Visual Navigation of Compound Graphs
Preprint

Marcus Raitner

University of Passau, D-94032 Passau, Germany,
`Marcus.Raitner@Uni-Passau.De`

Abstract. This paper describes a local update scheme for the algorithm of Sugiyama and Misue (IEEE Trans. on Systems, Man, and Cybernetics **21** (1991) 876–892) for drawing views of compound graphs. A view is an abstract representation of a compound graph; it is generated by contracting subgraphs into meta nodes. Starting with an initial view, the underlying compound graph is explored by repeatedly expanding or contracting meta nodes. The novelty is a totally local update scheme of the algorithm of Sugiyama and Misue. It is more efficient than redrawing the graph entirely, because the expensive steps of the algorithm, e. g., level assignment or crossing minimization, are restricted to the modified part of the compound graph. Also, the locality of the updates preserves the user's mental map: nodes not affected by the expand or contract operation keep their levels and their relative order; expanded edges take the same course as the corresponding contracted edge.

1 Introduction

A well-established technique to deal with huge graphs is to partition them recursively into a hierarchy of subgraphs; this leads to *compound graphs* [1] or *clustered graphs* [2]. Within both models, abstract representations of the graph, so-called *views*, can be defined [3, 4]. Intuitively, a view is generated by contracting subgraphs not needed in detail into *meta nodes*. Edges from within the contracted subgraph to nodes outside become edges from the meta node to the outside node. Views can be used for interactively exploring a large graph: one can choose which subgraphs to *contract* into meta nodes and which to *expand*. To this end, it is important that the drawing of the current view can be adjusted efficiently after these operations such that the user's mental map [5] is preserved. In this paper, an update scheme for the compound graph drawing algorithm of Sugiyama and Misue [1] is presented. Particular emphasis is laid on the locality of the updates: every expand and contract has only a local effect. The drawing of the new graph is computed only on the manipulated subgraph. This is a significant improvement for the time consuming phases of the algorithm, such as level assignment or crossing minimization. The user's mental map of the old view is preserved by keeping all uninvolved nodes on their levels and in the same relative order. Furthermore, expanded edges take the same course as the corresponding contracted edge.

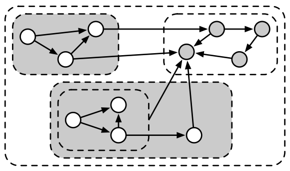

Fig. 1. A compound digraph; the hierarchy tree is depicted by the inclusion of the dashed rectangles

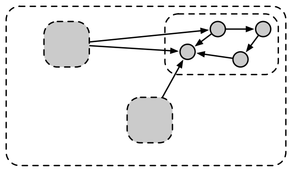

Fig. 2. The view having as leaves the darker shaded nodes of the compound digraph in Fig. 1

1.1 Problem Description

A *compound digraph* $D = (V, E, F)$ consists of nodes V, *inclusion edges* E, and *adjacency edges* F. It is required that the *inclusion digraph* $T = (V, E)$ is a rooted tree, and no adjacency edge connects a node to one of its descendants or ancestors; see Fig. 1. For convenience, we adopt the terminology of [1]: for a node $v \in V$, let $\mathrm{Ch}(v)$ denote the set of all *children* of v and $\mathrm{Pa}(v)$ the *parent* of v in T. The *descendants* of v, $\mathrm{De}(v)$, are all nodes in the subtree rooted at v (including v). The *depth* of v, $\mathrm{depth}(v)$, is the number of nodes on the path from the root of T to v. A *view* of D is a compound digraph $D[U] = (U, E[U], F[U])$ given by the nodes $U \subseteq V$, such that $T[U] = (U, E[U])$ with $E[U] = \{(u, v) \in E \mid u, v \in U\}$ is connected and contains the root of T; additionally, the leaves of $T[U]$ must *cover* the leaves of T, i. e., for each leaf u of T, (exactly) one of its ancestors is a leaf in $T[U]$; thus, $T[U]$ is a subtree of T from the root. The adjacency edges $F[U]$ comprise all edges $(u, v) \in F$ with $u, v \in U$ as well as *induced edges*: two *leaves* $u, v \in U$ are connected by an induced edge if and only if there are nodes $u' \in \mathrm{De}(u)$ and $v' \in \mathrm{De}(v)$ such that u' and v' are connected by an adjacency edge $(u', v') \in F$. Intuitively, given the designated set of leaves of $T[U]$, a view is constructed by shrinking each such leaf and all its descendants in T into a single meta node; see Fig. 2.

For the visual navigation of the underlying compound digraph D, the following operations shall be performed on a view $D[U]$:

- `expand`(v), where v is a leaf in $T[U]$; refines the view at v, i. e., the result is the view $D[U']$ with $U' = U \cup \mathrm{Ch}(v)$,
- `contract`(v), where $\mathrm{Ch}(v)$ are leaves in $T[U]$; coarsens the view at v, i. e., the result is the view $D[U']$ with $U' = U \setminus \mathrm{Ch}(v)$.

This paper concentrates on *visualizing* the above operations. We start with an initial layout of some view $D[U]$, and the user iteratively applies `expand` or `contract` operations; after each operation the previous layout has to be adjusted. Clearly, the obvious solution is redrawing the complete graph; unfortunately, it is neither efficient nor will it preserve the users mental map [5], which can be seen by comparing Figs. 3 and 5. In this context, our notion of preserving the mental map is that all old nodes U stay on their levels with their relative order

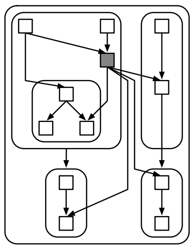
Fig. 3. Before expanding the shaded node

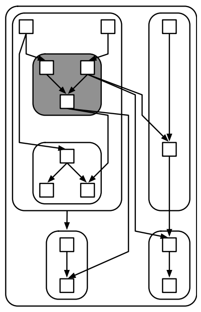
Fig. 4. Result of our update scheme

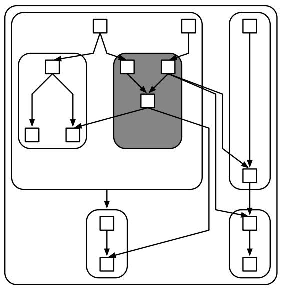
Fig. 5. Result of redrawing

unchanged; furthermore, expanded edges should take the same course as the corresponding contracted edge.

Our solution is an update scheme for the algorithm of Sugiyama and Misue [1] for drawing compound graphs. This algorithm, briefly recalled in Sect. 2, consists of four steps, which produce intermediate results. The initial view is laid out with the original algorithm; after each `expand` or `contract` operation, the intermediate results of the previous run are adjusted with our update scheme, as described in Sect. 3.

1.2 Related Work

There are several algorithms for drawing compound graphs, most notably the one of Sander [6], which differs from the algorithm of Sugiyama and Misue [1], among other things, by its *global layering*. While producing more compact – and supposedly more pleasant layouts –, this global layering is more difficult to update than the *local layering* of Sugiyama and Misue.

Dynamic or online graph drawing concentrates on updating layouts of ordinary graphs subject to insertions and deletions of nodes and edges [7]. The client-server model for online hierarchical graph drawing of North and Woodhull [8] allows insertion and removal of subgraphs, but only for ordinary DAGs and not for compound graphs. By using a *clan-based hierarchical decomposition*, the incremental drawing approach for DAGs of Shieh and McCreary [9] restricts the adjustments of the layout to the modified part. Visual navigation of compound (or clustered) graphs by expanding and contracting nodes has been introduced by Sugiyama and Misue [1], but they seem to implement them through reapplying their algorithm. Huang and Eades [10] briefly describe a system for handling huge clustered graphs visually, but their layout method is force-directed.

The visualization system of Abello and Korn [11] supports the interactive exploration of very large clustered graphs. They also use views as an abstraction of the underlying graph and provide methods for expanding *edges*. This expansion, however, restricts the view to the expanded edge, whereas our method preserves the relations of the expanded part to the remainder.

An efficient data structure supporting `expand` and `contract`, also known as *graph view maintenance problem* [3, 4, 12], is indispensable for an efficient implementation of the update method proposed in this paper. To this end, the software architecture of [13] should be considered as well.

2 Static Compound Graph Drawing

We recall the algorithm of Sugiyama and Misue [1] briefly, because our update scheme uses the intermediate results of these steps.

2.1 Step I: Hierarchization

Input of this step is the original compound graph $D = (V, E, F)$; the result is the *assigned compound graph* $D_A = (V, E, F_A, \mathrm{clev})$, where $\mathrm{clev} : V \to \mathbb{N}^+$ maps each node to its *compound level* and F_A are the adjacency edges F oriented from lower to higher level. This step internally uses the *derived compound graph* $D_D = (V, E, F_D, \mathrm{type})$; the edges F_D with their types $\mathrm{type} : F_D \to \{<, \leq\}$ are derived from F by replacing every adjacency edge $(u, v) \in F$ with edges between those ancestors of u and v having equal depth. The deepest such edge is of type $<$; all others of type $\leq$; see Fig. 6.

After resolving cycles in D_D, compound levels are assigned to the resulting cycle-free graph $D_F = (V, E, F_F, \mathrm{type})$. The root is placed on level (1); then children of already placed nodes are treated recursively. Note that the children of all nodes on the same level are always evaluated in a common recursive call. The local level of the children is determined by a standard level assignment algorithm that takes into account the two types of edges: $\mathrm{type}(u, v) = <$ enforces that the level of u is *less than* the level of v; with type $\leq$ the levels may also be *equal*. The compound level of a child v, $\mathrm{clev}(v)$, is built by appending its local level to its parent's compound level, $\mathrm{clev}(\mathrm{Pa}(v))$. Finally, the adjacency edges F are oriented from lower to higher level. Let the complexity of this step be $\mathcal{O}(\mathcal{S}_{\mathrm{I}}(n))$, where n is the size of the input D.

2.2 Step II: Normalization

In this step all adjacency edges of the assigned compound digraph D_A are made *proper*: an edge (u, v) is proper if $\mathrm{clev}(\mathrm{Pa}(u)) = \mathrm{clev}(\mathrm{Pa}(v))$ and $\mathrm{tail}(\mathrm{clev}(u)) = \mathrm{tail}(\mathrm{clev}(v)) + 1$, i. e., the parents lie on the same level and the children's levels differ by one. This is achieved by replacing each improper edge (u, v) with a linear compound graph as in Fig. 7. The result of this step is the *assigned proper compound graph* $D_P = (V_P, E_P, F_P, \mathrm{clev})$. Let the complexity of this step be $\mathcal{O}(\mathcal{S}_{\mathrm{II}}(n))$, where n is the size of the input D_A.

2.3 Step III: Vertex Ordering

Given the assigned proper compound graph D_P, this step calculates the relative order of the nodes on each level, so as to minimize edge crossings. To this end,

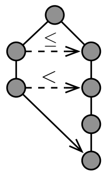

Fig. 6. The two dashed edges are derived from the solid one

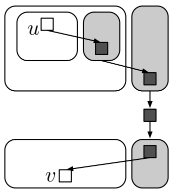

Fig. 7. Dummy nodes replace the improper edge (u, v)

the local order of all inner nodes' children on their levels is determined. The result of this step is the *ordered compound graph* $D_O = (V_P, E_P, F_P, \text{clev}, \sigma)$, where for each inner node $u \in V_P$, σ describes the local order of u's children.

The vertex ordering algorithm works depth-first: at an inner node u the compound graph induced by $\text{De}(u)$ is reduced to an ordinary DAG, the *local hierarchy*, by shrinking each child of u into a single node. This leads to two types of edges in the local hierarchy: edges between adjacent levels and those connecting nodes on the same level. A child u' of u or a descendant of u' may be adjacent to a node $v \notin \text{De}(u)$. By the definition of a proper edge, it follows that u and $\text{Pa}(v)$ lie on the same level. Since the algorithm has already ordered the children of all ancestors of u, it is known whether v lies to the left or to the right of u. Therefore, each child u' is annotated with two values $\lambda(u')$ and $\rho(u')$ counting the edges going to the left and to the right, respectively.

The crossing minimization for the local hierarchy starts with a preprocessing step – the so-called *splitting-method* – pinning the children u' with $\lambda(u') - \rho(u') > 0$ to the left end and those with $\lambda(u') - \rho(u') < 0$ to the right end of their level; the larger $|\lambda(u') - \rho(u')|$ is, the nearer to the end the node is placed. For the remaining nodes the crossings are minimized with a modified bary center heuristic that takes into account the horizontal edges. Let the complexity of this step be $\mathcal{O}(\mathcal{S}_{\text{III}}(n))$, where n is the size of the input D_P.

2.4 Step IV: Metric Layout

This step assigns coordinates and dimensions to the nodes of the ordered compound graph D_O. A recursive algorithm assigns local coordinates to the children relative to their parents position; for an inner node u it is applied to all children of u first, thus determining their width and height. The local coordinates are optimized with the so-called *priority method* on the *metrical local hierarchy*, which is basically the local hierarchy from the previous step without the horizontal edges. Similar to the bary center heuristic for minimizing crossings, the priority method improves the nodes positions by moving them – as far as possible without changing the order on the level – to their respective (metrical) bary centers. A final depth-first traversal calculates the absolute coordinates. The first phase is done with a recursive algorithm. Let the complexity of this step be $\mathcal{O}(\mathcal{S}_{\text{IV}}(n))$, where n is the size of the input D_O.

3 Update Scheme

Let $D[U] = (U, E[U], F[U])$ be a view of a compound graph $D = (V, E, F)$, where $D[U]$ already has been drawn with the standard algorithm; node $v \in U$ shall be expanded, resulting in a new view $D[U'] = (U', E[U'], F[U'])$, with $U' = U \cup \mathrm{Ch}(v)$. It is assumed that, given $D[U]$, the *structure* of $D[U']$ can be determined efficiently, which is where the data structures for maintaining hierarchical graph views [3, 4, 12] come into play.

3.1 Step I: Hierarchization

In this step the assigned compound digraph has to be updated. In order to preserve the mental map, all old nodes $u \in U$ stay on their levels; only for the children of v appropriate levels are determined. In other words, the level assignment function $\mathrm{clev} : U \to \mathbb{N}^+$ has to be extended to the set U'. Expanding v in the cycle-free graph $D_F[U]$ inherits the direction of edges incident to v to all corresponding expanded edges, i. e., if an edge was reversed during the cycle removal of the previous run, all corresponding expanded edges are reversed as well. In this setting, it is obvious that new cycles entirely consist of newly added edges, and the cycle removal can be restricted to the children of v.

From the definition of the derived graph, it follows that all edges adjacent to a leaf are of type $<$, and in the level assignment algorithm an edge of type $<$ causes the target node to be placed on a higher level than the source node. Therefore, v is not connected to any node on its level; neither is any child of v in the updated cycle-free graph $D_F[U']$. Consequently, the level assignment does not need to take into account children of nodes on the same level as v, but can be restricted to the subgraph of $D_F[U']$ induced by the children of v. After the level assignment clev has been extended to U', updating the assigned compound digraph $D_A[U]$ to $D_A[U']$ is just a matter of adding $\mathrm{Ch}(v)$ (and the corresponding inclusion edges) and inserting the new adjacency edges directed from lower to higher levels; induced adjacency edges incident with v are removed.

How much does the updated assigned compound digraph differ from the one the hierarchization algorithm of Sect. 2.1 applied to $D[U']$ would have produced? Since all old nodes stay on their level, it is not possible to place ancestors of v's neighbors on the same level as v. Compare, for instance, Fig. 9, which shows the level assignment produced by our update scheme, and Fig. 10, which would be the result of the hierarchization algorithm applied anew. The adjacency edge (u, v) in Fig. 8 leads to a type $<$ derived edge $(\mathrm{Pa}(u), v)$, which results in $\mathrm{Pa}(u)$ being placed on a level above v, where it is bound to stay during our update. If the derived graph would be built anew, this edge were of type $\leq$, and $\mathrm{Pa}(u)$ and v were placed on the same level, as shown in Fig. 10.

Property 1. *Let k denote the number of elements added to $D[U]$ by expanding v. The complexity of updating the assigned compound digraph $D_A[U]$ is $\mathcal{O}(\mathcal{S}_{\mathrm{I}}(k))$, compared to $\mathcal{O}(\mathcal{S}_{\mathrm{I}}(n+k))$ for reapplying step I to $D[U']$, where n is the size of $D[U]$. The user's mental map is supported by keeping all old nodes U on their level.*

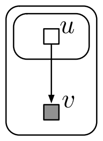

Fig. 8. Before expanding v

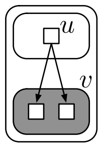

Fig. 9. Update of the derived graph

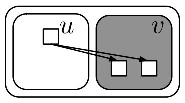

Fig. 10. Building the derived graph anew

3.2 Step II: Normalization

The assigned proper compound graph $D_P[U]$ is updated in two steps: first the node v is expanded and then the new improper adjacency edges are made proper. Expanding v means to add $\mathrm{Ch}(v)$ with appropriate inclusion edges, (induced) adjacency edges between two children of v, and those between a child of v and some other node $u \notin \mathrm{Ch}(v)$. For the latter the old induced edge connecting v and u has to be removed; if this edge has been made proper all the associated dummy nodes and edges are removed as well. Since the levels of the old nodes U are unchanged, the only improper edges are adjacent to at least one child of v. They are made proper exactly as described in Sect. 2.2. In the worst case this has to be done for every *new* adjacency edge, whereas for the other adjacency edges the construction from the previous proper compound digraph $D_P[U]$ is reused. Clearly, the result is exactly the same as if the normalization had been applied to the updated assigned compound digraph $D_A[U']$ as a whole.

Property 2. *Let k denote the number of elements added to $D_A[U]$ by expanding v. The complexity of updating the proper, assigned compound digraph $D_P[U]$ is $\mathcal{O}(\mathcal{S}_{\mathrm{II}}(k))$, compared to $\mathcal{O}(\mathcal{S}_{\mathrm{II}}(n+k))$ for reapplying step II to $D_A[U']$, where n is the size of $D_A[U]$.*

3.3 Step III: Vertex Ordering

In this step preserving the mental map means to keep the order of nodes that are common in the old and the new graph: only for the nodes that have been added during the update from $D_P[U]$ to $D_P[U']$ a position in the relative order on the respective levels has to be determined. These nodes are either children of v – including dummy nodes for edges between two children of v – or dummy nodes that belong to an edge between a child of v and a node $u \notin \mathrm{Ch}(v)$. As described in Sect. 2.3, the vertex ordering algorithm recursively calculates local orders for the children of an inner node on their levels; hence, determining the order of the children of v is just a matter of applying the algorithm to the subtree rooted at v. A precondition, however, is that the children of all ancestors of v already have been ordered. Therefore, the positions of dummy nodes that are not children of v have to be fixed prior to ordering the children of v.

Consider an induced edge $(v, u) \in F[U]$ (and symmetrically for an edge (u, v)); after expanding v, children $v_1, \ldots, v_k$ inherit this edge. Clearly, the subgraph inserted to make any of the (v_i, u) proper is – except for an extra dummy

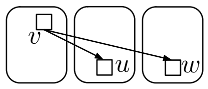

Fig. 11. Two proper non-local edges (v, u) and (v, w) before expanding v

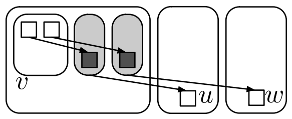

Fig. 12. After expanding v, the two new dummy node complexes inherit the order of u and w

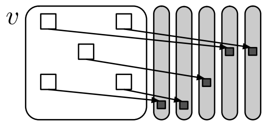

Fig. 13. The order of the dummy nodes is determined by the order of v's children

node complex on the level of v – identical to the one for edge (v, u). The idea is to reuse the positions of dummy nodes of the edge (v, u) for the dummy nodes of the edges $(v_1, u), \ldots, (v_k, u)$. The dummy nodes of these edges are treated as one block; the position of the block is the position of the respective dummy node of the edge (v, u). This reusing has the effect that the expanded edges take the *same course* as the old edge.

Since the positions of the old dummy nodes are reused, it can be assumed without loss of generality that the edge (v, u) is proper; if it is not, it suffices to expand the edge from v to the first dummy node of (v, u), which is proper. Expanding the proper edge (v, u) results in improper edges $(v_1, u), \ldots, (v_k, u)$; see Figs. 11 and 12. Each edge (v_i, u) is made proper with a dummy node complex consisting of nodes p_i and c_i with $\mathrm{Pa}(c_i) = p_i$ and edges (v_i, c_i) and (p_i, u); the nodes p_i are siblings of v and lie on the same level as v. We distinguish two types of proper edges (v, u): a *local* edge has $\mathrm{Pa}(v) = \mathrm{Pa}(u)$ and a *non-local* edge $\mathrm{Pa}(v) \neq \mathrm{Pa}(u)$. The reason is that if $\mathrm{Pa}(v) \neq \mathrm{Pa}(u)$, the definition of proper demands that $\mathrm{Pa}(v)$ and $\mathrm{Pa}(u)$ lie on the same level. Since the relative position of old nodes must be preserved, it is known whether $\mathrm{Pa}(u)$ is to the left or to the right of $\mathrm{Pa}(v)$. Clearly, this determines whether the dummy nodes p_i are to the left or to the right of v. For a local edge, the position of the new dummy node can be *anywhere* on the level of v.

All dummy nodes p_i belonging to the same expanded edge are treated as one block; therefore, one representant p is sufficient. In the local hierarchy induced by $\mathrm{Pa}(v)$'s children, a representant p for expanded edges belonging to a non-local edge (v, u) has $\lambda(p) - \rho(p) = \pm 1$, depending on whether $\mathrm{Pa}(u)$ lies to the left $(+1)$ or to the right (-1) of $\mathrm{Pa}(v)$. The splitting method (cf. Sect. 2.3) puts p to the left or right end of the level, with the exact position determined by the $\lambda(p) - \rho(p)$ value. Let q be the representant for the expanded edges of another non-local edge (v, w) such that $\lambda(p) - \rho(p) = \lambda(q) - \rho(q)$. Then p and q are indistinguishable in the splitting method; they are pinned to one end in *arbitrary* relative order. This order, however, should be the same as for the nodes u and w, which have both the same level $\mathrm{clev}(v) + 1$. This problem, incidentally, is immanent to the original algorithm of Sugiyama and Misue [1]. It can be solved by taking the relative order of the end nodes of the expanded edges as secondary sorting criterion in the splitting method. For our update scheme this means that a representant p is placed into the old order σ according to $\lambda(p) - \rho(p)$, and if there are more nodes with the same value, the order of the respective end nodes determines their order. Consider, for instance, the two dummy nodes on v's level in Fig. 12; having value -1 they are all placed to the right end. The order derived from the order of u and w shown in Fig. 12 clearly is the best choice.

After the splitting method, all representants for expanded edges of non-local edges are fixed; it remains to do the same for local edges. If (v, u) and (v, w) are two proper local edges, then u and w lie on the same level and thus determine the relative order of the representants p and q of the expanded edges for (v, u) and (v, w) respectively. Essentially, the only degree of freedom is whether to place p or q right or left of v. It makes no sense to have some non-dummy node x between p and v: otherwise the edges that p represents would cross x. In the local hierarchy induced by $\mathrm{Pa}(v)$'s children, dummy nodes like p and q have only one outgoing edge; hence, their bary center is identical to the position of u and w. The bary centers are used to decide whether representants are placed left or right of v.

It remains to determine the relative order of the expanded edges *within* their respective block. Since this order depends on the positions of v's children, the crossing reduction algorithm of Sect. 2.3 is applied to the local hierarchy of v first. Note that this is possible without knowing the exact order of the expanded edges: the representant already determines on which side they leave v; this information is sufficient for the λ and ρ values of v's children. Consider the edge (v, u) with its expanded edges $(v_1, u), \ldots, (v_k, u)$. The order of the dummy nodes $p_1, \ldots, p_k$ within the block represented by p is determined as follows: if p lies to the right of v, and if $v_{\sigma(1)}, \ldots, v_{\sigma(k)}$ is the order of v's children from bottom to top and within the same level from left to right, then $p_{\sigma(1)}, \ldots, p_{\sigma(k)}$ is the order of the dummy nodes from left to right; see Fig. 13. The case that p is right to v as well as the two cases for an incoming edge (u, v) are symmetric.

Property 3. *Let k denote the number of elements added to $D_P[U]$ by expanding v. Then the complexity of updating the local order σ is $\mathcal{O}(\mathcal{S}_{\mathrm{III}}(k))$, compared to $\mathcal{O}(\mathcal{S}_{\mathrm{III}}(n+k))$ for reapplying step III to $D_P[U']$, where n is the size of $D_P[U]$. The relative order of all old nodes U is preserved and expanded edges take the same course as the corresponding contracted edge.*

3.4 Step IV: Metric Layout

Expanding v changes the width and height of v, which leads to adjustments of the local coordinates for v's siblings; this, in turn, changes the width and height of $\mathrm{Pa}(v)$, and so on up to the root. On the other hand, the local coordinates of children of a node that is no ancestor of v are not affected. As described in Sect. 2.4, the metric layout consists of two steps: computing local coordinates followed by a traversal of the hierarchy to sum them up to absolute coordinates. Hence, we adjust the local coordinates at v and all its ancestors and then use the second step unalteredly. For the updates of the local coordinates basically the same recursive procedure as in Sect. 2.4 is used; the only difference is that recursive calls are made only for ancestors of v. The local coordinates in the subtrees rooted at nodes that are no ancestors are reused from the previous layout.

Property 4. *Let n denote the size of $D_O[U']$. In the worst case, the complexity of updating the coordinates is $\mathcal{S}_{\mathrm{IV}}(n)$. The final depth-first traversal to sum up the absolute coordinates is completely applied in any case; the local coordinates are adjusted only for ancestors of the expanded node.*

4 Contraction

Contracting a node v that has been expanded with the above update scheme is straightforward: in the contracted view $D[U']$ all nodes are old, i. e., $U' \subseteq U$; hence, the level assignment clev and the vertex order σ just need to be restricted to U'. The position of the dummy nodes for a new induced edge incident to v is given by the position of the blocks of the corresponding expanded edges. Since the width and height of v has changed, the metric layout has to be updated as described in Sect. 3.4. This has the side-effect that expanding and contracting are also *visually inverse*, i. e., the drawing after expanding and contracting a node v is the same as before expanding.

Why is contraction more complicated for nodes v that have not been expanded with our update scheme? Consider a child v' of v with an edge (u, v') such that v and $\mathrm{Pa}(u)$ lie on the same level, e. g., as in Fig. 10. As pointed out in Sect. 3.1, this cannot happen if v has been expanded before, yet it is possible in the layout of the initial view. Note that because of the deepest derived edge being of type $<$, for each edge (x, y) the compound levels $\mathrm{clev}(x)$ and $\mathrm{clev}(y)$ differ – after a common start sequence – by at least one position (cf. Sect. 2.1). The induced edge (u, v), representing (u, v') after contracting v, would violate this invariant, because $\mathrm{clev}(v)$ would be a subsequence of $\mathrm{clev}(u)$. This problem manifests itself in the derived graph: before contracting v the deepest edge, the one of type $<$, was adjacent to v' and is removed; therefore, the type of the derived edge between v and the ancestor of u at the same depth as v would have to be adjusted from $\leq$ to $<$, which could lead to substantial changes in the level assignment and thus to the user's mental map; see Fig. 9.

The easiest way to deal with this problem is to allow contraction only for nodes that have been expanded before, i. e., no node of the initial view can be contracted. Another way is to modify the algorithm of Sugiyama and Misue [1] used for the initial view such that all edges in the derived graph are of type $<$. The consequence is that the initial layout is less compact, because nodes with descendants that are connected never lie on the same level.

Property 5. *Let k denote the number of elements removed from $D_P[U]$; then the elements removed from the other (intermediate) compounds graphs are at most k. Updating the drawing after contracting a node v that has been expanded with our update scheme takes $\mathcal{O}(k)$ for steps I to III. Step IV is the same as after expanding; see Property 4.*

5 Summary

The proposed update scheme for the algorithm of Sugiyama and Misue [1] supports efficient visual navigation of compound graphs through expand and contract operations. The locality of our update scheme makes it much more efficient than redrawing the entire new view. For expanding a node the complexity of updating the drawing essentially is determined by applying each of the steps I to III to the *modified part* of the compound graph, followed by step IV adjusting the coordinates. The user's mental map of the old view is preserved well: old nodes stay on their levels in the same relative order and expanded edges take the same course as the corresponding contracted edge.

References

1. Sugiyama, K., Misue, K.: Visualization of structural information: Automatic drawing of compound digraphs. IEEE Trans. on Systems, Man, and Cybernetics **21** (1991) 876–892
2. Eades, P., Feng, Q.W.: Multilevel visualization of clustered graphs. In: Proc. 4th GD. Vol. 1190 of LNCS. (1996) 101–112
3. Buchsbaum, A.L., Westbrook, J.R.: Maintaining hierarchical graph views. In: Proc. 11th SODA. (2000) 566–575
4. Raitner, M.: Dynamic tree cross products. In: Proc. 15th ISAAC. LNCS. (2004)
5. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. Journal of Visual Languages and Computing **6** (1995) 183–210
6. Sander, G.: Graph layout for applications in compiler construction. TCS **217** (1999) 175–214
7. Branke, J.: Dynamic graph drawing. In Kaufmann, M., Wagner, D., eds.: Drawing Graphs – Methods and Models. Vol. 2025 of LNCS. Springer (2001) 228–246
8. North, S.C., Woodhull, G.: Online hierarchical graph drawing. In: Proc. 9th GD. Vol. 2265 of LNCS. (2001) 232–246
9. Shieh, F.S., McCreary, C.L.: Clan-based incremental drawing. In: Proc. 8th GD. Vol. 1984 of LNCS. (2000) 384–395
10. Huang, M.L., Eades, P.: A fully animated interactive system for clustering and navigating huge graphs. In: Proc. 6th GD. Vol. 1547 of LNCS. (1998) 374–383
11. Abello, J., Korn, J.: MGV: A system for visualizing massive multigraphs. IEEE Trans. on Visualization and Computer Graphics **8** (2002) 21–38
12. Buchsbaum, A.L., Goodrich, M.T., Westbrook, J.R.: Range searching over tree cross products. In: Proc. 8th ESA. Vol. 1879 of LNCS. (2000) 120–131
13. Raitner, M.: HGV: A library for hierarchies, graphs, and views. In: Proc. 10th GD. Vol. 1528 of LNCS. (2002) 236–243