{E}_{0,0.1}$	14	93	32	28
$\mathrm{E}_{0,0.5}$	10	88	32	28
$\mathrm{E}_{0,1}$	10	101	33	32
$\mathrm{E}_{0,10}$	12	101	33	36
$\mathrm{E}_{0,100}$	10	95	33	34
$\mathrm{E}_{0.001,0.001}$	14	92	32	28
$\mathrm{E}_{0.01,0.01}$	14	92	32	28
$\mathrm{E}_{0.1,0.1}$	14	92	32	28
$\mathrm{E}_{0.5,0.5}$	12	78	33	34
$\mathrm{E}_{1,1}$	13	52	39	46
$\mathrm{E}_{10,10}$	32	20	39	92
$\mathrm{E}_{100,100}$	32	20	39	92

as a baseline for all approaches that have prioVertexOrder set (i. e. V and V_{w_p,w_v}), N_E serves as a baseline for approaches with prioEdgeOrder (i. e. E, E_{w_p,w_v}). The resulting drawing of N_V and N_E is the same, we only count the $\prec_v$ violations by comparing the graph order of real vertices in the N_V case and use the edge graph order for N_E. All approaches that prioritize edge order have, therefore, fewer vertex order violations.



(a) A drawing with two $\prec_v$ violations for a prioVertexOrder approach and none with prioEdgeOrder.

(b) A drawing with two $\prec_p$ violations.

Figure 4: Vertex order violations are shown in teal and edge order violations in magenta.

Figure 4 visualizes how $\prec_v$ and $\prec_p$ order violations are counted for the different approaches. In Figure 4a, the shown vertex order violations are only counted for all prioVertexOrder approaches since prioEdgeOrder approaches order vertices by their incoming edges. In Figure 4b, we see two edge order violations. Furthermore, note that the V approach would not produce this drawing but the $\mathrm{V}_{0,0.001}$ approach would. The V would initially order the ports and vertices without creating violations (e. g. edge 1 above edge 2 and 3). If crossing minimization starts, the vertices v_2 to v_4 are in a free layer and their order is changed to comply with the port order to not produce additional crossings. For the $\mathrm{V}_{0,0.001}$ approach this creates two violations. The second run would then yield the drawing in Figure 4b since v_2 to v_4 are in the fixed layer for a backward sweep. The resulting drawing has no vertex order violations and no crossings and is, therefore, better than Figure 4a under the $\mathrm{V}_{0,0.001}$ approach.

The two approaches V_{w_p,w_v} and E_{w_p,w_v} that use the $\prec_v$ and $\prec_p$ metrics as a secondary criterion to edge crossings have the fewest total number of order violations for their respective approaches, as seen in Table 2.

4.3 Qualitative Evaluation

The V approach is evaluated compared to the normal approach. The results can be seen in Figure 5. Of these 54 models, 31 were consistent and 23 were conflicting. 20 of the consistent models improved the order and not just placed backward edges below normal ones. This can also be an improvement if it introduces a consistent drawing style but we chose to distinguish it from a real drawing order improvement. Three of the models that had conflicts did not change in the conflicting regions. 20 of the models that had conflicts nonetheless improved the overall layout. Of

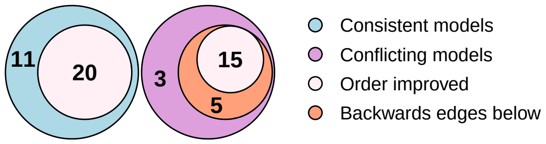


Figure 5: Changes of the 54 evaluated models.

these 20 models, 15 had order improvements, the rest just placed backward edges below normal edges. All three conflicting models that did not improve, were, by coincidence, ordered as good as possible.

4.4 Influence of Randomization

Table 2 shows that the N (unordered) approach has fewer total crossings (26) than all other (ordered) approaches, even though we argue that ordering vertices and ports before crossing minimization should not necessarily increase the number of crossings relative to the traditional approach. As it turns out, four of the 54 models have a different number of edge crossings than the N approach. One has one crossing less (from one to zero crossings), two have one crossing more (from zero to one crossing), and another one has either one or three additional crossings depending on the approach (from 15 to 16 or 18 crossings) in some regions of some models. The reason for this is randomization.

Randomization still has an influence on the quality of the solution for some kind of graphs. If a graph is conflicting or the drawing order produces additional crossings, randomizeLayers is still used to create the drawing, as seen in Algorithm 2. Therefore, these graphs can by coincidence produce more or fewer edge crossings or order violations.

In the ordered approach the first two runs do not randomize the order and use the graph order for a forward and backward sweep. The order after these two sweeps is not the same order the unordered approach uses for their first run. Therefore, the vertex and port order is already different to the N approach before randomization takes place. Since any kind randomization can lead to different results, some of these runs can randomly be a local minimum or no longer be a local minimum, and result in more or fewer crossings or order violations. Only increasing the thoroughness reduces the probability of additional crossings but does not solve this anomaly.

4.5 Evaluation

If one favors the edge order, E or E_{w_p,w_v} are the recommended approaches. For SCCharts E_{w_p,w_v} is recommended to still get better results if E falls back to a random starting permutation. If one expects the user to change the vertex order in the model file to create the desired layout, V_{0,w_v} should be used since it can be used to enforce vertex positions.

The proposed approach performs especially good for graphs that a tree-like with a final vertex were everything connects to (which might have a feedback loop) or several feedback loops to the root vertex or another central vertex. Without edge graph order, the different routes trough the tree are not ordered by their priority (e. g. the edge with priority 1 is on top, the edge with priority 2 is below, ...), as the secondary notation suggests, but randomly, which highly irritates the user and impedes understandability since "true" and "false" cases might change sides.

5 RELATED WORK

There were already several works that aim to identify or produce a human-like layout and operate at the so called NONO principle (Nothing is obviously non-optimal) (Kieffer et al., 2016). Kieffer et al. use this principle and human participants to identify aesthetic criterions and goals for a human-like orthogonal layout algorithm. Purchase et al. (Purchase et al., 2020) take this further and try to identify layouts that are obviously machine made. We tackle similar goals. Figure 1b looks obviously machine made and is a drawing a human would not produce. In contrast to Kieffer et al., we choose to still use the layered algorithm and try to conform with the NONO principle by using the graph order as additional layout information to solve obvious problems instead of developing a whole new algorithm.

The are several extensions to the Sugiyama algorithm. We will discuss some of them that aim to influence the order of vertices and edges.

(Waddle, 2001) uses constraints to order vertices and to force them on specific positions even if this causes additional crossings. We try to preserve the initial order but still try to minimize crossings if possible. We focus on layout *creation*. Waddle, however, focuses on layout *adjustment* and prioritizes order constraints over crossings. Our ordering does not constrain the solution, but rather creates a better crossing-optimal solution. They use layout adjustment to maintain the mental map, we assume that the graph order in the model file is a representation of the modelers mental map.

(Mennens et al., 2019) aim to produce stable drawings by maintaining a global order of vertices. This global order constrains vertex movement in the same layer. Again, this order will not be changed if additional crossings are produced. Moreover, they do not consider the ordering of edges, since there is at most one edge from one vertex to another vertex.

(Böhringer and Paulisch, 1990) introduce absolute and relative constraints to fix the order of vertices. This is a viable solution to maintain graph order but this, again, does not prevent additional crossings. We want to automatically produce drawings that maintain the graph order without causing additional crossings. Again, Böringer and Paulisch do not constrain edges but only vertices, which solves only one of our problems since no dummy vertices can be constrained.

6 CONCLUSION

We presented a solution to preserve the graph order by setting an initially best ordering for crossing minimization. This allows us to maintain the graph order without causing additional crossings introduced by local minima other than through coincidence or order constraints for many models.

Including the proposed graph order metric in the crossing minimization step additionally to the crossings as a secondary criterion seems beneficial. Therefore, prioEdgeOrder with weighted vertices and ports (E_{w_p,w_v}) is one potential option for SCCharts. Another one is V since it allows to control the layout without changing the semantic by changing the vertex graph order. Therefore, we make this setting configurable for SCCharts and to evaluate this further.

Future work on this project should evaluate whether SCCharts that are created in a tool that visualizes the diagram taking the graph order into account results in more consistent models or otherwise changes the way modelers design.

REFERENCES

Barth, W., Mutzel, P., and Jünger, M. (2004). Simple and efficient bilayer cross counting. *Journal of Graph Algorithms and Applications*, 8(2):179–194.

Böhringer, K.-F. and Paulisch, F. N. (1990). Using constraints to achieve stability in automatic graph layout algorithms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 43–51, New York. ACM.

Domrös, S. and von Hanxleden, R. (2021). Preserving order during crossing minimization in sugiyama layouts. Technical Report 2103, Christian-Albrechts-Universität zu Kiel, Department of Computer Science. ISSN 2192-6247.

Eades, P., Lai, W., Misue, K., and Sugiyama, K. (1991). Preserving the mental map of a diagram. In *Proceedings of the First International Conference on Computational Graphics and Visualization Techniques*, pages 34–43.

Eades, P. and Wormald, N. C. (1986). The median heuristic for drawing 2-layered networks. Technical Report 69, University of Queensland, Department of Computer Science.

Garey, M. R. and Johnson, D. S. (1983). Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):312–316.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations (Proceedings of a Symposium on the Complexity of Computer Computations, March, 1972, Yorktown Heights, NY)*, pages 85–103. Plenum Press, New York.

Kieffer, S., Dwyer, T., Marriott, K., and Wybrow, M. (2016). HOLA: human-like orthogonal network layout. *IEEE Trans. Vis. Comput. Graph.*, 22(1):349–358.

Mennens, R. J., Scheepens, R., and Westenberg, M. A. (2019). A stable graph layout algorithm for processes. In *Computer Graphics Forum*, volume 38, pages 725–737. Wiley Online Library.

Misue, K., Eades, P., Lai, W., and Sugiyama, K. (1995). Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210.

Petre, M. (1995). Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM*, 38(6):33–44.

Purchase, H. C. (1997). Which aesthetic has the greatest effect on human understanding? In *Proceedings of the 5th International Symposium on Graph Drawing (GD '97)*, volume 1353 of *LNCS*, pages 248–261. Springer.

Purchase, H. C., Archambault, D., Kobourov, S., Nöllenburg, M., Pupyrev, S., and Wu, H.-Y. (2020). The turing test for graph drawing algorithms. In *International Symposium on Graph Drawing and Network Visualization*, pages 466–481. Springer.

Spönemann, M., Fuhrmann, H., von Hanxleden, R., and Mutzel, P. (2010). Port constraints in hierarchical layout of data flow diagrams. In *Proceedings of the 17th International Symposium on Graph Drawing (GD '09)*, volume 5849 of *LNCS*, pages 135–146. Springer.

Sugiyama, K., Tagawa, S., and Toda, M. (1981). Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125.

von Hanxleden, R., Duderstadt, B., Motika, C., Smyth, S., Mendler, M., Aguado, J., Mercer, S., and O'Brien, O. (2014). SCCharts: Sequentially Constructive Statecharts for safety-critical applications. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*, pages 372–383, Edinburgh, UK. ACM.

Waddle, V. (2001). Graph layout for displaying data structures. In *Proceedings of the 8th International Symposium on Graph Drawing (GD '00)*, volume 1984 of *LNCS*, pages 98–103. Springer.