Layout of Compound Directed Graphs

Technical Report A/03/96

Georg Sander
(`sander@cs.uni-sb.de`)

Universität des Saarlandes,
FB 14 Informatik,
66041 Saarbrücken

June 5, 1996

Abstract: We present a method for the layout of compound directed graphs that is based on the hierarchical layer layout method. Our method has similarities with the method of Sugiyama and Misue [7] but gives different results: It uses a global partitioning into layers and tries to produce placements of nodes such that border rectangles can be drawn around the nodes of each subgraph. The method is implemented in the VCG tool.

1 Introduction

Compound graphs are graphs where nodes are grouped into nested subgraphs. Compound graphs are useful in many situations. For instance, in interprocedural control flow diagrams, the nodes of each procedure should be grouped together, and in circuit diagrams, the elements of each module should be surrounded by rectangles indicating its borders.

There are still not many layout algorithms for compound graphs. Most algorithms recursively deal with subgraphs as large nodes [4, 3] but ignore the global connectivity: Each subgraph is laid out independently which may result in too much edge crossings, if there are edges going beyond the borders of subgraphs. Other methods are restricted to planar graphs [2].

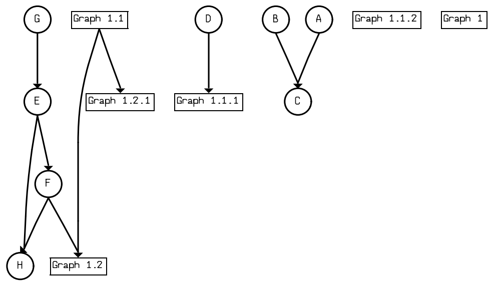
We present a layout method for general directed compound graphs that is based on layout in layers [8, 1]. A similar method was presented by Sugiyama and Misue [7]. There, a *proper compound digraph* is constructed where each subgraph consists of local layers. Different to that, we use a global partitioning into layers and try to produce placements of the nodes within the layers such that border rectangles can be drawn around the nodes of each subgraph afterwards. Thus, we consider our method rather as an alternative to [7] than as an extension of [7].

2 General Notation

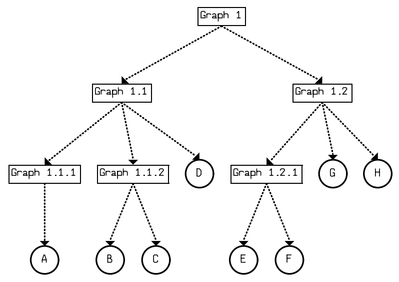
A *simple directed graph* $G = (V, E)$ consists of the set of nodes V and the set of edges $E \subseteq V \times V$. The set $\mathbf{pred}(v) = \{w \in V | (w, v) \in E\}$ contains the *direct predecessors*, the set $\mathbf{succ}(v) = \{w \in V | (v, w) \in E\}$ contains the *direct successors* of a node v. We write $v \xrightarrow[G]{} w$ for an edge $(v, w) \in E$ and $v \xrightarrow[G]{}^* w$ for a (potentially empty) sequence of edges $v \xrightarrow[G]{} v_1 \xrightarrow[G]{} \ldots \xrightarrow[G]{} v_n \xrightarrow[G]{} w$ (a *path*). A *cycle* is a nonempty path $v \xrightarrow[G]{}^* v$. If a simple graph does not contain cycles, we call it *acyclic* or *DAG*. A *tree* is a DAG $T = (V, E)$ consisting of n nodes and $n - 1$ edges which has a special root node $r \in V$ with $r \xrightarrow[T]{}^* v$ for each $v \in V$. The *leaves* v of a tree have the property $\mathbf{succ}(v) = \emptyset$. All other nodes except the leaves are called *inner nodes* of the tree.

A *compound graph* $G = (G', T')$ consists of a simple directed graph $G' = (B \cup S, E_G)$ and a tree $T' = (B \cup S, E_T)$ (Fig. 1). The set B contains the leaves of T' which are called *base nodes*, and the set S contains the inner nodes of T' which are called *subgraphs.* In a compound graph, G' represents a connectivity relation between the base nodes and subgraphs. T' represents a nesting relation: subgraphs may contain other subgraphs or base nodes. We say v *belongs to* $u \in S$ iff $u \xrightarrow[T]{}^* v$.

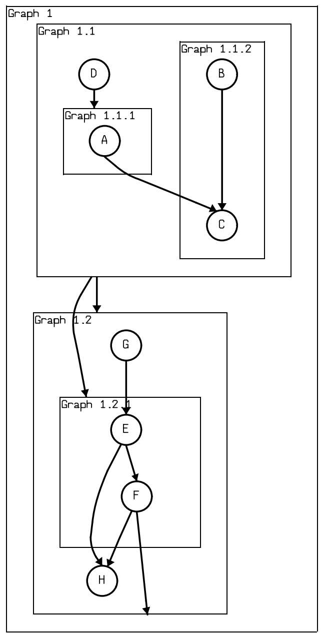
Remark: a compound graph is not recursively defined as a graph whose nodes might be graphs whose nodes might be graphs etc. There, we would only allow connectivity edges between nodes at the same nesting level. The notion *compound graph* is more general because it may contain connectivity edges that cross the borders of nested subgraphs.



Connectivity relation G'



Nesting relation T'



Layout of the compound graph

Figure 1: Compound Graph

3 Layout Conventions

The aim is to visualize the connectivity relation by arrows and the nesting relation by nested border rectangles. These show which nodes and subgraphs belong to the same subgraph. We use the following layout conventions: (a) Base nodes are partitioned into layers. All nodes of the same layer have (nearly) the same y-coordinate. (b) Base nodes do not overlap edges or other base nodes. (c) Connectivity edges may have bend points or may cross each other, but we try to avoid bend points and crossings when possible. (d) Connectivity edges should be uniformly oriented when possible (e.g., top down). If the graph is cyclic, it is not always possible. (e) A border rectangle of a subgraph u contains exactly the base nodes and the rectangles of subgraphs that belong to u. (f) The border rectangles of two nonnested subgraphs do not overlap. (g) Connectivity edges may cross border lines, but we try to avoid such crossings when possible.

The layout algorithm to satisfy these conventions is a variant of the hierarchical method of layout in layers [8, 1]. First, we partition the base nodes into layers. Here we must consider conventions (a), (d) and (g). Connectivity edges crossing several layers are split into sequences of dummy nodes and short edges. The result is a *proper hierarchy* where all connectivity edges are between adjacent layers. Next, we reorder the base nodes and dummy nodes within the layers such that the result has few crossings. Here we must consider conventions (c), (e) and (f). Finally, we calculate the absolute coordinates of the base nodes, dummy nodes and the border rectangles (considering conventions (b), (e) and (f)) and draw the edges as line sequences between the dummy nodes.

4 Partitioning into Layers

Layers are numbered top down. The uppermost layer has the number 1. The task is to calculate ranks $\mathbf{R}(v) \in \mathbb{N}$ for all base nodes indicating the layer v belongs to. Connectivity edges should preferably point top down. A subgraph u has an upper border with rank $\mathbf{R}_{min}(u)$ and a lower border with rank $\mathbf{R}_{max}(u)$. The nesting conventions imply the following rule:

- If a bordering rectangle of a subgraph u is above (or below, resp.) another base node or subgraph w then all nodes v belonging to u are automatically above (or below, resp.) w.

This influences the partitioning. Example: Let $G = (G', T')$ be a compound graph, let two subgraphs u_1 and u_2 be nonnested, let the base node v_1 belong to u_1 and v_2 belong to u_2. If two edges $(u_1, u_2) \in E_G$ and $(v_2, v_1) \in E_G$ exist, then one of these must be reverted, i.e., it points converse to the uniform edge orientation, although $G' = (B \cup S, E_G)$ does not contain a cycle. The combination $(B \cup S, E_G \cup E_T)$ contains the cycle. On the other hand, if $(v_1, u_1) \in E_G$ exist, then the combination $(B \cup S, E_G \cup E_T)$ contains the cycle $v_1 \underset{G'}{\longmapsto} u_1 \underset{T'}{\longmapsto} v_1$ but the edge (v_1, u_1) need not to be reverted. It may point to the lower border of the rectangle of u_1.

Definition 1 *A* **legal rank assignment** *consists of numberings* $\mathbf{R} : B \to \mathbb{N}$*,* $\mathbf{R}_{min} : S \to \mathbb{N}$*, and* $\mathbf{R}_{max} : S \to \mathbb{N}$*, where*
(1) $\mathbf{R}_{min}(u) < \mathbf{R}_{max}(u)$ *for all* $u \in S$*.*
(2) $\mathbf{R}_{min}(u) < \mathbf{R}(v) < \mathbf{R}_{max}(u)$ *for all* $u \in S,\ v \in B$ *with* $u \underset{T'}{\longmapsto} v$*.*
(3) $\mathbf{R}_{min}(u_1) < \mathbf{R}_{min}(u_2) < \mathbf{R}_{max}(u_2) < \mathbf{R}_{max}(u_1)$ *for all* $u_1, u_2 \in S$ *with* $u_1 \xrightarrow[T']{} u_2$*.*

To calculate the positions of the upper and lower borders of rectangles, we simply introduce two dummy nodes $u^{(-)}$ and $u^{(+)}$ for each subgraph u and assign ranks to them. We call them *border nodes.* Then, we define $\mathbf{R}_{min}(u) = \mathbf{R}(u^{(-)})$ and $\mathbf{R}_{max}(u) = \mathbf{R}(u^{(+)})$, i.e., the upper (or lower, resp.) border of the rectangle is drawn in the layer that contains $u^{(-)}$ (or $u^{(+)}$, resp.). Since the borders of rectangles are usually much thinner than base nodes, we do not mix base nodes and border nodes within the layers. Layers either contain only border nodes or only base nodes. Between two layers of base nodes, there may be at most $2k$ layers with border nodes where k is the depth of the tree T'. Thus, we assign multiples of $2k+1$ as ranks to the base nodes while all other layers are reserved for border nodes (Fig. 2, left). After the partitioning, empty layers are removed.

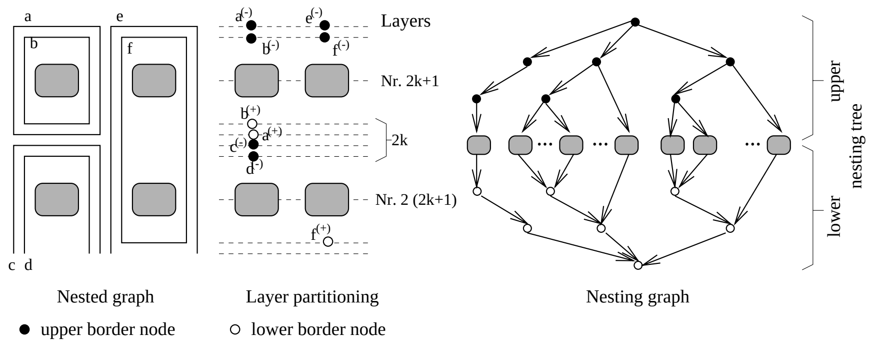


Figure 2: Layer Assignment and Nesting Graph

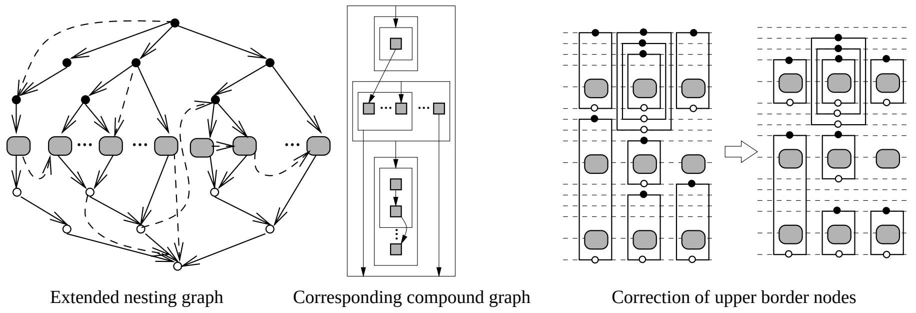

Extended nesting graph
Corresponding compound graph
Correction of upper border nodes

Figure 3: Extended Nesting Graph and Border Correction

Definition 2 *The* **nesting graph** *of a compound graph* (G', T') *consists of*

(1) the set of nodes $B \cup \{u^{(-)} \mid u \in S\} \cup \{u^{(+)} \mid u \in S\}$,

(2) an edge $(u^{(-)}, v)$ *for each* $(u, v) \in E_T$ *with* $u \in S, v \in B$,

(3) an edge $(u_1^{(-)}, u_2^{(-)})$ *for each* $(u_1, u_2) \in E_T$ *with* $u_1, u_2 \in S$,

(4) an edge $(v, u^{(+)})$ *for each* $(u, v) \in E_T$ *with* $u \in S, v \in B$,

(5) an edge $(u_2^{(+)}, u_1^{(+)})$ *for each* $(u_1, u_2) \in E_T$ *with* $u_1, u_2 \in S$.

We call the edges of the nesting graph **nesting edges**.

The nesting graph consists conceptually of two copies of the nesting tree T' that are fused at the leaves. In the upper copy, the nodes $u^{(-)}$ are used. In the lower copy, the nodes $u^{(+)}$ are used and the edges are reverted such that all paths point to the copy $r^{(+)}$ of the root $r \in T'$ (Fig. 2, right). The nesting graph is acyclic.

Theorem 1 *If we traverse the nesting graph in topological order and calculate* $\mathbf{R}(v) = \max\{\mathbf{R}(w) \mid w \in \mathbf{pred}(v)\} + 1$, *then this produces a legal rank assignment.*

This even holds, if we add edges to the nesting graph but keep it acyclic. Thus, we successively add the following edges of E_G to the nesting graph if they do not produce cycles (Fig. 3, left):

(1) (v, w) for each $(v, w) \in E_G$ with $v, w \in B$,
(2) $(u^{(-)}, v)$ for each $(u, v) \in E_G$ with $u \in S, v \in B$ and $u \xrightarrow[T']{}{}^* v$,
(3) $(u^{(+)}, v)$ for each $(u, v) \in E_G$ with $u \in S, v \in B$ and $\nexists u \xrightarrow[T']{}{}^* v$,
(4) $(v, u^{(+)})$ for each $(v, u) \in E_G$ with $u \in S, v \in B$ and $u \xrightarrow[T']{}{}^* v$,
(5) $(v, u^{(-)})$ for each $(v, u) \in E_G$ with $u \in S, v \in B$ and $\nexists u \xrightarrow[T']{}{}^* v$,
(6) $(u_1^{(-)}, u_2^{(-)})$ for each $(u_1, u_2) \in E_G$ with $u_1, u_2 \in S$ and $u_1 \xrightarrow[T']{}{}^* u_2$,
(7) $(u_1^{(+)}, u_2^{(+)})$ for each $(u_1, u_2) \in E_G$ with $u_1, u_2 \in S$ and $u_2 \xrightarrow[T']{}{}^* u_1$,
(8) $(u_1^{(+)}, u_2^{(-)})$ for each $(u_1, u_2) \in E_G$ with $u_1, u_2 \in S$ and $\nexists u_1 \xrightarrow[T']{}{}^* u_2$ and $\nexists u_2 \xrightarrow[T']{}{}^* u_1$.

After adding these edges a topological sorting as mentioned above gives the rank assignment. This rank assignment tends to select the smallest possible ranks. This is unfavorable for the upper border nodes because the upper border of a subgraph u should be positioned near to the base nodes of u. Thus, $\mathbf{R}_{min}(u)$ should be as large as possible. Thus, the extended nesting graph is finally traversed in converse topological order and

$$\mathbf{R}(u^{(-)}) = \min\{\mathbf{R}(w) \mid w \in \mathbf{succ}(u^{(-)})\} - 1$$

is calculated to correct this situation (Fig 3, right). Because $\mathbf{succ}(u^{(-)})$ cannot be empty in the nesting graph, the formula is well defined.

5 Production of Dummy Nodes

If we delete the nesting edges from the extended nesting graph, the result is a layer hierarchy that identifies for all edges between which layers they must be routed. Next, we must insert the edges $e \in E_G$ that could not be added to the nesting graph because they had produced cycles. In this step, we can select which border of a subgraph is appropriate as anchor point of an edge, because for instance $(u_1^{(+)}, u_2^{(-)})$, $(u_1^{(-)}, u_2^{(-)})$ and $(u_1^{(+)}, u_2^{(+)})$ represent the same connectivity edge. This again avoids reverted edges (convention (d)). If the choice of the borders still produces an edge against the edge orientation, we insert a reverted edge and mark it such that later the arrow head can be drawn at the correct end point.

After the removal of empty layers, we now split long edges that range over several layers into sequences of edge segments and dummy nodes. After this, the span of each edge (v, w) is $\mathbf{R}(w) - \mathbf{R}(v) = 1$. In a nested graph, each node must be assigned to a subgraph u. Thus, we must also add the dummy nodes to the nesting tree T'.

The border nodes $u^{(-)}$ and $u^{(+)}$ belong to u, i.e., we add the edges $u \xrightarrow[T']{} u^{(-)}$ and $u \xrightarrow[T']{} u^{(+)}$ to T'. Since unnecessary crossings of edges and borders of the subgraph rectangles

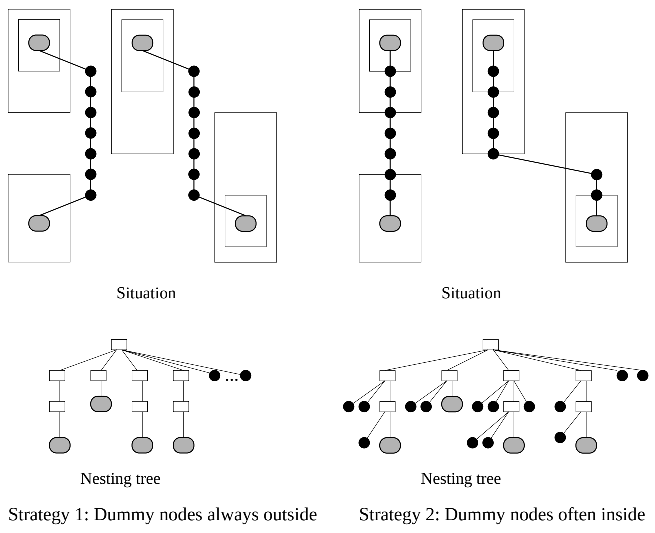

Situation
Situation
Nesting tree
Nesting tree
Strategy 1: Dummy nodes always outside
Strategy 2: Dummy nodes often inside

Figure 4: Strategies for the Dummy Node Assignment

should be avoided (convention (g)), a dummy node belonging to an edge (w_1, w_2) must be assigned at least to the subgraph that contains both w_1 and w_2. By this way, we avoid that an edge leaves a border rectangle unnecessarily and later reenters the rectangle producing two crossings with borders. There are two alternative strategies.

1) Edges (w_1, w_2) are routed mostly outside of border rectangles. If an edge cannot be routed completely inside $\Box(u)$, because either w_1 or w_2 do not belong to u, then all dummy nodes are placed outside of $\Box(u)$. As effect in the final layout, the edges cross the borders of the rectangles at the sides (Fig. 4, upper left). We look for the first common predecessor u of w_1 and w_2 in the nesting tree and add all dummy nodes of (w_1, w_2) to this subgraph u (Fig. 4, lower left).

2) Edges (w_1, w_2) are routed mostly inside of border rectangles. If w_1 or w_2 are inside $\Box(u)$, then all dummy nodes d of (w_1, w_2) with $\mathbf{R}_{min}(u) \leq \mathbf{R}(d) \leq \mathbf{R}_{max}(u)$ belong to u. In T' on the path from w_1 (and from w_2, resp.) to the root of T', we look for the first subgraph u that satisfies this condition and add the dummy node d to u (Fig. 4, lower right). As effect in the final layout, the edges often (but not always) cross the upper or lower borders of the rectangles (Fig. 4, upper right).

Because dummy nodes are added as leaves to T', we consider dummy nodes as elements of the set B in the following.

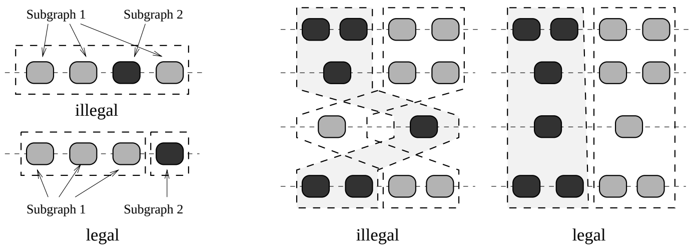


Figure 5: Forbidden Ordering within the Layers

6 Reduction of Crossings

Now, we have a proper hierarchy of layers [9] consisting of base nodes, border nodes that are used as start or end point of connectivity edges between subgraphs, and normal dummy nodes. Each node v has a relative position $\mathbf{P}(v)$ within its layer and belongs to one subgraph, i.e., it occurs in the nesting tree T'. All edge segments point downwards and have the span 1. In the usual layer layout method for simple graphs [8], nodes are now reordered within the layers according to the barycenter weights

$$\begin{aligned} W_p(v) &= \frac{1}{|\mathbf{pred}(v)|} \sum_{w \in \text{pred}(v)} \mathbf{P}(w) \\ W_s(v) &= \frac{1}{|\mathbf{succ}(v)|} \sum_{w \in \text{succ}(v)} \mathbf{P}(w) \end{aligned}$$

W_p is used in a top down traversal and W_s in a bottom up traversal of the layers. By this reordering, we get better $\mathbf{P}(v)$ for all base nodes v. However, because of the conventions (e) and (f) of compound graphs, there are two special rules:

- Nodes $v_1, \ldots, v_n$ of a subgraph u of the same rank must be placed in an unbroken sequence within the layer. If there is a node w not belonging to u that is placed between v_i and v_j, then it is not possible to draw a border rectangle around v_i and v_j that does not contain w (Fig. 5, left).
- If two subgraphs u and u' are not nested then u must be placed unambiguously to the left or to the right of u', i.e., both subgraphs must not be intertwined. Assume v_1 and v_2 belong to u and v_1' and v_2' belong to u' with $\mathbf{R}(v_1) = \mathbf{R}(v_1')$ and $\mathbf{R}(v_2) = \mathbf{R}(v_2')$. If $\mathbf{P}(v_1) < \mathbf{P}(v_1')$ and $\mathbf{P}(v_2) > \mathbf{P}(v_2')$, then it is impossible to draw two rectangles for u and u' that do not cross. Thus, this situation is illegal (Fig. 5, right).

Normal crossing reduction with barycenter weights does not respect these rules. However, if we start with the situation after normal crossing reduction, we have already few

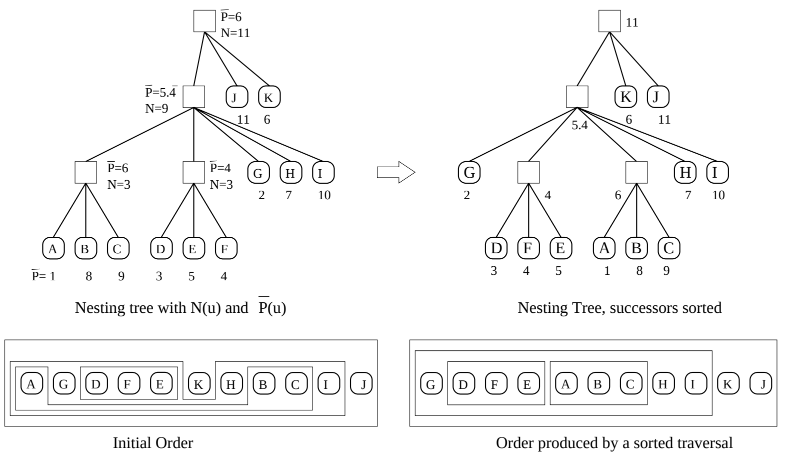


Figure 6: Sorted Traversal of Reduced Nesting Tree T'_i

edge crossings. This is a good starting point. Afterwards, we reposition the nodes within the layers carefully such that nodes of the same subgraph are placed in an unbroken sequence and subgraphs are not intertwined. The guide value of the relative position of a subgraph is the average position of its nodes. The *complete average position* of a subgraph u is

$$\overline{\mathbf{P}}(u) = \frac{1}{|\{v \in B \mid u \xrightarrow[T']{}^* v\}|} \sum_{v \in B, u \xrightarrow[T']{}^* v} \mathbf{P}(v)$$

However, we first consider each layer independently. The *average position* of u for layer i is

$$\overline{\mathbf{P}}_i(u) = \frac{1}{|\{v \in B \mid \mathbf{R}(v) = i, u \xrightarrow[T']{}^* v\}|} \sum_{v \in B, \mathrm{R}(v)=i, u \xrightarrow[T']{}^* v} \mathbf{P}(v)$$

The idea: if $\overline{\mathbf{P}}_i(u_1) < \overline{\mathbf{P}}_i(u_2)$ then we expect that many nodes belonging to u_1 are left of nodes of u_2.

We can annotate the nesting tree in time $O(|T'|)$ with the average positions because we need only one traversal of T'. We store at each node of T' the number $N_i(u)$ of leaves of T' (base nodes, border nodes or dummy nodes) that belong to u and have rank i. For simplicity of the formulas, we define $\overline{\mathbf{P}}_i(v) = \mathbf{P}(v)$ and $N_i(v) = 1$ for leaves of T'. Thus, during the traversal, we need only to consider the direct successors of u in T' according to the formula

$$\begin{aligned} N_i(u) &= \sum_{u \xrightarrow[T']{} w, \mathrm{R}(w)=i} N_i(w) \\ \overline{\mathbf{P}}_i(u) &= \frac{1}{N_i(u)} \sum_{u \xrightarrow[T']{} w, \mathrm{R}(w)=i} N_i(w)\, \overline{\mathbf{P}}_i(w) \end{aligned}$$

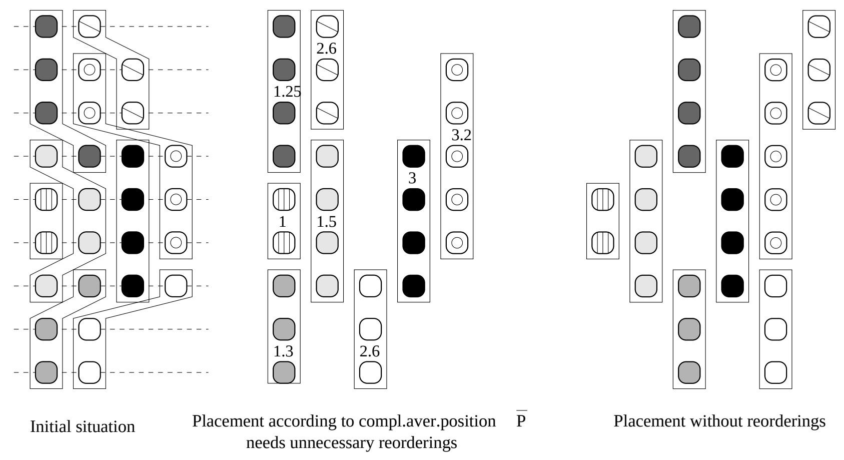

2.6
1.25
3.2
3
1
1.5
1.3
2.6
Initial situation
Placement according to compl.aver.position $\overline{\mathrm{P}}$
needs unnecessary reorderings
Placement without reorderings

Figure 7: Bad Reordering Using $\overline{\mathbf{P}}$

If we consider layer i independently, we use the nesting tree T'_i that is reduced such that it contains only leaves of rank i (Fig. 9, compare first and second row). This reduced nesting tree can be traversed such that the children of each inner node are sorted according to $\overline{\mathbf{P}}_i(u)$. This needs time $O(|T'_i| \log |T'_i|)$. By a sorted traversal, the leaves of T'_i are sorted such that all nodes of the same subgraph are placed in an unbroken sequence (Fig. 6).

If all layers are reordered by a sorted traversal of each T'_i, we know that we can draw subgraph rectangles at each layer, but – with respect to all layers – still two subgraphs can be intertwined. We could reorder all nodes of all layers in a similar way by the complete average position $\overline{\mathbf{P}}(u)$ instead of the average position per layer $\overline{\mathbf{P}}_i(u)$. This avoids intertwined subgraphs, but often reorders the layers much more than necessary (Fig. 7). Instead, we use the *subgraph ordering graph* (Fig. 8).

Definition 3 *The* **subgraph ordering graph** *consists of all nodes $w \in S \cup B$. It contains an edge (w, w') iff there are nodes v, v' and a subgraph u_0 with $\mathbf{R}(v) = \mathbf{R}(v')$ and $\mathbf{P}(v) = \mathbf{P}(v') - 1$ (i.e., v and v' are directly neighbored nodes at the same layer) and $w \neq w'$ and $u_0 \underset{T'}{\mapsto} w \underset{T'}{\to}^* v$ and $u_0 \underset{T'}{\mapsto} w' \underset{T'}{\to}^* v'$.*

The subgraph ordering graph consists of edges corresponding to the relation 'is left of'. If two nodes v and v' are neighbored and belong to subgraphs $u_1 \underset{T'}{\mapsto} \ldots \underset{T'}{\mapsto} u_n$ and $u'_1 \underset{T'}{\mapsto} \ldots \underset{T'}{\mapsto} u'_m$, then each u_i is left of each u'_j, thus we normally need edges between all pairs (u_i, u'_j) with $1 \leq i \leq n$ and $1 \leq j \leq m$. However, if u_1 is left of u'_1, the nesting structure automatically implies that each u_i is left of each u'_j. Thus, it is sufficient to add the edge (u_1, u'_1) to the subgraph ordering graph.

If we sort the subgraph ordering graph topologically, we get the ordering λ_O denoting which subgraph is left of other subgraphs. If two subgraphs are intertwined, the subgraph ordering graph is cyclic and cannot be sorted topologically. We must break these

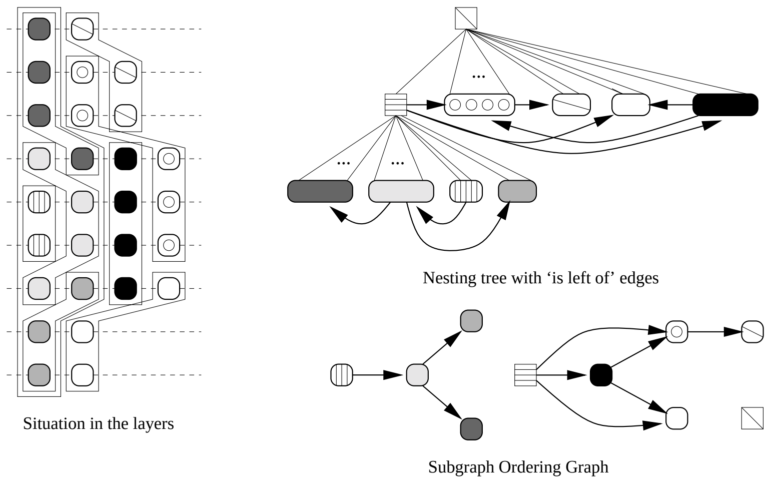


Figure 8: Subgraph Ordering Graph

cycles by removing edges. Note that there is no direct relation between the number of removed edges and the number of reordered nodes. Thus it is not necessary to look for the minimum number of edges to be removed (minimum feedback arc set problem, which is NP complete). Instead, we use as heuristics the complete average position of subgraphs. Cycles of the subgraph ordering graph are broken at the node w with smallest $\overline{\mathbf{P}}(w)$.

If we perform a sorted traversal of all layers according to the order λ_O, all nodes are reordered such that they are placed in an unbroken sequence within the layers and no subgraphs are intertwined. As a variant, we can make the subgraph ordering acyclic by the heuristics mentioned above, but it is not necessary to use the *same* λ_O for each layer. Topological orderings need not be unique for a given graph. As long as a $\lambda_{O,i}$ for the ith layer is a topological ordering of the subgraph ordering graph, we get a legal placement of the nodes. If during the calculation of $\lambda_{O,i}$ there is a choice between several next nodes, we take the node with the smallest average barycenter weight

$$W_{p,i}(u) = \frac{1}{|\{v \in B \mid \mathbf{R}(v) = i, u \xrightarrow[T']{}^* v\}|} \sum_{v \in B, \mathrm{R}(v)=i, u \xrightarrow[T']{}^* v} W_p(v)$$

Because this formular is similar to the formula $\overline{\mathbf{P}}_i(u)$, we can use the same calulation method in time $O(|T_i'|)$. This combines the normal crossing reduction by barycenter weights with the special method of compound graphs. Of course, W_p is used during a top down traversal of the layers, and a similarly defined $W_{s,i}$ is used during bottom up traversals of the layers. To sum up, the algorithm is

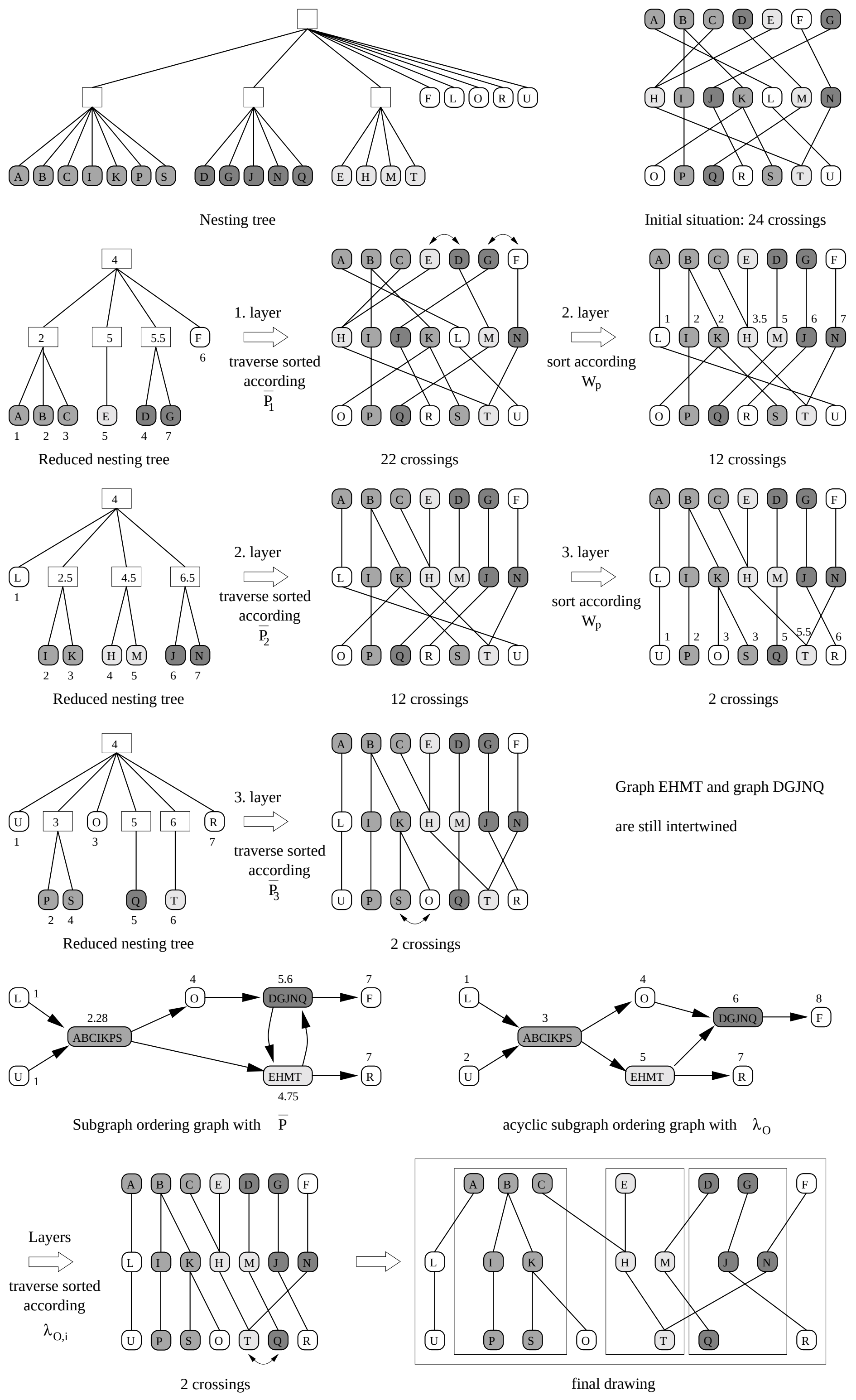

Nesting tree
Initial situation: 24 crossings
Reduced nesting tree
1. layer
traverse sorted according $\overline{P}_1$
22 crossings
2. layer
sort according W_p
12 crossings
Reduced nesting tree
2. layer
traverse sorted according $\overline{P}_2$
12 crossings
3. layer
sort according W_p
2 crossings
Reduced nesting tree
3. layer
traverse sorted according $\overline{P}_3$
2 crossings
Graph EHMT and graph DGJNQ
are still intertwined
ABCIKPS
DGJNQ
EHMT
Subgraph ordering graph with $\overline{P}$
acyclic subgraph ordering graph with λ_O
Layers
traverse sorted according $\lambda_{O,i}$
2 crossings
final drawing

Figure 9: Crossing Reduction of Compound Graphs

```
Algorithm: Crossing Reduction of Compound Graphs
        Given: Proper hierarchy $H = [V_1, \ldots, V_n]$ with set $E_G$ of connectivity edges
                  and set $S$ of subgraphs. Let $T'$ be the nesting tree of this compound graph
(1)     $C$ = Number of Crossings in this hierarchy;
(2)     while $C$ is still unsatisfactory do
(3)           Sorted Traversal of $T'_1$ according to $\overline{\mathbf{P}}_1$
(4)           // this produces a placement with unbroken sequences of subgraph nodes in layer 1
(5)           for each $V_i$ from $i = 2$ to $n$ do
(6)                 for each $v \in V_i$ do calculate $W_p(v)$; od
(7)                 Sort $V_i$ by $W_p$;
(8)                 Sorted Traversal of $T'_i$ according to $\overline{\mathbf{P}}_i$
(9)                 // this produces a placement with unbroken sequences of subgraph nodes in layer i
(10)          od
(11)          Calculate subgraph ordering graph OG;
(12)          Sort OG topologically and break cycles if necessary
(13)          for each $w \in V_1 \cup S$ do $W_{p,1}(w) = 1$; od
(14)          // We need an artificial $W_{p,1}$ to calculate $\lambda_{O,1}$
(15)          Sorted traversal of $T'_1$ according to $\lambda_{O,1}$
(16)          for each $V_i$ from $i = 2$ to $n$ do
(17)                for each $w \in V_i \cup S$ do calculate $W_{p,i}(w)$; od
(18)                // We need $W_{p,i}$ to calculate $\lambda_{O,i}$
(19)                Sorted traversal of $T'_i$ according to $\lambda_{O,i}$
(20)          od
(21)          ...similar with bottom up layer traversals ...
(22)          $C$ = Number of Crossings in this hierarchy;
(23)    od
```

Lines (3)-(10) produce a ordering with unbroken sequences at the layers. This is the precondition that the subgraph ordering graph OG can be calculated. Lines (13)-(20) reorder the layers such that subgraph u is placed completely left of subgraph u', if (u, u') is an edge in OG after OG is made acyclic. If we do this at layer i, the barycenter weights of layer $i+1$ can change completely. Thus, the $W_{p,i+1}$ values must be recalculated and influence the placement by $\lambda_{O,i+1}$. Figure 9 shows a traversal of the layers by this method.

7 Positioning of Nodes and Edges

Finally, we calculate the absolute positions of the nodes such that there is sufficient space for the border rectangles. Here, we cannot place each layer independently, because this may yield positions where it is impossible to draw straight vertical border lines between the subgraphs (compare with Fig. 7, left). Border rectangles should not overlap. Thus we introduce artificial sequences of dummy nodes to the left and to the right of each subgraph. A *left border segment* of a subgraph u consists of dummy nodes v_i for each layer $i = \mathbf{R}_{min}(u), \ldots, \mathbf{R}_{max}(u)$ and invisible edges (v_i, v_{i+1}) between these dummy nodes.

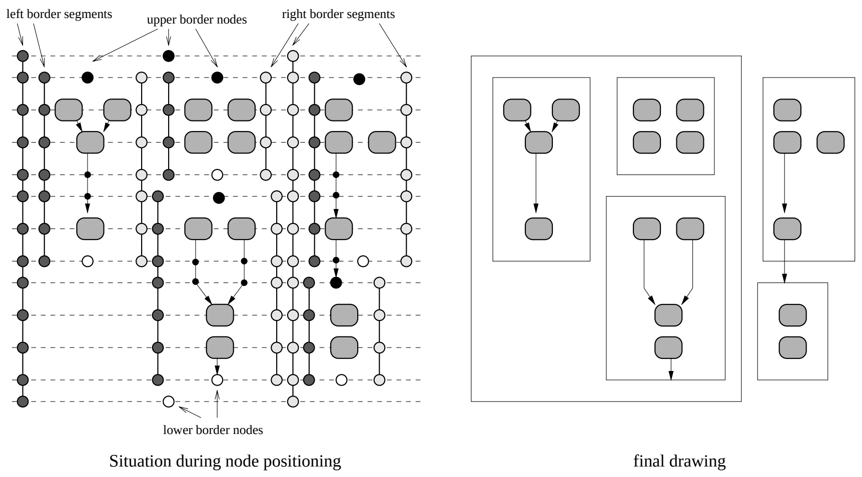


Figure 10: Border Segments and Border Nodes

The dummy nodes are added to the left of the nodes of u into the layers. A *right border segment* is added similarly (Fig. 10).

In [6], we called such sequences of dummy nodes and edges *linear segments*, and we presented a layout method that places linear segments as straight vertical lines without bendings. This method additionally considered conventions (a), (b) and (c). We use it now to place the nodes and to route the edges. Because left and right border segments are placed as straight lines, we can draw the real border of the subgraph rectangle there. It runs exactly along the border nodes such that in the final picture, the edges to subgraphs seem to point to the rectangles.

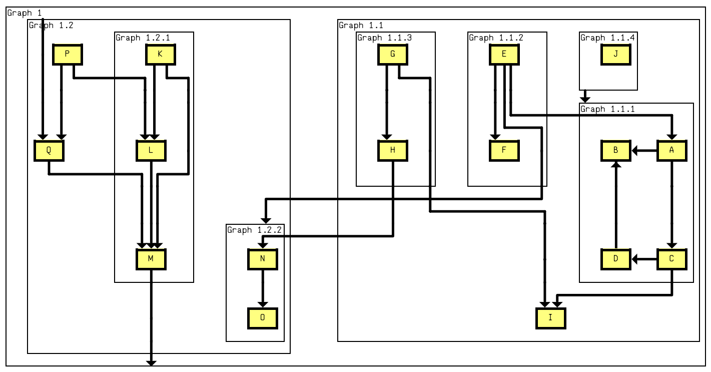
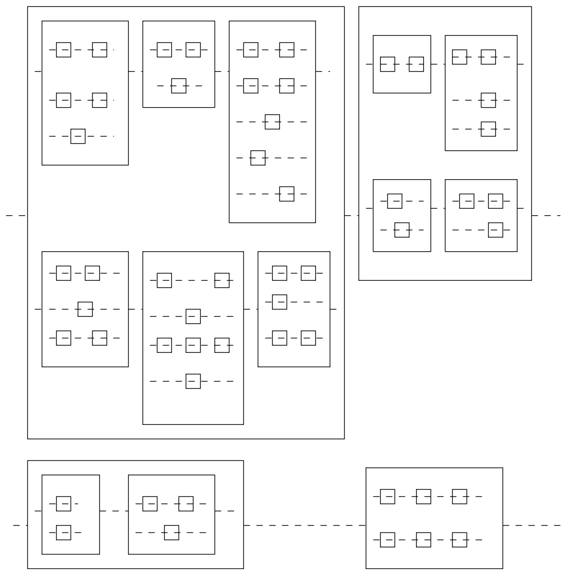


Figure 11: Compound Graph


Layers in D-Abductor

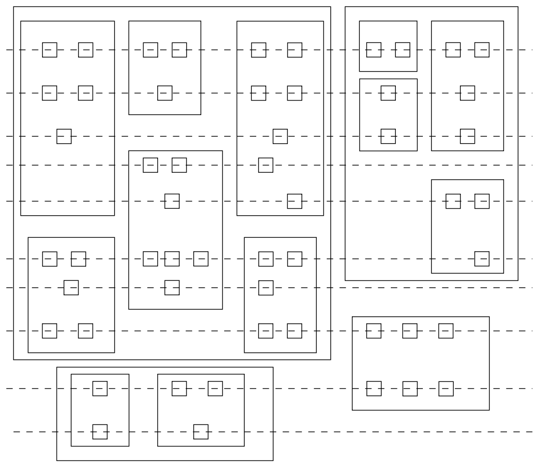

Layers in VCG tool

Figure 12: Differences of Layout Methods

8 Conclusion and Comparision with Similar Methods

The algorithm is implemented in the VCG tool [5]. Figure 1 was produced by the VCG tool. Figure 11 shows another typical result produced with this algorithm.

As mentioned above, the presented layout method has similarities with the layout method used in the tool D-Abductor [7]. Both methods are based on the layout in layers. There are some differences:

- In [7], partitioning of nodes into layers is based on a labeling of the *nesting tree* instead of the *nesting graph*. In principle, this corresponds to the calculation of the upper rank $\mathbf{R}_{min}$ of subgraphs while the lower rank $\mathbf{R}_{max}$ is ignored. This may result in unnecessary reversions of edges while we try to avoid these as described in section 5.

- The VCG tool starts the layout from a global situation of the graph while D-Abductor preferably treats subgraphs locally. For instance, the layer lines are globally valid for all subgraphs in the VCG tool, i.e. all nodes of one layer have nearly the same y coordinate independent of the subgraphs they belong to (Fig. 12, right). In [7], each subgraph has own layer positions (Fig. 12, left).

- The layer partitioning in [7] is converted into a *proper compound digraph* by using compound dummy nodes. In a proper compound digraph, the upper corners of subgraphs of one layer are placed at the same y coordinate. Arrangement that vertically overlap as in Fig. 12, right, are not allowed while they can be produced by the VCG tool.

- As side effect of the proper compound digraph, edges are routed mostly outside of border rectangles and cross the borders at the sides. This can be compared with the effect of Fig. 4, above left. Our algorithm is more flexible here.

- However, there are often less dummy nodes in a proper compound digraph than in the global partitioning used by the VCG tool. Since the speed of the layout methods is influenced by the number of dummy nodes, the method of [7] is often faster.

- Crossing reduction is based on the barycenter sorting of the layers in both methods. [7] works recursivly through the nest of subgraphs and calculates local crossing reduction while respecting the global situation at the same time. We start with a global crossing situation, where subgraph borders are still not respected, and reorder the layers carefully afterwards such that subgraph rectangles can be drawn.

Other layout methods are restricted to recursive layout where subgraphs are treated as large nodes. Edges pointing beyond the border of subgraphs are ignored during the crossing reduction. Thus they are not routed optimally [3, 4]. Such a method is used in the Edge tool [4] that allows constraint specifications by the user to influence partitioning and crossing reduction.

The method of the VCG tool gived good results, is flexible and fully automatically, i.e. it does not need user constraints. It is well suited for the layout of block diagrams and nested compiler graphs, as the VCG tool is designed for. It is well integrated in the normal layout method of simple graphs such that it is possible to use all variants of layouts of the VCG tool (e.g. compound orthogonal layouts, compound spline layouts and compound polygon layouts).

References

[1] **Eades, P.; Sugiyama, K.**: *How to Draw a Directed Graph*, Journal of Information Processing, 13 (4), pp. 424-437, 1990.

[2] **Feng, Q.W.; Eades, P.; Cohen, R.F.**: *Planar Drawings of Clustered Graphs*, Technical Report 05-95, Department of Computer Science, The University of Newcastle, Australia, 1995.

[3] **Henry, T.R.**: *Interactive Graph Layout: The Exploration of Large Graphs*, Ph. D. Thesis, TR 92-03, Department of Computer Science, The University of Arizona, 1992.

[4] **Paulisch, F.N.; Tichy, W.F.**: *EDGE: An Extendible Graph Editor*, Software – Practice and Experience 20 (S1), pp. 63-88, 1990.

[5] **Sander, G.**: *Graph Layout Through the VCG Tool*, in Tamassia, R.; Tollis, I.G., eds.: Graph Drawing, Proc. DIMACS Intern. Workshop GD'94, LNCS 894, pp. 194-205, Springer, 1995.

[6] **Sander, G.**: *A Fast Heuristic for Hierarchical Manhattan Layout*, in Brandenburg, F.J., ed.: Graph Drawing, Proc. Symposium on Graph Drawing, GD'95, LNCS 1027, pp. 447-458, Springer, 1996.

[7] **Sugiyama K.; Misue K.**: *Visualization of Structural Information: Automatic Drawing of Compound Digraphs*, IEEE Trans. Sys., Man, and Cybernetics, 21(4), pp. 876-892, 1991.

[8] **Sugiyama, K.; Tagawa, S.; Toda, M.**: *Methods for Visual Understanding of Hierarchical Systems*, IEEE Trans. Sys., Man, and Cybernetics, SMC 11(2), pp. 109-125, 1981.

[9] **Warfield, J.N.**: *Crossing Theory and Hierarchy Mapping*, IEEE Trans. Sys., Man, and Cybernetics, SMC 7(7), pp. 505-523, 1977.