Preserving the mental map in interactive graph interfaces

Manuel Freire
EPS-Universidad Autónoma de Madrid
Av. Tomás y Valiente 11,
ES-28049 Madrid, Spain
manuel.freire@uam.es

Pilar Rodríguez
EPS-Universidad Autónoma de Madrid
Av. Tomás y Valiente 11,
ES-28049 Madrid, Spain
pilar.rodriguez@uam.es

ABSTRACT
Graphs provide good representations for many domains. Interactive graph-based interfaces are desirable to browse and edit data for these domains. However, as graphs increase in size, interactive interfaces risk information overload and low responsiveness. Focus+context approaches overcome these problems by presenting abridged views of the graph. Users can then navigate among views with a level-of-detail mechanism. If jumps from each view to the next are easy to follow, users will gain a good mental map of the whole graph; otherwise, they may become disoriented.

In this work, we identify three factors that affect mental map preservation during navigation of interactive focus+context graphs: the predictability of navigational actions, the degree of change from one view to the next, and the traceability of changes once they occur. Strategies for preserving user orientation are classified according to these factors, and new strategies developed for the CLOVER visualization environment are presented.

Categories and Subject Descriptors
H.5.2 [User Interfaces]: [GUI, interaction styles]; G.2.2 [Graph Theory]: [Graph algorithms]

General Terms
Algorithms, Design

Keywords
Focus+context, graph visualization

1. INTRODUCTION
Graphs are a convenient way of representing information for many domains, such as software programs, hypermedia structures, network topologies, and co-citation maps. In some applications, graphs have a strong structure which the representation should highlight. For instance, in software call graphs or citation networks, groups of highly interrelated nodes can be found - which communicate only weakly with other such groups. This “small world” [13] property can be used to alleviate the problems that result from naive representations of the full graph: large graphs require large amounts of time to be laid out, and moreover such layouts are of scarce use due to information overload.

The existence of highly interrelated nodes, or “node clusters”, can be used to provide better layouts, by first laying out the clusters themselves (as if they were single nodes) and then filling in their component nodes [11]. Clustering can also be used to provide abstraction, by not filling in nodes later, and using the abstracted version “as is” to provide an overview of the structure. Clustering can be carried on recursively, yielding “hierarchical clustering”. Hierarchical clustering can provide good insights into a graph’s structure. This makes it a good base for “semantical” focus+context (where uninteresting parts are abstracted), as compared to “geometrical” focus+context (where uninteresting parts are preserved, but receive less screen space).

While interacting with a graph, a user will gather and maintain an idea of the general structure of this graph and the general location and relations between parts of the whole: a “mental map” [4]. When the interface relies heavily on changing the viewpoint (as do all focus+context approaches), special care must be taken to prevent breaking the user’s current mental map. This is the focus of this paper.

In the next section, related work in the field of focus+context graphs is presented. Then, a classification of methods to deal with mental map preservation is proposed, using three guiding factors: predictability, degree of change and traceability; alternatives are discussed in relation to each. Section 4 then provides a practical example, discussing the implementation of the CLOVER[5] system. Finally, conclusions and future work are outlined.

2. RELATED WORK
Many researchers have worked on focus+context approaches to graph representation. Contributions arise in the fields of Information Visualization, Graph Drawing, Software Analysis, among others. An overview of focus+context techniques can be found in [9]; and description of the original semantical focus+context for graphs can be found in [8]. Focus+context systems with hierarchical clustering can be classified according to the type of clustering performed, the mix of distortion techniques that is present, and when and how graph layout is performed. This section deals only with distortion and layout, which are intimately related.
**Fixed vs. incremental layouts**

Some systems do not perform graph relayout between navigational steps, and instead rely on a full layout (possibly accelerated via clustering) which is performed only once. This “fixed layout” approach is followed by [12], [10] and others. For instance, in [12], the whole layout is precomputed in advance, using a layout algorithm where proximity is closely related to cluster-belonging. When navigating, three areas are defined around the cursor, which behaves as a “magnifying lens”: in the outermost area, clusters are represented as large nodes, at a fixed maximum level of abstraction; in the intermediate area, the degree of abstraction varies between this maximum level and full detail; finally, in the central area there is no abstraction, and variable geometric zoom is performed.

Fixed-graph approaches such as the above can avoid recomputing layouts, because when substituting high-level clusters for single ‘cluster nodes’, these are simply located at the average position of the “leaf” nodes they represent. This minimizes change, and is an effective means for preserving user orientation. However, a single full layout cannot be used if the graph is being edited instead of simply navigated (adding or deleting edges from graphs can potentially alter the whole graph structure, and require a new clustering and a full relayout). A second problem is that generating a good layout for a large graph can require a long time (see [1] for an in-depth study of layout algorithms).

**The incremental approach**

On the other hand, incremental layout approaches (mainly [7] and [3]) do not need to perform full layouts, make a better use of display space, and can accommodate interactive graph editing. However, because layout is slow, the number of displayed nodes must be relatively small to preserve interactive behavior. This favors semantic zooming over geometrical distortion: the former lowers displayed node count, and the latter is most useful when large amounts of nodes are visible.

### 3. PRESERVING THE MENTAL MAP

Performing incremental layouts after each navigational or editing step implies that there will be greater differences between successive views, and therefore greater care is required to preserve the mental map during interaction.

We have identified a series of general “factors” that contribute to mental map preservation. While they affect all focus+context navigation schemes, our discussion will deal only with hierarchical graphs that undergo relayout after an edit or navigational action.

#### 3.1 Predictability

A first factor is to make the jump from one view to the next predictable. The user should be aware of what changes are to be expected in the view before triggering them. This goal can be furthered by providing visual feedback of actions before they take place, educating the user as to the internal workings of navigational actions, and keeping these simple to understand and use. As a side effect, any strategy that increases predictability of navigational actions will also offer valuable insights about the graph structure. This can lead to a better mental model. However, excessive navigational aids can also lead to information overload.

#### 3.2 Degree of change

A second factor is to minimize the jump itself. The new view will hide some nodes by collapsing them into higher-level clusters, and will expand new clusters into their components. Other sources of change include altering the underlying graph, or if the graph is being filtered before presentation, changes to the filtering.

Keeping a low degree of change between views depends heavily on the chosen incremental layout algorithm. Many factors determine the “goodness” of a layout. There is agreement on some criteria, but no officially accepted “goodness” metric - which would, in any case, depend heavily on the application.

Force-directed algorithms (FDAs), first proposed by [2], produce good layouts on a wide range of graphs, and are particularly well suited to incremental layout. Iterative implementations using gradient descent are easy to code, and can be tuned by altering the forces or adding new ones. Complexity of FDAs is usually referred to as $O(n^2)$, since each pass requires $O(n^2)$ calculations to find repulsive forces between every pair of nodes, and more iterations are needed for larger graphs.

When minimizing change, a balance must be struck between keeping changes low and finding a “better” layout (which will probably differ from the current one because of the stochastic nature of FDAs). This can be achieved during relayout or by transforming the resulting layout “a posteriori” into a version that is more similar to the original one. A posteriori transformations such as affine transforms (rotation, shear, symmetry) do not modify the “energy” of the layout, but can result in a better match to the previous one. However, weakly-connected subgraphs can undergo such transforms independent of each other, and detecting and reverting each of them is a difficult problem.

#### 3.3 Traceability

The last factor is to evidence the changes as they take place, so that they can be tracked and integrated into the user’s mental map. This is usually achieved via animation. A simple approach is to interpolate a series of frames between the old and the new view, and present them in quick succession. [6] presents a detailed study on the desirable characteristics of graph animations. Better animation results if the intermediate frames are individually correct graph drawings, avoiding such things as overlapping nodes and overlapping or easily confused edges, which may result in display of non-existing structures. Another important property is the “structure” of movements – a rotation of several nodes at once is much easier to follow than several independent movements. However, all this can be expensive to calculate, and animation should be perceived as continuous and smooth in order to be effective. If incremental layouts are performed and changes are kept small, the animation structuring techniques used in Marcy [6] are probably overkill.

### 4. MAP-PRESERVATION IN CLOVER

CLOVER[5] stands for Cluster-Oriented Visualization Environment. It uses hierarchical clustering and incremental layout to provide focus+context. Its main purpose is to be used as a framework for real-world graph-based editors and viewers. We have used it to implement several techniques that can be classified according to the above factors.
while trying to increase the usability of the system for an adaptive hypermedia editor. In the next section, the “aura”, layout cache, incremental FDA and animation mechanisms used in CLOVER are described and associated with the model. Table 4 depicts the relationships between mechanisms and model properties.

Table 1: Mechanisms and properties: predictability (P), degree of change (DoC) and traceability (T)

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>P</th>
<th>DoC</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aura</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Incremental FDA</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Animation</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Aura

The “aura” mechanism, depicted in fig. 1, seeks to improve predictability of navigational actions via node rollovers. Whenever the pointer is located over a node, the user will receive a visual hint describing the effects on the current view of choosing the hovered-over node as new focus. Hinting is done via a temporary colored border, which is currently green for those nodes that would be expanded should the focus change to the new prospective position, and red for those that will be collapsed (the rest receive no border at all). This allows users to rethink actions that would hide parts of the graph they don’t want to collapse just then. The user can then use the “freeze node” feature to mark nodes that should not be altered by the LoD algorithm.

Figure 1: Aura: hovering over node “B” highlights nodes that will be collapsed (empty triangles) or expanded (full triangles) if “B” is selected.

An incremental FDA

A modified force-directed layout is used to perform incremental layout. Force-directed layouts are good for incrementality, because they start their search for a local minimum starting from any position, for example the old node positions. On the minus side, unless care is taken, the graph may distort considerably from its original layout. One possibility is to try to undo these transformations with an affine transforms after they take place, and use the resulting positions as the layouts. However, any bridge or cut-point can result in a warped subgraph, and affine transforms are best applied on the complete graph. Our modified layout simulates an extra edge between the old position of each old node and the current node. No “virtual nodes” are added, therefore there is no extra repulsion, and the added complexity to the algorithm’s running time is small, as the dominating factor is by large the $O(n^2)$-per-pass required to calculate repulsive forces. Two consecutive screen captures of our incremental layout are visible in 2. As a last step, node overlap is removed using a modified version of the Force-Transfer Algorithm described in [14].

Figure 2: Before (left) and after (right) an incremental layout. The shift is that of of fig. 1.

History

Another strategy that improves predictability is the use of history. After a given action, if that action is undone (say, transferring the focus to the prior location), a user would expect to see the same view again. This can be implemented with a view-layout cache, where whenever a view is revisited, it can be restored from the cache. In graphs that do not change, the approach is simple and effective. However, if the user is allowed to edit the graph structure (and, potentially, the clustering), additional measures are necessary.

The history mechanism, depicted in fig. 3, maintains a layout cache for each view that has been presented (up to a configurable limit of layouts can be kept; once reached, a LRU replacement policy is enforced). Whenever a view is ready for representation, the position of all nodes (including subsumed ones, which inherit their cluster’s position) is stored in the cache. A hash of the currently-visible nodes is also stored. If no editing has occurred, the hash is useful to quickly retrieve a previous layout form the cache. If editing has occurred, the hash of visible nodes, not considering the edit, is searched for. If it is found, the stored positions are used to initialize an incremental layout. This works regardless of changes to the clustering structure, which is never stored in the layout cache.

Figure 3: History mechanism: the last frame reuses the layout from the first one

Animation

A state machine is used to achieve animation of layout changes:
first, all node collapses take place (animated so that nodes migrate towards what will become the new cluster). Small, incremental layouts are performed after every collapse.

- then, all node expansions take place (animated so that the nodes in exploded clusters migrate towards their new positions).

- finally, a last burst of layout, longer than the incremental layouts in previous steps, beautifies the results.

Figure 4 shows screen captures of the different steps of the animation algorithm applied to the focus change of fig. 1.

While not strictly related to animation, the “forward” and “backward” navigation actions enabled by the above-mentioned history mechanism allow a user to replay an animation. These actions have the same semantics found in web browsers: choosing to go “backward” undoes the last navigational action, repeating the animation in reverse order. The animation can then be replayed with the “forward” action. This can help a user regain orientation after a particularly confusing transition.

5. CONCLUSIONS AND FUTURE WORK

Interactive graph interfaces that implement focus+context via hierarchical clustering and incremental layout are good candidates for large graph visualization. To deliver on this promise, it is critical to preserve user orientation. This task can be decomposed into a series of properties: predictability demands that user actions have expected consequences, degree of change should be kept low to lessen the gap between one view and the next, and when changes do take place, traceability requires that they be as easy to follow as possible. We have described these factors, and related map-preservation mechanisms used in the CLOVER system to the factors they seek to address.

We are currently trying to extend map preservation to a multiple simultaneous view interface, where different focus+context views of a single graph, possibly with a different filters on each, can be used in parallel, and changes to any view are reflected on all others. The field of interactive graph visualization offers a wealth of challenges, and we are confident that this type of interfaces will soon reach widespread use.

6. ACKNOWLEDGMENTS

This work has been sponsored by the Spanish Ministry of Science with project code TIN2004-03140.

7. REFERENCES


