Dynamic Graph Drawing of Sequences of Orthogonal and Hierarchical Graphs

Carsten Görg[1], Peter Birke[1], Mathias Pohl[1], and Stephan Diehl[2]

[1] Saarland University, FR Informatik, D-66041 Saarbrücken, Germany
goerg@cs.uni-sb.de
[2] Catholic University Eichstätt, Informatik, D-85072 Eichstätt, Germany
diehl@acm.org

Abstract. In this paper we introduce two novel algorithms for drawing sequences of orthogonal and hierarchical graphs while preserving the mental map. Both algorithms can be parameterized to trade layout quality for dynamic stability. In particular, we had to develop new metrics which work upon the intermediate results of layout phases. We discuss some properties of the resulting animations by means of examples.

1 Introduction

In many applications graphs are not drawn once and for all, but change over time. In some cases all changes are even known beforehand, e.g. if we want to visualize the evolution of a social network based on an email archive, or the evolution of program structures stored in software archives. In these kinds of applications each graph can be drawn being fully aware of what graphs will follow. Unfortunately, to the best of our knowledge there exist only two algorithms that take advantage of this knowledge, namely TGRIP [6] and Foresighted Layout [8]. See Section 6 for a discussion of these and other approaches. While the former was restricted to spring embedding, the latter is actually a generic algorithm.

Recently we introduced *Foresighted Layout with Tolerance (FLT)* [7] for drawing sequences of graphs while preserving the mental map and trading layout quality for dynamic stability (tolerance). The algorithm is generic in the sense that it works with different static layout algorithms with related metrics and adjustment strategies. As an example we looked at force-directed layout. In this paper we apply FLT to orthogonal and hierarchical layout, which means that we have to develop adjustment strategies and metrics for these. We also improve FLT by introducing the importance-based backbone as a generalization of the supergraph of a sequence of graphs.

2 Improved Adjusted Foresighted Layout

In our previous work the supergraph, which is the union of all graphs in a graph sequence played a crucial role. The reason for using the supergraph was that it provided all information about the graph sequence and that its layout could be used as a sketch for all graphs in the sequence. However, the supergraph is

J. Pach (Ed.): GD 2004, LNCS 3383, pp. 228–238, 2004.

restrictive, as it induces a layout for all nodes without taking into account that they are of different relevance for the sequence.

To improve that model we now introduce the concept of a backbone of a sequence. Therefore we need a function that defines the importance of a node in the sequence $g_1, \ldots, g_n$. In the following we assume that $g_i = (V_i, E_i)$.

Definition 1 (Backbone). *Given a sequence of graphs $g_1, \ldots, g_n$, and a mapping* $\mathsf{importance} : V \to \mathbb{N}$*, then* $V_B = \{v \in \bigcup_{i=1}^{n} V_i \mid \mathsf{importance}(v) \geq \delta_B\}$ *and* $E_B = \{(u, v) \in \bigcup_{i=1}^{n} E_i \mid u, v \in V_B\}$ *define the* backbone $B = (V_B, E_B)$ *of a graph sequence $g_1, \ldots, g_n$ with respect to a threshold $\delta_B \in \mathbb{N}$.*

This concept of the backbone is a generalization of the concept of the supergraph: The backbone is less restrictive and is adjusted to the given graph sequence. But setting $\delta_B = 0$ will create a backbone that is equal to the supergraph.

Dependent on the choice of the importance function, the backbone represents different base models. There are several possibilities for choosing an importance function: We can define the function depending on the structure of the sequence (for example the number of occurrences of a node in the sequence: $\mathsf{importance}(v) = |\{i \mid v \in V_i\}|$ for a graph sequence $g_1, \ldots, g_n$). If we know enough about the semantics of the graphs, we can instead choose an importance function that takes this information into account, i.e. we can use application-domain specific importance functions.

The improved algorithm for foresighted layout that uses the backbone instead of the supergraph now looks as follows:

Algorithm 1 Improved Foresighted Layout with Tolerance.

```
compute global layout L for the backbone B of g_1, ..., g_n
for i := 1 to n do
  L_i := L|_{g_i}
  l_i := adjust(...)
end for
animate graph sequence
```

In this improved version the global layout does not provide initial layout information for nodes $v \in V_i - V_B$, i.e. those that are not part of the backbone. So the adjustment functions have to assign initial positions to these nodes.

3 Orthogonal Foresighted Layout with Tolerance

Brandes et al. presented in [2] an orthogonal graph drawing algorithm that produced an orthogonal layout with few bends in the Kandinsky model while preserving the general appearance of a given sketch. The angle and the bend changes can be controlled by parameters α and β. In this section, we show how to extend this approach so that it fits in our framework, i.e. it applies the backbone concept and is guided by metrics. We assume that the reader is familiar with the Kandinsky network [2, 13].

First we present the general algorithm. After computing the orthogonal layout for the given backbone and obtaining the corresponding quasi-orthogonal shape, we build a sketch S_i for each graph of the sequence and adjust it through the adjustOrth() algorithm. The sketch is a combination of the previous graph's layout and the backbone's layout restricted to the current graph. If a conflict between the layout of the backbone and the previous graph exists, we choose the one of the backbone.

Algorithm 2 Orthogonal Foresighted Layout with Tolerance.

compute orthogonal layout L_0 for the backbone B of $g_1, \ldots, g_n$
$Q_0 :=$ quasiOrthogonalShape(L_0)
for $i := 1$ **to** n **do**
  $S_i := (L_0 \oplus L_{i-1})|_{g_i}$ // $(L_i \oplus L_j)(x)$ is defined as $L_i(x)$ if $x \in dom(L_i)$ and $L_j(x)$ otherwise.
  $(Q_i, L_i) :=$ adjustOrth$(S_i, g_i, L_{i-1}, Q_{i-1})$
end for
animate graph sequence

The adjustOrth() algorithm first computes the extended network of the sketch. Since the sketch was restricted to the current graph g_i, we only have to handle insertions of new nodes and edges. The insertion of a new node creates a new vertex-node in the Kandinsky network. How to insert new edges adjacent to vertex-nodes with a degree greater than 0 is presented in [2]. The insertion of a new edge adjacent to a vertex-node with a degree of 0 does not create a new face-node.

We initialize the locally (for every edge) used parameters α and β. Then we compute the quasi-orthogonal shape as described in [2]. To compare this shape with that of the previous graph, we define a new metrics for quasi-orthogonal shapes. To this end, we extend the definition of a quasi-orthogonal shape given in [2]. With $Q(f, i)$ we denote the i-th tuple of $Q(f)$, with $\mathsf{edge}(Q, f, i)$ the value of the edge field, with $a(Q, f, i)$ the value of the angle field, and with $b(Q, f, i)$ the value of the bend field of $Q(f, i)$. The value of the edge field of the successor tuple of $Q(f, i)$ is $\mathsf{succEdge}(Q, f, i) = \mathsf{edge}(Q, f, (i + 1) \bmod |Q(f)|)$.

Definition 2 (Quasi-orthogonal-shape metrics).
Let $\mathcal{Q}$ be the set of quasi-orthogonal shapes. The function $\mathsf{diff}_\alpha : \mathcal{Q} \times \mathcal{Q} \to \mathcal{P}(E)$,

$$(Q_1, Q_2) \mapsto \{e = \mathsf{edge}(Q_1, f, i) \mid \exists f', j : e = \mathsf{edge}(Q_2, f', j) \ \wedge \ \mathsf{succEdge}(Q_1, f, i) = \mathsf{succEdge}(Q_2, f', j) \ \wedge \ a(Q_1, f, i) \neq a(Q_2, f', j)\}$$

defines the set of edges with the same successor edge, but with different angles in two quasi-orthogonal shapes. The function $\mathsf{diff}_\beta : \mathcal{Q} \times \mathcal{Q} \to \mathcal{P}(E)$,

$$(Q_1, Q_2) \mapsto \{e = \mathsf{edge}(Q_1, f, i) \mid \forall f', j \text{ with } e = \mathsf{edge}(Q_2, f', j) : b(Q_1, f, i) \neq b(Q_2, f', j)\}$$

defines the set of edges with different bends in two quasi-orthogonal shapes. Then the function Δ_α with $\Delta_\alpha(Q_1, Q_2) = |\mathsf{diff}_\alpha|$ is called angle metrics and the function Δ_β with $\Delta_\beta(Q_1, Q_2) = |\mathsf{diff}_\beta|$ is called bend metrics.

If the angle metrics does not fulfill the given angle threshold and there is an α that is lower than the maximal value (the maximal value $6 \cdot |V_i|$ results from the

construction of the Kandinsky network and [13]), we increment the corresponding α. We deal analogously with the bend metrics and β. The construction of the modified Kandinsky network implies that incrementing β could lead also to a change of angle between two edges. If angle stability is more important than bend stability, then both β and α have to be incremented if the bend metrics does not fulfill the given bend threshold.

The last step concerns compaction. To be able to preserve the edge length of the sketch S_i, we extend the compaction algorithm from [9] by edges of prescribed length. This extension is done straightforwardly by extending the length function: let $e = (u, v)$ be an edge and (u_x, u_y) the position of u in S_i, then

$$\mathsf{length}'(e) = \begin{cases} |u_x - v_x| + |u_y - v_y|, & \text{if e is fixed} \\ \mathsf{length}(e), & \text{otherwise} \end{cases}$$

An edge can be fixed if it is in the current graph as well as in the previous one, and if the values of the corresponding bend fields are equal. We compute the final layout by applying the extended compaction algorithm. If the metrics does not fulfill the given threshold we fix one more edge if there are any left.

Algorithm 3 adjustOrth$(S_i, g_i, L_{i-1}, Q_{i-1})$ predecessor dependent.

$N_i :=$ compute extended network(S_i, g_i)
$\forall e \in E_i : \alpha_e := 0, \beta_e := 0$
repeat
  $Q_i :=$ quasiOrthogonalShape(N_i, α, β)
  if $\Delta_\alpha(Q_i, Q_{i-1}) > \delta_\alpha \wedge \exists e \in \mathsf{diff}_\alpha(Q_i, Q_{i-1}) : \alpha_e < 6 \cdot |V_i|$ **then**
    $\forall e \in \mathsf{diff}_\alpha : \mathsf{inc}(\alpha_e)$
  end if
  if $\Delta_\beta(Q_i, Q_{i-1}) > \delta_\beta \wedge \exists e \in \mathsf{diff}_\beta(Q_i, Q_{i-1}) : \beta_e < 6 \cdot |V_i|$ **then**
    $\forall e \in \mathsf{diff}_\beta : \mathsf{inc}(\beta_e)$
  end if
until done
fixedEdges $:= \emptyset$
repeat
  $L_i =$ compact$(Q_i, S_i,$fixedEdges$)$
  if $\Delta(L_{i-1}, L_i) > \delta \wedge$ fixedEdges $\subset \{E_i \cap E_{i-1}\} - \{\mathsf{diff}_\beta\}$ **then**
    extend fixedEdges by one edge of $\{E_i \cap E_{i-1}\} - \{\mathsf{diff}_\beta\}$
  end if
until done
return (Q_i, L_i)

So far, we have seen how to apply orthogonal layout to the predecessor layout strategy. But it is also possible to apply it to the simultaneous layout strategy. In this case the backbone layout is used as sketch and we use global parameters α and β instead of local ones to achieve a more uniform adjustment of angles and bends over the whole sequence. The adjustOrth() algorithm first computes the quasi-orthogonal shapes for all graphs. If the condition for the angle metrics $\exists i : \Delta_\alpha(Q_{i-1}, Q_i) > \delta_\alpha \ \wedge \ \alpha < 6 \cdot |V_i|$ is not fulfilled, i.e. there is a tuple of successive shapes which do not hold the angle metrics condition and there is some

space for improvement, α is increased. Analogously, β is changed depending on the bend metrics. To compute the final layouts we proceed as in the predecessor-dependent layout algorithm.

4 Hierarchical Foresighted Layout with Tolerance

The computation of a hierarchical layout of a graph following the Sugiyama approach needs several phases: First all nodes are distributed in discrete layers (the ranking phase), then the nodes of each layer are arranged, and finally the layout is computed from the layers and their arrangements. One of the problems that occur when trying to apply FLT to hierarchical layout is that there is no option for global layout adjustment such as temperature annealing in the force-directed approach. Instead, we have to divide the adjustment in standard foresighted layout into two different adjustments: an adjustment for the ranking phase and an adjustment for the rank sorting phase. However, after the ranking adjustment has been performed, we cannot apply standard metrics, as the graphs are not fully layouted. Therefore we will introduce a new kind of metrics which only concerns the rankings of two graphs.

4.1 Predecessor Dependent Layout

In this section we describe the two different adjustment steps of hierarchical foresighted layout. Starting from the input sequence, we compute the backbone first. As the nodes of the backbone are of highest importance, we try to preserve the mental map of the graph sequence by fixing these nodes to a certain rank for the entire graph sequence. A good approach is to fix the node to the median of all local rankings, which are computed in advance. So we achieve an optimal rank for at least one graph.

Definition 3 (Average ranking). *A ranking $R : V \to \mathbb{N}$ is a mapping from a node set to the set of natural numbers. Given a sequence of graphs $g_1, \ldots, g_n$ with rankings $R_1, \ldots, R_n$, the average ranking $\bar{R} : V \to \mathbb{N}$ is defined by the median of all $R_i(v)$.*

After that, we compute local rankings for each graph, with respect to the ranking of the backbone. In the second phase, we try to arrange the nodes on each rank, such that we preserve the mental map, but try to reduce the edge crossings at the same time. The general algorithm for hierarchical foresighted layout using the predecessor dependent adjustment is shown in Algorithm 4.

Rank Assignment. In this section we describe how the ranks are adjusted. We compute a new ranking by sorting g_i topologically, but all nodes of the backbone are ranked to their given backbone rank. If the metrics of the rank distance (which we describe below) between the current and the previous ranking exceeds the given threshold δ_R, we fix the rank of one more node to the rank of the previous layout. We choose a node with maximal importance from the node

Algorithm 4 Hierarchical Foresighted Layout with Tolerance.

compute backbone B of $g_1, \ldots, g_n$
compute average ranking R_0 of B
for $i := 1$ **to** n **do**
  $R_i^l := R_0|_{g_i}$
  $R_i := \mathsf{adjustRank}(R_i^l, R_{i-1}, g_i)$
  $(l_i, \sigma_i) := \mathsf{adjustOrder}(g_i, R_{i-1}, R_i, \sigma_{i-1}, l_{i-1})$ // with $dom(\sigma_0) = \emptyset$ and $dom(l_0) = \emptyset$
end for
animate graph sequence

set with the following properties: the nodes are contained in the current and previous graph, but not in the backbone. Then we compute a new topological sorting. We repeat this process until the given threshold is no longer exceeded or until all nodes have fixed ranks. In the second case, we stop with a result which has minimal rank distance.

Algorithm 5 $\mathsf{adjustRank}(R_i^l, R_{i-1}, g_i)$ predecessor dependent.

compute R_i by sorting g_i topologically with respect to R_i^l
repeat
  if $\Delta_R(R_{i-1}, R_i) > \delta_R$ **then**
    add node $v \in \{w \mid w \in (V_i \cap V_{i-1}) - dom(R_i^l)$ and $\forall u \in (V_i \cap V_{i-1}) - dom(R_i^l) :$ $\mathsf{importance}(w) \geq \mathsf{importance}(u)\}$ to R_i^l and let $R_i^l(v) = R_{i-1}(v)$
    compute R_i by sorting g_i topologically with respect to R_i^l
  end if
until $(V_i \cap V_{i-1}) - dom(R_i^l) = \emptyset \vee \Delta_R(R_{i-1}, R_i) \leq \delta_R$
return R_i

Mental Distance on Ranks. As described in our previous work, we use several metrics to check the mental distance between two layouted graphs. In the layer-assignment phase of hierarchical layout we need a metrics to check the distance between two layer-assignments, but layered graphs do not provide all necessary information for a standard metrics. The only known value is in which layer a node belongs. Therefore we introduce a new kind of metrics for the mental distance, the rank metrics.

Definition 4 (Rank metrics). *Let (g, R) be a graph g with a ranking R. Then the function Δ_R that maps $((g, R), (g', R'))$ to a positive real number is called a rank metrics. In particular, $\Delta_R((g, R,), (g', R')) = 0$ means that g and g' have a non-distinguishable ranking.*

It turns out that there is only a small degree of freedom in the choice of a reasonable rank metrics. A very general approach for such a metrics could be the distance-rank metrics.

Definition 5 (Distance-rank metrics). *Given (g, R) and (g', R'), the function Δ_D with*

$$\Delta_D((g, R), (g', R')) = \sum_{v \in V \cap V'} |R(v) - R'(v)|$$

is called distance-rank metrics.

The definition of the distance-rank metrics could be changed by using the term $(R(v) - R'(v))^2$ instead of $R(v) - R'(v)$. This change would cause the metrics to be more sensitive to nodes that jump over several ranks.

Arrangement of Layers. In this phase we try to minimize edge crossings while staying as close as possible to the predecessor arrangement of layers. Therefore we define an order in each layer.

Definition 6 (Order within ranks). *Given a ranking R of a graph $g = (V, E)$, the function $\sigma : V \to \mathbb{N}$ denotes the order within ranks, if the following property holds: $\forall v, w \in V : R(v) = R(w) \Rightarrow \sigma(v) \neq \sigma(w)$. From the function σ we derive the partial order $<_\sigma$ on nodes: $v <_\sigma w \Leftrightarrow \sigma(v) < \sigma(w)$.*

Algorithm 6 computes an initial order σ_i of nodes which fulfills the following relative orderedness conditions with respect to its predecessor (for $i > 1$):

1. $\forall v, w \in V_i \cap V_{i-1} \wedge R_i(v) = R_{i-1}(v) \wedge R_i(w) = R_{i-1}(w)$:
   $v <_{\sigma_i} w \iff v <_{\sigma_{i-1}} w$
2. $\forall v \in V_i \cap V_{i-1} \wedge R_i(v) \neq R_{i-1}(v)$:
   $\left|\sigma_i(v) - \frac{\sigma_{i-1}(v)}{|\{w|R_{i-1}(w)=R_{i-1}(v)\}|} \cdot |\{w \mid R_i(w) = R_i(v)\}|\right| \leq 1$

The first condition states that the relative order of the nodes in the same rank in the current and predecessor graph is preserved. The second condition says that nodes which have changed their rank from the predecessor to the current layout preserve their relative layout position.

Then we compute $\hat{\sigma}_i$ by smoothly sorting the layers of g_i, where $<_{\hat{\sigma}_i}$ restricted to the j-th layer $\{v|R_i(v) = j\}$ forms a total order. As there exists no constraints for σ_1, $\hat{\sigma}_1$ is obtained by sorting the layers of g_1.

The layers of g_i can be sorted either by the barycenter heuristic or the median heuristic (see [1]). Sorting smoothly with respect to sortmax means using an arbitrary comparison-based sorting algorithm[1] where $a \leq b \cdot \mathsf{sortmax}$ is used instead of $a \leq b$. Similarly to simulated annealing, we can use linear, logarithmic or exponential decrease of the factor sortmax.

Definition 7 (Final layout). *Given a ranking R and an order of ranks σ of graph g, then $\mathcal{L}(R, \sigma)$ is the final hierarchical layout of g.*

Computing the final layout includes all remaining phases after sorting the ranks and yields the absolute positions of all nodes and edges. Thus we can now check whether the mental map is preserved using some standard metrics. If not, we decrease sortmax and start over.

4.2 Simultaneous Layout

In this section we illustrate how to apply the simultaneous adjustment strategy to hierarchical layout. The predecessor adjustment strategy of the previous section tries to adjust a layout as much as possible with respect to its predecessor.

[1] E.g. bucketsort is one of the rare cases that does not belong to this class.

Algorithm 6 adjustOrder$(g_i, R_{i-1}, R_i, \sigma_{i-1}, l_{i-1})$ predecessor dependent.

```
sortmax := 1
σ_i := initialOrder(σ_{i-1}, R_{i-1}, R_i)
repeat
  σ̂_i := smoothSort(g_i, σ_i, R_i, sortmax)
  l_i := L(R_i, σ̂_i)
  dec(sortmax)
until Δ(l_{i-1}, l_i) ≤ δ ∨ sortmax < 0
return (l_i, σ̂_i)
```

In contrast the simultaneous adjustment strategy provides a uniform adjustment of all graphs. The main problem in applying the simultaneous adjustment strategy to hierarchical layout arises in the rank assignment phase. A possible approach in the rank phase would be to perform a topological sorting on all graphs simultaneously. But this requires that in each iteration one node in each graph is ranked and the mental distance on ranks has to be checked. If the check fails, backtracking has to be performed and the rank of the last node that was ranked has to be fixed. Indeed, this approach is not a good choice for the layer assignment of large graph sequences – in that case it is more efficient to limit the simultaneous adjustment strategy to the layer assignment phase and to use the predecessor dependent rank assignment phase.

The goal of the simultaneous arrangement of layers is to preserve the relative node order in ranks over the whole sequence. Nodes which change their ranks should preserve at least their relative position. To achieve this goal we compute a global enumeration σ^* of the nodes which is consistent throughout the entire graph sequence. Therefore we build the supergraph, layout it using a static hierarchical layout algorithm and after that we retrieve the desired enumeration by projecting the nodes on the x-axis and reading them from left to right.

A local improved enumeration σ' can be derived from σ^* by adjusting the enumeration such that nodes which have changed their rank preserve their relative position (as described in Section 4.1, second relative orderedness condition). Using σ^* and σ' we define $\sigma = (\sigma_1, \ldots, \sigma_n)$:

$$\sigma_i = \begin{cases} \sigma_1^*, \text{ if } i = 1 \\ \sigma_i^*, \, i > 1 \text{ and } \Delta(\mathcal{L}(R_i, \sigma_i^*), \mathcal{L}(R_{i-1}, \sigma_{i-1})) < \Delta(\mathcal{L}(R_i, \sigma_i'), \mathcal{L}(R_{i-1}, \sigma_{i-1})) \\ \sigma_i', \text{ otherwise} \end{cases}$$

In Algorithm 7, starting with this initial order, we now use the same iteration as in Algorithm 6, except that we use a global sortmax-variable.

5 Examples

In Figure 1 we show snapshots from three different animations of the same graph sequence, which consists of evolving Hesse-graphs. (Hesse-graphs represent divisibility on natural numbers: there is an edge between v and w, if w is divisible by v.) In the graphs 1 to 15 the nodes representing these numbers are inserted

Algorithm 7 adjustOrder$((g_1, \ldots, g_n), (R_1, \ldots, R_n))$ simultaneous.

```
sortmax := 1
σ* := initialGlobalOrder((g_1, ..., g_n))
σ' := initialLocalAdjustedOrder(σ*, (R_1, ..., R_n))
σ := initialSimultaneousOrder(σ*, σ', (R_1, ..., R_n))
repeat
  for i := 1 to n do
    σ̂_i := smoothSort(g_i, σ_i, R_i, sortmax)
    l_i := L (R_i, σ̂_i)
  end for
  dec(sortmax)
until ∀i : Δ(l_{i-1}, l_i) ≤ δ ∨ sortmax < 0
return (l_1, ..., l_n)
```

successively. In graph 16, node 1 is deleted and node 16 is inserted. In Figure 1a) the ad-hoc approach is shown: for each graph a new layout is computed by using a static layout algorithm. The mental map is poorly preserved as all nodes change their ranks and more than half of the nodes also change their order within the ranks. In Figure 1b) the predecessor dependent layout strategy with $\delta_R = 0$ and a small δ is shown: the mental map is well preserved. No node changes its rank, and the order within the ranks is stable as well. But the local layouts are worse as there are more edge crossings. In Figure 1c) the predecessor dependent layout strategy with $\delta_R = 2$ and a large δ is shown: the left graph is equal to that produced by the ad-hoc approach. But in the next graph, all nodes contained in the backbone do not change their rank. So it is a good compromise between preserving the mental map and achieving local layout quality.

Further examples, e.g. visualization of the evolution of call graphs, are available at `http://www.cs.uni-sb.de/~diehl/ganimation`.

6 Related Work

Most work on dynamic graph drawing [4] is related to the online problem, which means that only information about the previous graphs in a sequence is used for computing a layout. This includes work on hierarchical graph drawing [12], spring embedding [3], and certain kinds of directed graphs [5]. To the best of our knowledge, the only two approaches that consider all graphs in the sequence are TGRIP and Foresighted Layout. TGRIP [6, 10] is an extension of the spring embedder GRIP for large graphs. The basic idea is very intuitive: time is modeled by springs in the third dimension. To this end each graph of the sequence is layouted in a 2D plane. Nodes representing the same vertex in subsequent graphs are connected by additional springs, but each node can only move within the 2D plane to which it belongs. In contrast to Foresighted Layout, this approach does not allow using different mental map metrics, because the metrics is built into the heuristic for minimizing the forces.

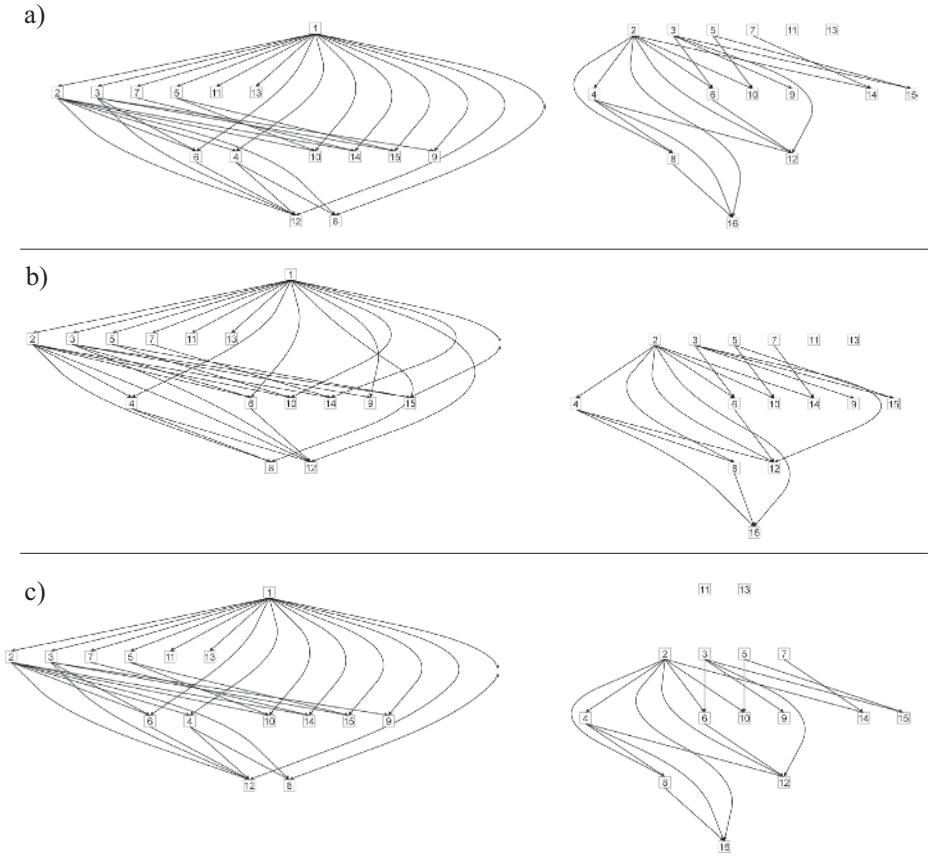

Fig. 1. Layouts of graphs 15 and 16 of evolving Hesse-graph using a) ad-hoc layout, b) FLT with small δ, $\delta_R = 0$ and c) FLT with large δ, $\delta_R = 2$.

7 Conclusions

While implementing FLT for spring embedding was relatively simple, applying the approach to orthogonal and hierarchical layout turned out to require many more changes to the static layout algorithms.

Phased Algorithms. Both algorithms work in phases, and we had to introduce new metrics which work on the results of these phases instead of on the final layouts. When the mental distance of two intermediate results exceeds a given threshold, then we restrict the search space either locally, i.e. for some nodes or edges, or globally, i.e. for all nodes or edges.

Global Restrictions. For spring embedding, the global temperature was reduced to allow fewer position changes of all nodes. Similarly, for hierarchical layout the variable sortmax influences all nodes in the sorting phase.

Local Restrictions. In the ranking phase of the hierarchical layout, we fix the rank of the not yet fixed node of highest importance. Thus, all remaining nodes can still change their ranks. For orthogonal layout the metrics, in fact, also gives a hint what to restrict. As a side-effect of computing the quasi-orthogonal-shape metrics, we do get a set of edges for which we can increment the α and β parameters of one or more of these edges, i.e. restrict the number of angle and bend changes.

Future Work. The theory and implementations of FLT are now at a stage such that we can start to apply them in different domains. The effectiveness of the resulting animations is currently being studied as part of a master thesis in psychology at the Catholic University Eichstätt.

Finally, work is underway to make force-directed, orthogonal and hierarchical FLT available as web services that produce animations in the SVG format.

References

1. O. Bastert and C. Matuszewski. Layered drawings of digraphs. In *Drawing Graphs [11]*. Springer, 2001.
2. U. Brandes, M. Eiglsperger, M. Kaufmann, and D. Wagner. Sketch-Driven Orthogonal Layout. In *Proc. of Graph Drawing 2002*. Springer LNCS 2528:1-11, 2002.
3. U. Brandes and D. Wagner. A Bayesian paradigm for dynamic graph layout. In *Proc. of Graph Drawing 1997*. Springer LNCS 1353:236-247, 1997.
4. J. Branke. Dynamic graph drawing. In *Drawing Graphs [11]*. Springer, 2001.
5. R.F. Cohen, G. Di Battista, R. Tamassia, and I.G. Tollis. Dynamic graph drawings: Trees, series-parallel digraphs, and *st*-digraphs. *SIAM Journal on Computing*, 24(5), 1995.
6. C. Collberg, S. G. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *Proc. of ACM Symposium on Software Visualization SOFTVIS'03*, San Diego, 2003. ACM SIGGRAPH.
7. S. Diehl and C. Görg. Graphs, They are Changing – Dynamic Graph Drawing for a Sequence of Graphs. In *Proceedings of Graph Drawing 2002*. Springer LNCS 2528:23-30, 2002.
8. S. Diehl, C. Görg, and A. Kerren. Preserving the Mental Map using Foresighted Layout. In *Proceedings of Joint Eurographics – IEEE TCVG Symposium on Visualization VisSym'01*. Springer Verlag, 2001.
9. M. Eiglsperger and M. Kaufmann. Fast Compaction for Orthogonal Drawings with Vertices of Prescribed Size. In *Proceedings of Graph Drawing 2001*. Springer LNCS 2265:124-138, 2002.
10. C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. Yee. GraphAEL: Graph Animations with Evolving Layouts. In *Proc. of Graph Drawing 2003*. Springer LNCS 2912:98-110, 2003.
11. M. Kaufmann and D. Wagner, editors. *Drawing Graphs – Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
12. S.C. North. Incremental Layout in DynaDAG. In *Proc. of Graph Drawing 1995*. Springer LNCS 1027:409-418, 1996.
13. U.Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In *Proceedings of Graph Drawing 1995*. Springer LNCS 1027:254-266, 1996.