A LAYOUT ALGORITHM FOR THE VISUALIZATION OF MULTIPLE RELATIONS IN GRAPHS

Géraldine Bous
SAP Research Sophia Antipolis, Business Intelligence Practice, Av. Dr. Maurice Donat 805, 06250 Mougins, France

Keywords: Visualization Algorithms, Hierarchical Graph Drawing, Multiple Relations, Structural Constraints.

Abstract: Many recent applications involve data models that rely on heterogeneous graphs (multiple node and relation types). Drawing these graphs is more difficult than drawing standard graphs, as it is desirable to take into account the heterogeneity of graphs in the form of constraints, possibly stemming from user preferences, to compute the layout. In this paper we propose a method for hierarchical graph drawing that is based on *structural constraint modeling*. These constraints are combined with crossing minimization algorithms to yield the desired visual effect. Three types of constraints are considered and illustrated, giving special attention to the drawing of multiple relations for interactive graph visualization.

1 INTRODUCTION

Graph Drawing serves the purpose of determining a layout, i.e. a representation in the plane, such that a graph can be visualized and analyzed by a user. This involves, for example, making all components of a graph visible (vertices, edges, attribute values, labels, etc.), addressing crossings, etc. Although the discipline has a long and solid history (see, e.g., (Battista et al., 1999) for an overview), several recent applications have opened new research challenges in the area. A key example is the drawing of the *heterogeneous graphs* that arise in the modeling of systems, networks and knowledge structures - models that need a proper visual representation in order to be understood and analyzed by users.

In this paper, we address the problem of heterogeneous graph drawing on the basis of (user) constraints. We present a method that captures the heterogeneity of a graph by incorporating several types of 'visual constraints' directly into the graph. By visual constraints, we refer to requirements on the final layout of graph. For example, the user may specify constraints relative to the order of certain nodes, wish to visualize nodes clustered by type or simply require the simultaneous visualizations of multiple relations (i.e. multiple graphs) on a unique set of nodes. We focus on hierarchical graph drawing (Sugiyama et al., 1981) and model the constraints structurally, i.e. not using parametric approaches, but rather by incorporating the constraints into the graph itself.

The paper is structured as follows: we describe the problem and give a brief overview of related work in section 2. The adaptation of hierarchical graph drawing algorithms to incorporate structural constraints is discussed in section 3. Next, we address the drawing of graphs with multiple relations (section 4) and pursue with extensions of our approach to other types of visual constraints (section 5). We finally conclude in section 6.

2 PROBLEM DESCRIPTION & RELATED WORK

Traditionally, graph drawing algorithms are centered towards the drawing of *homogeneous graphs*, i.e. graphs in which all nodes and edges are alike. In other words, it is in general assumed that graphs are composed of abstract entities, i.e. vertices, that all are of the same type; similarly, the edges of the graph are simply assumed to be abstract connections (between the vertices) that share the same properties. Modern information and knowledge models are however frequently built on the basis of heterogeneous graphs. For example, semantic models involve different types of nodes and relations. Likewise, business oriented social networks can be designed to contain more than just connections between employees: it is possible to incorporate the hierarchical structure of the company, extra-hierarchical structures that model crossorganizational projects and teams, as well as other types of entities and relations like furniture (e.g., cars with relationships like 'drives' or 'maintains'), data transfer processes, budgets, etc. These last examples are actually only a few of the actual user requests we have encountered in practice in the context of software design for business social networks. The reader may refer to (Cuvelier et al., 2012) for more details.

2.1 Challenges in Drawing Multi-relational Graphs

The main challenge in visualizing multiple relationships, or, to be more general, in visualizing several graphs defined on a common set of vertices, is not the visualization of the graphs themselves. It is indeed possible to visualize the different graphs separately using state-of-the-art graph drawing algorithms (Battista et al., 1999; Herman et al., 2000, are excellent surveys). However, a visualization approach based on 'multiple views', each showing one relation at a time, makes it difficult to show the 'true' connectivity in the network and is hence impractical for analysis purposes. Hence, the challenge resides in deciding how multiple relations should be displayed, possibly simultaneously and/or offering the possibility for user interaction.

First, it must be decided whether *all* relationships should be visualized at the same time or whether it is better to approach the problem in an interactive manner, allowing the user to select a subset of relationships. We advocate the use of the second solution: indeed, not all users may want to visualize the same relationships, which calls for an interactive approach where the user selects what she wants to see. Also, the simultaneous representation of all relationships may lead to visually loaded graphs that are visually intractable and inappropriate for analysis.

The choice of an interactive approach to the visualization of multiple relationships may, in turn, be approached in several ways. The simplest solution to this problem actually consists in selecting a reference graph, e.g. a hierarchical structure, and to superpose the other graphs on demand. This approach requires the computation of the optimal layout for the hierarchical structure and, as the user selects or un-selects other relationships, the layout is updated by adding new edges (resp. removing them) without changing the position of the nodes. The main criticism of such an approach is that it is likely to lead to a large number of edge crossings, which are usually considered to be undesirable for the understanding and analysis of the graph.

A second solution would be to re-compute a new layout every time the user selects (or de-selects) a relationship. As long as the number of nodes is reasonable, it is indeed possible to optimize the layout for a specific subset of graphs in real-time. Although this approach would produce a much more appealing visual output, there is one important disadvantage: adding as little as one relation to a given (current) display may lead to a complete disruption of both node and edge locations. Indeed, what is optimal, e.g. in terms of edge crossings, for a given set of relationships, is not granted to be optimal for another one. Re-optimization is thus problematic in that it destroys the *mental map* of the user (Misue et al., 1995), which requires her to re-adapt to the new display. Intuitively, but also experimentally, it has been shown that this is not desirable (Purchase, 2000; Purchase et al., 2006; Saffrey and Purchase, 2008).

The question is thus whether it is possible to compute a layout that would simultaneously be optimal in the sense of edge-crossings *and* preserve the mental map of the user as she navigates from one subset of relationships to another. The two criteria may nonetheless be antagonistic, as minimizing the number of crossings is totally independent from – and a priori unrelated to – the idea of preserving the mental map (i.e. the vertex positions in two-dimensional space).

In this paper, we propose an approach designed to meet both criteria. The general idea is to pre-compute an optimal layout, in terms of crossings, for an aggregated graph containing all relationships. As the user selects (resp. un-selects) relationships, edges and nodes are just added (resp. removed), but their locations do not need to change (as the layout is optimal for all relationships).

2.2 Related Work

As the reader will see shortly, the work presented here is essentially that of visualizing several graphs on the same set of nodes at the same time. From this perspective, the topic is related to the drawing of compound graphs. A *compound graph* is a graph composed of a directed tree and one adjacency graph. The few contributions to the resolution of this problem generally focus on sub-cases where the adjacency graph only connects nodes not having common ancestors (Sugiyama and Misue, 1991; Bertault and Miller, 1999; Forster, 2002) and represent the inclusion nodes as rectangular containers. The work of (Raitner, 2004) is an extension to (Sugiyama and Misue, 1991) that addresses the immersive visualization of the tree structure. The work discussed in (Holten, 2006) uses a technique based on edge-bundling and illustrates the use of other container representations than rectangles (hierarchical drawings, treemaps, etc.). Our work distinguishes itself from the research topics mentioned above in two ways. First, in the topic it addresses: the modeling of 'visual constraints' for the visualization of graphs containing heterogeneous data. Second, in the algorithmic approach used here, which is based on structural constraints in combination with a hierarchical graph drawing algorithm.

The work of (Shen et al., 2006) addresses the visualization of social network graphs, where, like is the case in this paper, several relationships may exist between nodes. (Shen et al., 2006) use the description of these relationships, called 'ontology', to filter and cluster the original graph data into meaningful agglomerates; in contrast, our work addresses the visualization of the relationships themselves, ideally in such a way that the result is visually tractable. (Fekete et al., 2003) use a treemap-based approach to visualizing a hierarchy and set of additional edges or relationships, the latter being drawn 'on top' of the treemap. The problem addressed is thus similar, but the goal of the method is different: our approach seeks to organize the layout in such a way that requirements on it (i.e. the visual constraints) can be met.

In the forthcoming section, we review the algorithmic procedure for drawing hierarchical graphs. We explain how a generic structural constraint is modeled and embedded into a graph and at what stage this occurs in the algorithmic process.

3 HIERARCHICAL GRAPH DRAWING WITH STRUCTURAL CONSTRAINTS

In this section we describe a methodology to adapt traditional hierarchical graph drawing algorithms to the case of graphs that are to be drawn taking account visual constraints in view of computing a layout that enables the interactive visualization of multiple relations. The only requirement on the visual constraints is that they have to be modeled as graphs. More specifically, the approach we discuss here consists in aggregating the constraints and the hierarchical structure into a single graph that can be drawn with state-of-the-art algorithms. The key element to turning the structural constraints into the desired visual constraints is achieved thanks to 'design' of the structural constraints in combination with the crossing minimization step. In this section, we describe the approach in its general form; specific types of constraints are addressed in sections 4 and 5.

3.1 Problem Formulation and Notations

Let $H(V,E)$ be a hierarchy, i.e. a directed acyclic tree of vertices $v \in V$ and edges $(v,w) \in E$ with $E \subseteq V \times V$. In parallel, let there be I (directed or undirected) graphs $G_i(V_i,E_i)$, $i \in I$, with $V_i \cap V \neq \emptyset$ (i.e. V_i is not restricted to $V_i \subset V$, but both must have a least one node in common). The heuristic method we discuss here is based on four distinct steps. First, the layering step that determines the layer $y(v)$ at which every node $v \in H$ of the tree should be placed. Second, an aggregation phase to embed H and the constraints G_i into a *unique* directed tree H_G with vertices V_G and edges E_G (note that $V \subseteq V_G$). The third step determines the order $o(v)$ of the vertices in every layer of H_G with the purpose of minimizing the number of edge crossings. The final step refines the horizontal position $x(v)$ of every vertex for the screen layout. The reader should note that the steps one, two and four correspond to the steps of the Sugiyama algorithm (Sugiyama et al., 1981). It is also worth emphasizing that, although we here focus on trees, the Sugiyama algorithm – and hence the method presented here – can be extended to graphs in general; the reader may refer to (Battista et al., 1999) for details.

3.2 Layering

The first step of our approach computes the vertical position $y(v)$ of the vertices of the hierarchy $v \in H$. It is important to note that the layer computed at this level is maintained throughout the different steps of the algorithm: the layer of a node $v \in V$ hence is identical in the trees H and H_G.

Several methods can be used to assign a layer to the nodes of a directed tree (Battista et al., 1999); we use the shortest path method, that is, a method that assigns every node to the layer immediately below that of its closest predecessor. Let

$$\delta_H(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

If $\delta_H(i,j) = 1$, it means that the vertices i and j are connected by an edge directed towards j. Solving the following linear program provides the vertical coordinates of the nodes of the hierarchy H (note that layer 1, the top layer, is assigned to the root of the tree):

$$\begin{array}{lll} \min & \sum_{i \in V} y(i) & \\ \text{s.t.} & y(j) - y(i) \geq 1 & \text{if } \delta_H(i,j) = 1 \\ & y(i) \geq 1 & \forall i \in V. \end{array} \tag{2}$$

3.3 Graph Aggregation

The graph aggregation step concerns the embedding of the structural constraints into the hierarchy H. In general terms, the aggregation step consists in performing the union of the graphs, i.e.

$$H_G = H \bigcup_{i=1}^{I} G_i, \tag{3}$$

where H_G denotes the aggregated graph. The satisfaction of the visual constraints is attained by the embedding *in combination* with the crossing minimization step discussed below. More precisely, the constraints must be designed in such a way that they lead to crossings in H_G when they are not satisfied. The detailed modeling of different types of constraints and the corresponding aggregation procedures are discussed in sections 4 and 5.

3.4 Crossing Minimization

The crossing minimization problem of hierarchical graphs is generally approached by minimizing the crossings of all pairs of consecutive layers in the tree. This problem, known as the “two-layer crossing problem”, is NP-hard, which justifies the existence of several heuristic methods to solve it. In the context of constrained graph drawing, we seek to minimize not only the crossings in the hierarchy H, but also the crossings that would arise through the violation of the structural constraints. In other words, crossings are minimized for the set of vertices $\{v \in V_G \mid y(v) = k\}$ and $\{v \in V_G \mid y(v) = k+1\}$ for $k = 1, \ldots, y_{\max}(H_G) - 1$, where $y_{\max}$ denotes a function that returns the maximum depth of a tree. The goal of the two-layer crossing problem is to determine the order in which the vertices of the two layers should be placed as to minimize the number of crossings between the edges that connect them. Two well known and well documented techniques are the “median heuristic” and the “swap heuristic”, that exist in several variants and generally demand the use of dummy (temporary) vertices to lead to good results. The interested reader may refer to (Battista et al., 1999) and the references therein for a detailed description of these two, as well as other, heuristic methods.

3.5 Final Coordinate Assignment

Once the horizontal order of the vertices of H_G has been determined, two distinct steps must actually be performed. First, all dummy vertices introduced in previous steps must be removed. This includes those added to model structural constraints, as well as those added during the crossing minimization step. The final step consists in refining the horizontal coordinates of the remaining vertices (i.e. those of H), without changing their order, to obtain a more uniform and visually attractive result on the screen. Several methods exist to this purpose, some of which are documented in (Battista et al., 1999).

4 AGGREGATION FOR MULTI-RELATIONAL GRAPHS

In this section we describe the application of graph drawing with structural constraints to the case of multi-relational graphs, which we illustrate by the joint visualization of linear processes, in view of allowing an interactive visualization without disrupting the mental map of the user. The different relations are embedded into the hierarchical graph in order to compute a layout that is optimal, in terms of crossings, for any subset of relations the user may want to explore. We first discuss the modeling of the constraints as graphs, as well as the aggregation process (which was presented in a generic manner in section 3.3); next we provide some experimental results regarding the computational complexity of our approach.

4.1 Approach Description

Like previously, let $H(V,E)$ be a hierarchy and $G_i(V_i,E_i)$ be a series of I graphs, $i \in I$, with $V_i \subseteq V$. We consider the special case where every G_i models a linear process, that is, a sequence of vertices $(v_{i1}, v_{i2}, \ldots, v_{in_i})$, where n_i represents the length of process i. Each process is thus a separate graph - a relation - on different, not necessarily disjoint, subsets of V.

The method of structural constraints makes it possible to model proximity and adjacency constraints for nodes in H. This type of constraints cannot be taken into account in traditional hierarchical graph drawing and are necessary to visualize the process in both in a ‘linear way’ and to restrict each process to a well defined portion of the (visual representation of the) hierarchy. The constraints are designed to force the vicinity of vertices that are connected to each other and have been assigned the same layer in the y-coordinate determination step (section 3.2); in addition, this minimizes the number of edges that cross the layout from one side to another. For example, in the illustration of figure 1(a), superposing the linear process on the hierarchy implies both edge crossings and same-layer traversing edges (from vertex B to vertex E).

The aggregated graph H_G is built in I steps: starting with the hierarchy alone, each graph G_i is integrated sequentially. The constraints are modeled according to the pseudocode below:

```
for $k = 1$ to $n_i$ do
  if $y(v_{i,k}) \neq y(v_{i,k+1})$ then
    if $y(v_{i,k}) < y(v_{i,k+1})$ then
      $\delta_{H_G}(v_{i,k}, v_{i,k+1}) = 1$
    else
      $\delta_{H_G}(v_{i,k+1}, v_{i,k}) = 1$
    end if
  else
    add temporary vertex $v^t$ to $H_G$
    $y(v^t) = y(v_{i,k}) + 1$
    $\delta_{H_G}(v_{i,k}, v^t) = 1$
    $\delta_{H_G}(v_{i,k+1}, v^t) = 1$
  end if
end for
```

If two consecutive vertices in process G_i are not in the same layer, they are connected in H_G with an edge directed from the lower layer vertex to the higher level vertex (root-to-leaf direction), in order to avoid creating cycles. To the contrary, if two consecutive vertices in G_i have the same layer, a constraint is added in order to enforce a placement in which the two vertices will be direct neighbors. This constraint is modeled on the basis of a temporary vertex v^t that is connected to and placed below the two process vertices. Figure 1(b) illustrates this principle.

The application of the crossing minimization procedure on the aggregated graph H_G achieves the visual consistency for the joint visualization of the hierarchy and the linear processes. Indeed, the connection of process vertices thanks to temporary vertices and edges in the 'aggregated hierarchy' H_G forces the proximity of process vertices. Intuitively, it is clear that, the further away two process vertices are from each other (i.e. the more other vertices are placed between them), the greater the number of edge crossings will be. The principle is illustrated in figures 1(b) and 1(c), that, respectively, show the aggregated hierarchy before and after crossing minimization. The final result of this method, i.e. after removal of temporary vertices, is shown in figure 1(d).

When several relations exist on the same hierarchy, pre-computing the layout for all relations simultaneously with this technique allows determining a visual representation that is optimal for the aggregated graph, i.e. for all relations. It is hence easy to use an interactive approach where to user is allowed to select the relations that she wants to visualize by simply adding or hiding the corresponding edges without modifying the layer and order of the vertices, thereby maintaining the her mental map.

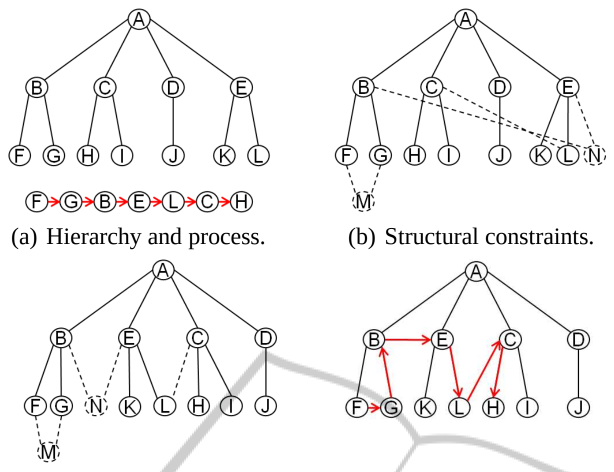
4.2 Experimental Results

The main drawbacks of hierarchical graph drawing algorithms are the computation time and, if heuristic methods are used to compensate for the computational complexity, the potential sub-optimality of the solutions also becomes a disadvantage. The computational complexity of our method is higher than that of drawing the base hierarchy alone, as the aggregated graph H_G contains a certain number of dummy vertices to model the structural constraints. It is thus of interest to determine the impact of the number of process nodes (and indirectly of the number of process dummy vertices) on the average computation time with the algorithm proposed in the previous section.

We have performed an experimental evaluation of the crossing minimization step of the algorithm, i.e. the step with highest computational complexity. The algorithm implements the median heuristic in Matlab. We evaluated the computation time for randomly generated trees of size n and one or more linear processes. More precisely, for every randomly generated tree, a total of lp nodes, where lp is a random number satisfying $2 \leq lp \leq n$, were assigned to one or more linear processes (we thus have $lp = \sum_i n_i$). We measure the average computation time for every pair (n, lp). Note that the actual number of nodes is larger than n, not only due to the structural constraint approach, but also due to the other dummy vertices that are needed in the algorithm (crossing minimization step).

We performed a total of 55700 tests with $5 \leq n \leq 230$ and $5 \leq lp \leq \min(n, x_{\max} \cdot y_{\max})$, where $x_{\max}$ and $y_{\max}$ are, respectively, the maximum width and depth of a tree. The scatter plot in figure 2(a) shows the CPU time (in seconds) as a function of the number of nodes n in the base tree. The color code used for the dots is a function of lp. More precisely, lp has been divided into six equal parts, with each color assigned to an interval: blue < cyan < green < yellow < magenta < red. The scatter plot in figure 2(b) shows the CPU time (in seconds) as a function of the number of process nodes lp. Again, we divide the outcomes in six categories, this time according to the size of the tree n, and use the same color code as above. The purpose of the color code is to show if there is a correlation between the CPU time and the variable that is not shown. That is, for figure 2(a) the color code shows the impact of lp on a plot that shows $(n, f(n))$; for 2(b), the color gives a hint of the impact of n on a plot that shows $(lp, f(lp))$.

An analysis of both figures leads to the following conclusion: the decisive factor in the computation time is n, the size of the base tree. Indeed, when showing $(n, f(n))$, we can see that lp has but little correla-



(a) Hierarchy and process. (b) Structural constraints. (c) Optimization. (d) Final result.

Figure 1: Example of a structural constraint to embed a linear process into a tree. The edge-crossing minimization step in combination with the structural constraint allows visualizing the process in a linear manner, without traversing edges.

tion since the 'color code' is randomly distributed in the cloud of data-points (note that a correlation is visible for lower values of n as $lp \leq n$). This is confirmed by the scatter plot $(lp, f(lp))$, which shows an almost perfectly horizontal cloud with layered colors. Hence, higher CPU time is correlated essentially to n and not to lp – which implies that the method of structural constraints does not have a significant negative impact on the computational complexity and on the performance of the algorithm.

In addition to the problem of computational complexity, we have tested the quality of the method here proposed with respect to the number of crossings. Both the hierarchies and linear processes used for the tests are randomly generated and approximately 5000 tests were run. The reference for the comparison is a tree whose layout is determined by the 'standard Sugiyama algorithm' (with the Swap Heuristic), but where no processing is done on the nodes of the linear processes. In other words, the edges of the linear processes are simply drawn 'on top' of the tree. The average number of crossings was then determined both as a function of n and lp. Figure 3(a) shows the ratio of the average number of crossings without processing to the average number of crossings with our method as function of n; similarly figure 3(b) shows the ratio as a function of lp. The first figure shows that the structural constraints method is particularly efficient for $n < 50$, where no processing doubles the number of crossings. In general, it appears that our method improves the result by approximately 25%. As for the second figure, as function of lp, the trend is more steady, with a 20% improvement in favor of our method.

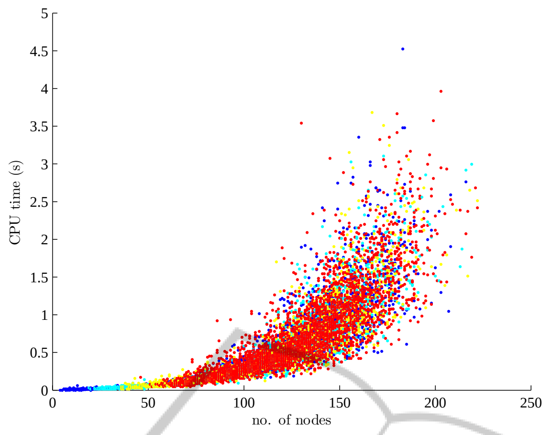
5 EXTENSION TO OTHER TYPES OF CONSTRAINTS

The technique of structural constraints described in section 3 can be easily extended to other types requirements than those of representing multiple relations (linear processes) in a visually consistent manner. In this section we illustrate how to perform the graph aggregation for two additional types of constraints on hierarchical graphs: first, the clustered visualization of multiple node types and, second, for order constraints on a subset of the nodes of the hierarchy.

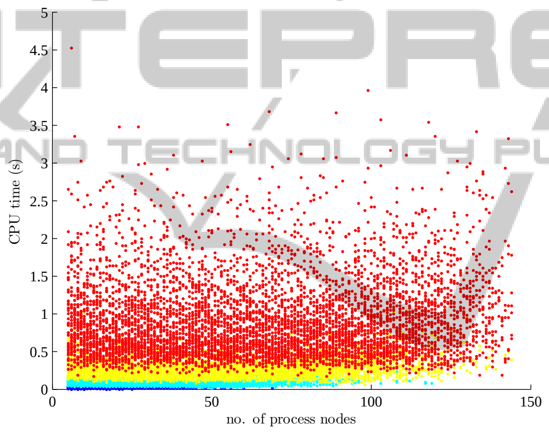
5.1 Visual Node Clustering in Hierarchical Graphs

For graph-based data structures – here assumed hierarchical – that contain different node types (e.g. persons, furniture, etc.), it may be desirable to draw the graph such that nodes of the same type are shown together, thereby forming clusters of identical nodes. Like for multi-relational graphs, this can be achieved by designing adequate structural constraints that are embedded into the tree structure, previous to the crossing minimization step.

As usual, let $H(V,E)$ denote the hierarchy; moreover, assume that every node in the hierarchy has a given type i, where $i = 1,\ldots,I$. In this case, we define the structural constraints as graphs $G_i(V_i,E_i)$, such that $V_i = \{v_i^t \cup (v \in V \mid type(v) = i)\}$ and $E_i = \{(v_i^t, v) \mid v \in V_i \setminus v_i^t\}$. In other terms, we define a constraint graph for every node type; the graph is com-



(a) CPU as a function of n.



(b) CPU as a function of lp.

Figure 2: Computational complexity of our method. The figures show that higher CPU time is correlated essentially to n and not to lp.

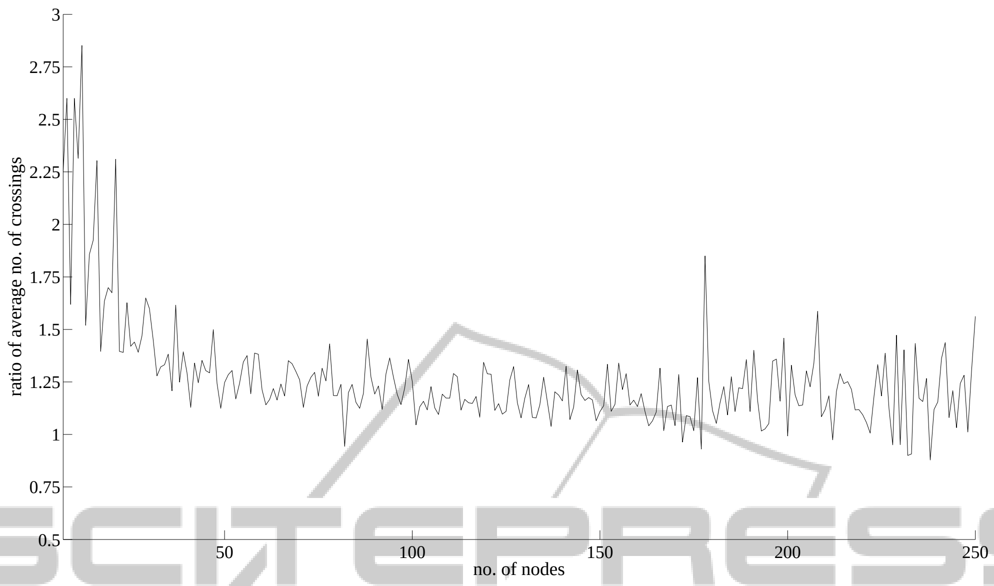
posed of a temporary vertex v_i^t that is connected to all nodes of H that have type i.

The constraint graphs are built and embedded sequentially into the hierarchy H, forming the aggregated hierarchy H_G. Initializing H_G as H, the process for the aggregation of one graph G_i is described below:

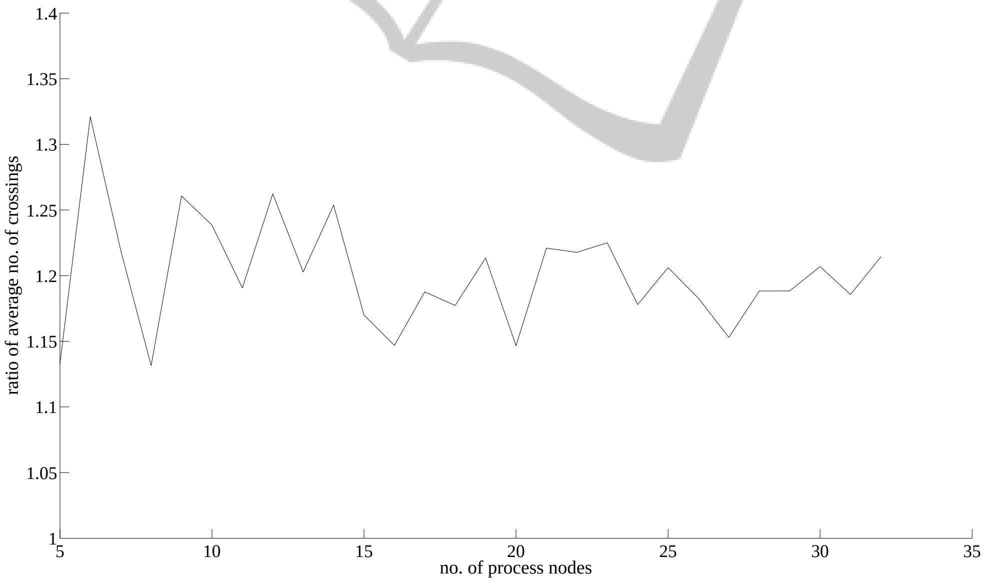
```
add temporary vertex $v_i^t$ to $H_G$
$y(v_i^t) = y_{\max}(H) + 1$
for all $v \in V \mid type(v) = i$ do
    $\delta_{H_G}(v, v_i^t) = 1$
end for
```

The temporary vertex, symbolizing the cluster for a given node type, is added on the fly as the constraint is embedded into the hierarchy. Temporary vertices are placed one layer below all nodes of H, i.e. at layer $y_{\max}(H) + 1$ (so that $y_{\max}(H_G) = y_{\max}(H) + 1$), and connected to all vertices of type i.

The 'clustering effect' is again obtained thanks to the combined use of structural constraints and a crossing minimization algorithm on the graph H_G. Indeed, mixing nodes of different types creates crossings on the (temporary) edges that bind the nodes to their corresponding 'cluster vertex'; the constraints and the crossing minimization thus implicitly induce horizontal regions that, whenever possible, contain nodes of one specific type. This is illustrated in figure 4, where we use the same base hierarchy as in figure 1 with nodes in three colors to symbolize their types (figure 4(a)). Figures 4(b) and 4(c) respectively show the aggregated hierarchy graph before and after the crossing minimization; the final result, after removal of dummy vertices and edges, is shown in figure 4(d).



(a) Crossing ration as function of n.



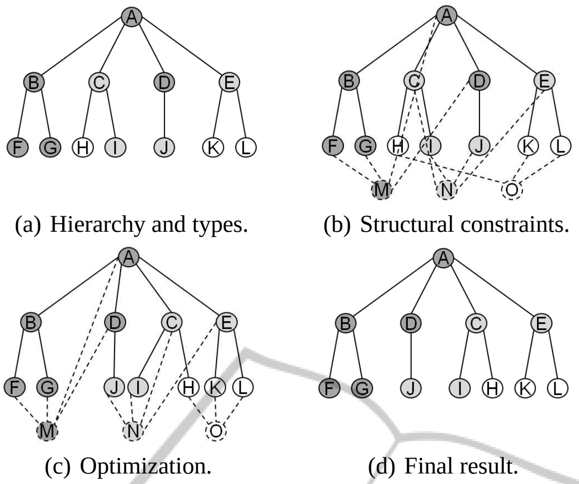
(b) Crossing ration as function of lp.

Figure 3: Ratio of the average number of crossings without processing to the average number of crossings with the structural constraints technique.

5.2 Vertex Order Constraints

The use of structural constraints also enables the end-user to specify order constraints on a subset of the vertices of the tree H. Again, the idea is to design the constraints in such a way that the application of the crossing minimization algorithm on the aggregated graph H_G forces the satisfaction of the constraints.

Let, as usual, $H(V,E)$ denote the hierarchy and assume that there exists a subset of vertices $V_o \subseteq V$ for which the (relative) order is defined, i.e. $o(v_1) < o(v_2) < \ldots < o(v_{|V_o|})$ for $v_i \in V_o$. If the aggregated



(a) Hierarchy and types. (b) Structural constraints. (c) Optimization. (d) Final result.

Figure 4: Example of a structural constraint to cluster the nodes of a tree according to their type.

hierarchy is initialized as $H_G = H$, then an order constraint can be embedded into the hierarchy with the following procedure:

```
for $i = 1$ to $|V_o|$ with $v_i \in V_o$ do
    add temporary vertex $v_i^t$ to $H_G$
    $y(v_i^t) = y_{\max}(H) + 1$
    $o(v_i^t) = i$
    $\delta_{H_G}(v_i, v_i^t) = 1$
    if $i > 1$ then
        $\delta_{H_G}(v_{i-1}, v_i^t) = 1$
    end if
end for
```

The process described therein is quite simple: for each ordered node v_i, one dummy vertex is added below the hierarchy (i.e. $y(v_i^t) = y_{\max}(H) + 1$) and is connected to v_i. Since the order of the dummy vertex is imposed ($o(v_i^t) = i$) it suffices to apply the crossing minimization *excluding* the dummy vertex layer, i.e. taking the edges into account but without reordering the nodes, to impose an order on the vertices of $v_i \in V_o$. Without the if-loop and the constraint added at line 7, the order of the nodes will be preserved by the crossing minimization algorithm, but may only be a *relative order*. That is, no constraints restrict other nodes ($v \in V \setminus V_o$) to be placed in between the nodes of V_o. If the end-user prefers a *strict order*, then it is necessary to add an additional constraint that connects a vertex $v_i \in V_o$ not only to its corresponding dummy vertex v_i^t, but also to that of its successor in the order relation.

An illustration for this case would be quite similar to the one we used for the linear process in figure 1; indeed, a linear process, as defined in section 4, is nothing but a strict linear order (see, e.g., (Roman, 2008) for an exhaustive definition of order relations).

In the case of order constraints, the modeling is however different: all temporary vertices are placed in the bottom layer so that the crossing minimization algorithm can be applied only to the layers of the original tree H.

To conclude this section, we emphasize that the use of structural constraints is not limited to the constraints we discuss here. For example, it is conceivable to combine ordering constraints with clustering constraints or constraints relative to multiple relations.

6 DISCUSSION AND FUTURE WORK

We have introduced the concept of structural constraints and shown how these can be used to model visual constraints for heterogeneous graphs. The constraints are designed as to be treated by crossing minimization algorithms. We explored three different cases, with their respective constraints, to which the method can be applied. We give particular focus to the visualization of multiple relations, showing how our technique can be used for the computation of layouts destined to interactive graph visualization without having a significant impact on the base computational complexity of algorithm, dominated by the crossing minimization step.

While it is certainly conceivable to design specific algorithms for each of the examples given here, the advantage of using structural constraints is precisely to offer the possibility to embed the constraints directly *into* the data. Consequently, it is possible to use generic state-of-the-art hierarchical graph drawing algorithms without further modifications. From the perspective of large scale software design, the use of generic, reusable and extensible modules are criteria of capital importance, which our method allows to satisfy.

Although the method presented here produces good results, further experiments are required to validate it for larger graphs, as well as to analyze its applicability when different types of constraints are combined together. The quality of the results is highly impacted by the quality of the output provided by the heuristic methods used in the crossing minimization step. This imposes further research on using this method in combination with other types of approaches and visualization techniques, like those presented in (Burch and Diehl, 2008; Burch et al., 2010) for the visualization of time-varying compound graphs. Interesting is also the work of (Shen et al., 2006), as well as the approach presented in (Muelder and Ma, 2008), which combines clustering techniques and tree-maps for fast layout computation of large graphs. Another important direction for future research is related to the non-discriminative application of crossing algorithms. The embedding of all relations and constraints into a unique graph makes it impossible to discriminate edge and node types (constraint, hierarchy, relation, etc.). When no zero-crossing solutions exist, it is impossible for current crossing minimization algorithms to distinguish between ‘favorable’ and ‘unwanted’ crossings. For example, the end-user may specify that she prefers crossings in the hierarchy rather than in the linear processes she visualizes. Hence, the design of algorithms that can prioritize crossings according to the edge and node types is an important next step to consider.

Heterogeneous graph drawing, in general, is a topic rich in challenges where many problems remain to be addressed. These range from the user experience domain (e.g. adequate layout criteria and ‘coherent’ visual representations), the management of zoom in/out approaches following several criteria, to the interactivity with respect to the visible subset of relations themselves.

ACKNOWLEDGEMENTS

The work presented here was developed in the context of the ARSA project (Analyse des Réseaux Sociaux pour Administrations), partially funded by the French DGCIS (Direction Générale de la Compétitivité, de l’Industrie et des Services). The author wishes to thank our end-user, the City Administration of Antibes, with a special mention to our contact, Patrick Duverger.

REFERENCES

Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. G. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Upper Saddle River.

Bertault, F. and Miller, M. (1999). An algorithm for drawing compound graphs. In *Proceedings of the 7th International Symposium on Graph Drawing (GD’99)*, pages 197–204.

Burch, M. and Diehl, S. (2008). Timeradartrees: Visualizing dynamic compound digraphs. In *Proceedings Eurographics/ IEEE-VGTC Symposium on Visualization 2008*, volume 27, pages 823–830.

Burch, M., Fritz, M., Beck, F., and Diehl, S. (2010). Timespidertrees: A novel visual metaphor for dynamic compound graphs. In Hundhausen, C. D., Pietriga, E., Daz, P., and Rosson, M. B., editors, *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing VL/HCC 2010*, pages 168–175.

Cuvelier, E., Bous, G., Aufaure, M.-A., and Kleser, G. (2012). ARSA: Analyse des Réseaux Sociaux pour les Administrations - une expérience d’intégration de réseaux sociaux internes et externes dans une administration. *Ingénierie des Systèmes d’Information*. To appear.

Fekete, J.-D., Wang, D., Dang, N., and Plaisant, C. (2003). Overlaying graph links on treemaps. In *IEEE Symposium on Information Visualization Conference Compendium (demonstration)*.

Forster, M. (2002). Applying crossing reduction strategies to layered compound graphs. In *Proceedings of the 10th International Symposium on Graph Drawing (GD’02)*, pages 276–284.

Herman, I., Melanon, G., and Marshall, S. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6:24–43.

Holten, D. (2006). Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12:741–748.

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

Muelder, C. and Ma, K.-L. (2008). A treemap based method for rapid layout of large graphs. In *Proceedings of Visualization Symposium, 2008. PacificVIS ’08. IEEE Pacific*, pages 231 –238.

Purchase, H. (2000). Effective information visualization: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13:147–162.

Purchase, H., Hoggan, E., and Grg, C. (2006). How important is the “mental map”? an empirical investigation of a dynamic graph layout algorithm. In *Proceedings of 14th International Symposium on Graph Drawing (GD’06)*, pages 184–195.

Raitner, M. (2004). Visual navigation of compound graphs. In *Proceedings of the 12th International Symposium on Graph Drawing (GD’04)*, pages 403–413.

Roman, S. (2008). *Lattices and Ordered Sets*. Springer, New York.

Saffrey, P. and Purchase, H. (2008). The “mental map” versus “static aesthetic” compromise in dynamic graphs: a user study. In *Proceedings of the 9th Conference on Australasian User Interface*, pages 85–93.

Shen, Z., Ma, K.-L., and Eliassi-Rad, T. (2006). Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE Transactions on Visualization and Computer Graphics*, 12:1427 – 1439.

Sugiyama, K. and Misue, K. (1991). Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21:876–892.

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