

Visualizing Large Graphs with Compound-Fisheye Views and Treemaps^{*}

James Abello¹, Stephen G. Kobourov², and Roman Yusufov²

¹ DIMACS Center,
Rutgers University

abello@dimacs.rutgers.edu

² Department of Computer Science,
University of Arizona

{kobourov,ryusufov}@cs.arizona.edu

Abstract. Compound-fisheye views are introduced as a method for the display and interaction with large graphs. The method relies on a hierarchical clustering of the graph, and a generalization of the traditional fisheye view, together with a treemap representation of the cluster tree.

1 Introduction

Many of the challenges in visualization today arise from the volume of data. As the volume of data grows, so too does our desire to visualize the data. Often the data contain relationships between objects, and can be represented as a graph. A great deal of research and investment has gone into developing better display systems, high-resolution screens, and visualization walls. However, no matter how good our display systems get and how many pixels per square inch can be obtained, there will always be graphs that are too large to be fully displayed and too complex to comprehend as a whole.

Graphs with hundreds of thousands of nodes and millions of edges are commonplace in many of today's applications, such as telecommunications, software engineering, and databases. Recent graph drawing algorithms allow us to compute layouts for large graphs in reasonable times. However, exploring and interacting with such graphs in their entirety is likely to be ineffective.

A visualization technique that relies on *fisheye views*, *clustering*, and *treemaps* is introduced in order to provide a way to explore and interact with large graphs. In this context, clustering implies any hierarchical decomposition of the graph. The cluster computation in turn yields level-views of the input graph at different levels of detail. Navigation from one level of the hierarchy to the next is provided by partial refinement and/or coarsening of different parts of the view. These operations correspond to zooming in and zooming out. *Compound-fisheye views* are introduced as a technique that provides high level of detail in the focus area while also providing a global view of the graph, through distortion of the view in

^{*} This work is partially supported by the NSF under grant ACR-0222920 and by ITCDI under grant 003297.

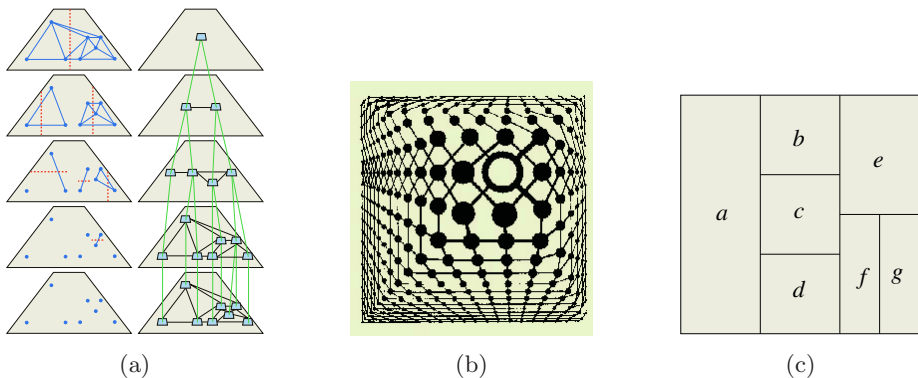


Fig. 1. Different graph views: (a) Multi-level view of a geometrically clustered graph; (b) Fisheye view of a grid-like graph; (c) Treemap for a small tree.

areas away from the focus. The compound-fisheye view combines clusters from different levels of the cluster-hierarchy by showing high detail clusters in the area of interest and progressively lower detail clusters away from the focus. A treemap view of the cluster tree is also used to provide a global view of the original graph.

1.1 Related Work

Multi-level display algorithms are described in the context of visualization for clustered graphs in [9]. Compound and clustered graphs are studied in [10, 18]. Multi-level views [8, 9] show large graphs at multiple abstraction levels. A natural realization of such multiple level representations is a 3D drawing with each level drawn on a plane at a different z -coordinate, and with the clustering structure drawn as a tree in 3D; see Fig. 1(a). The related concept of a graph sketch is introduced in [1] and is used in the *MGV* system [2].

The above algorithms assume that the clustering of the graph is given. In the case where the input graph has no clustering information, hierarchical clustering algorithms based on the structural properties of the graph can be used [4, 15]. Alternatively, geometric graph clustering based on binary space partitions can also be used to display large graphs, as described in [8]. The quality of the resulting multi-level drawings depends on the initial embedding of the graph in the plane. Hence, a good initial embedding of a large graph is a prerequisite for this method. Recently, a number of efficient algorithms for layout of large graphs have been developed, based on multi-scale, high-dimensional and spectral methods [12–14, 20]. Data structures supporting cluster-graph operations (such as cluster-expand and cluster-collapse) have been studied in [6, 7].

Fisheye views are introduced in the context of viewing and filtering computer programs [11]. Fisheye views show one area of interest quite large and in detail and show other areas successively smaller and in less detail by using a distortion function; see Fig. 1(b). 2-D fisheye view graph drawings with position and size

distortions are studied in [16]. Finally, treemaps [17] have been studied for over a decade as an efficient space-filling layout of tree-like structures; see Fig. 1(c). The nodes of the tree are displayed as nested rectangles in the treemap. The children of a node are within the rectangle of the parent. Squarified treemaps ensure good aspect ratio for the rectangles [5] and ordered treemaps keep related items spatially close to each other in the map [3].

1.2 Our Contribution

A visualization technique called a *compound-fisheye view* is described. It makes possible to extend the effective use of the traditional fisheye view to larger graphs. The technique relies on creating a clustered graph from the original graph, via a hierarchical clustering algorithm. The resulting cluster tree is then shown as a treemap and is also used to navigate the compound-fisheye view.

Compound-fisheye views allow the exploration of an area of interest in detail, by providing an interactive view of the graph, while still capturing the global context. When a focus node is selected from the current view, the corresponding subgraph at the next level is depicted and the view is updated. Similarly, less details can be requested about a particular node in the current view, which results in the replacement of the node (and its siblings in the cluster tree) with its parent, and the subsequent update of the view. These operations provide the ability to zoom in and out with respect to the current view.

Moreover, the compound-fisheye view has clusters from different levels in the cluster tree, depending on how close they are to the area of interest. In a way similar to traditional fisheye views, when more detail is requested in a particular area, the areas farthest away from the focus are automatically reduced in detail. Unlike traditional fisheye views, however, the reduction in the detail is achieved by replacing parts of the graph far away from the focus with coarser representations from the cluster tree (rather than just shrinking the area allocated to these parts, via distortion).

To aid the comprehension of the overall structure, a treemap view of the cluster tree provides global context. A prototype of the visualization system has been implemented and tested with graphs of varying sizes, up to 10,000 nodes. The screen-shots in Fig 7 show the system in action.

2 Hierarchical Clustering

A graph clustering algorithm is *geometric* if nodes are clustered according to their spacial locality, given an initial embedding of the entire graph. Similarly, a graph clustering algorithm is *structural* if nodes are clustered based on structural features of the original graph (such as connectivity and density). Any clustering algorithm can be used for the purpose of compound-fisheye view navigation, provided that the clustering is hierarchical. One structural clustering algorithm and one geometric clustering algorithm have been implemented as a part of the prototype.

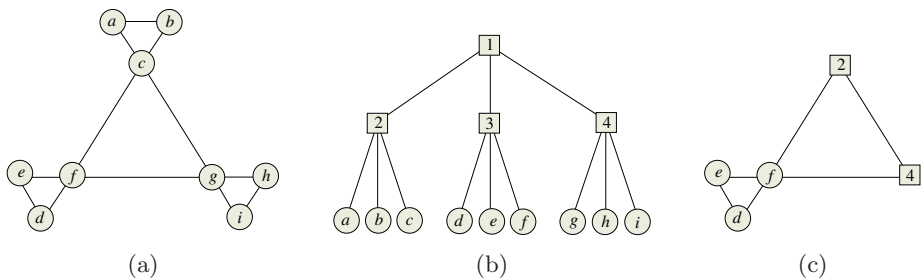


Fig. 2. (a) Input graph G ; (b) Cluster-tree T : lettered nodes represent input graph nodes and numbered nodes represent clusters; (c) Compound-fisheye view of $C = (G, T)$.

The structural clustering algorithm implemented in the system is a Markov clustering algorithm [19]. Markov clustering, also known as MCL, uses a random walk in the graph to identify densely connected components. MCL is also general enough to allow weighted graphs (including negative weights) that can be directed or undirected.

The geometric clustering algorithm implemented in the system is a Binary Space Partition (BSP) algorithm, similar to that in [8]. Starting with a 2D layout of the entire graph, a k -d tree recursive partition is used to obtain the clustering. The initial embedding is obtained using a high-dimensional multi-level method, similar to that in [12].

2.1 Clustered Graphs

Whether structural or geometric, the clustering algorithm produces a cluster tree. A leaf node in this tree represents a node from the original graph. An internal node represents a cluster of nodes, which consists of all the nodes in its subtree. A cluster may contain leaf nodes and/or other clusters. The cluster tree data structure is the interface between the clustering algorithm, the compound-fisheye view, and the treemap.

Fig. 2(a-b) show an example of an input graph G and the recursive clustering defined by a tree T . Together the input graph and the cluster tree make up the clustered-graph $C = (G, T)$. All the nodes of T at a given depth i represent the clusters of that level. A *view at level i* , $G_i = (V_i, E_i)$, consists of the nodes of depth i in T and a set of representative edges. The edge $(u, v) \in E_i$ if there is an edge between a and b in G , where a is in the subtree of u and b is in the subtree of v . Fig. 2(c) shows a compound-fisheye view of the clustered graph. The compound-fisheye view is initialized with the root of the tree and interaction is accomplished by means of the two clustered-graph operations: **cluster-expand** and **cluster-collapse**. The compound-fisheye view may contain a combination of nodes from different levels in T .

Consider the **cluster-expand** operation. This operation takes a node in the compound-fisheye view, called a *cluster-node*, replaces it with its children in the cluster tree, and performs the necessary updates to the edges in the graph.

Cluster-nodes do not exist in the original graph, but are created as part of a clustering tree. In Fig. 2(c), all numbered nodes are cluster-nodes and all the nodes from the input graph are leaves in the cluster tree. When expanding a cluster, determining which nodes need to be added to the graph is straightforward, since they will always be the children in the clustering tree of the cluster-node being expanded. The challenge arises when trying to determine what new edges should be added to the current view. An edge exists between two cluster-nodes only if some member from one cluster is adjacent to a member of the other cluster in the original graph. In Fig. 2 an edge exists between cluster nodes 2 and 4 because nodes *c* and *g* are adjacent in the original unclustered graph.

The **cluster-collapse** operation is simpler. When a cluster is to be collapsed, one of its children in the clustering tree must be selected, since the cluster-node representing the cluster will not be in the graph. In Fig. 2(b), the cluster-node 3 can only be collapsed by selecting one of its children, *d*, *e*, or *f* from the compound-fisheye view in Fig. 2(c). Thus, in order to collapse a cluster, the children of the cluster-node (siblings of the selected node in the cluster tree) must be identified. Next, the set of nodes that are adjacent to any of those children is identified. Finally, an edge is added between the collapsed cluster-node and each adjacent node.

3 Compound-Fisheye Views

Fisheye views of graph drawings allow a user to understand the structure of a graph near a specific set of nodes (local detail), and at the same time they display the graph's overall structure (global context). Such views achieve smooth integration of both local detail and global context by repositioning and resizing nodes and edges in the graph. However, even for graphs with a few hundred nodes, the benefits of this approach are lost as the areas away from the focus become too congested to comprehend.

The fisheye view idea is applied on a hierarchically clustered graph to obtain a *compound-fisheye view*; see Fig. 3 (a-b). The compound-fisheye view makes it possible to extend the effective use of the traditional fisheye view to larger graphs. Conceptually, the nodes in the compound-fisheye view of a clustered graph can be obtained by taking the intersection of an inverted cone with the level-views of the clustered graph; see Fig. 3(c). The cone is centered at the area of interest in the original graph (at the deepest level in the multi-level view). The farther away from the point of the cone, the coarser are the views that it intersects. Thus, parts of the graph that are far away from the focus are represented using clusters at higher levels in the cluster tree.

The compound-fisheye view allows the user to navigate it, modify it, and interact with it. The underlying representation provides an adaptive view with the look and feel of a normal graph. Interaction with the compound-fisheye view is accomplished through the **cluster-expand** and **cluster-collapse** operations. These operations correspond to zooming in and out with respect to the current view. When a focus area is selected, the corresponding subgraph at the next (higher or lower) level is depicted and the current view is updated.

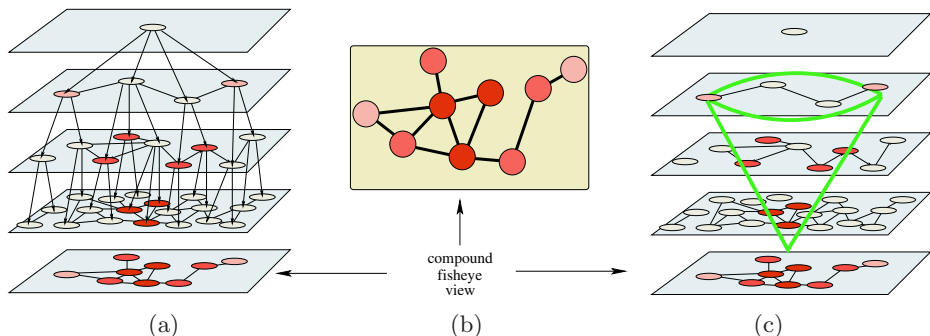


Fig. 3. Compound-fisheye view of a clustered graph: (a) A multi-level view of a clustered graph with highlighted nodes that make up the compound-fisheye view at the bottom; (b) The compound-fisheye view is made of clusters from three different levels of the hierarchy; (c) Conceptual view: the intersection of the multi-level view with an inverted cone.

Once the nodes in the current compound-fisheye view have been identified, the edges connecting them can be determined. Similar to the edges in the level-views, the edges in a compound-fisheye view are easily defined: the edge (u, v) is in the compound-fisheye view if there is an edge between a and b in G , where a is in the subtree of u and b is in the subtree of v . Unlike in level-views, the nodes in the compound-fisheye view are made of clusters from different levels in the cluster tree. This makes the edge computations more challenging, especially for large graphs. While efficient data structures have been designed for this problem [6, 7], a simple node and edge hashing scheme are employed in the implementation of the prototype system.

To ensure that the compound-fisheye view does not become too cluttered, the parts of the view away from the focus are automatically collapsed. With standard fisheye views, it is not difficult to support more than one focus. This idea generalizes to compound-fisheye views as well, although this functionality has not yet been implemented.

4 Treemaps

Treemaps are a space-filling graph visualization technique first introduced in [17]. An important feature of treemaps is that they make very efficient use of display space. Thus it is possible to display large trees with many hierarchical levels in a minimal amount of space. Fig. 4(a) shows a sample tree structure and Fig. 4(b) shows the corresponding treemap. The algorithm used to partition the display space is known as the “slice-and-dice algorithm” and functions like a k -d tree space partition. The positioning of tree nodes in a treemap is a recursive process. First, the children of the root are placed across the display area horizontally, where each node’s area is directly proportional to its weight. Then, for each node n already displayed, each of n ’s children is placed across vertically within n ’s display area. This process is repeated, alternating between horizontal and vertical placement until all nodes have been displayed.

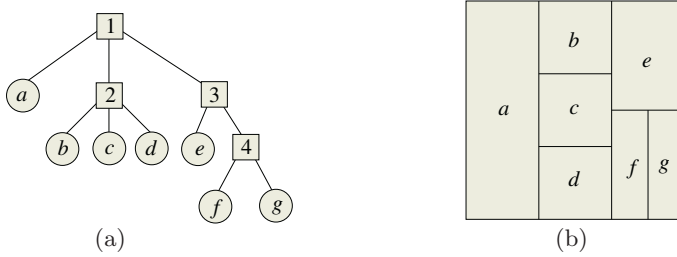


Fig. 4. A cluster tree (a) and its treemap representation (b).

Treemaps can be especially helpful when dealing with large clustered graphs. While the compound-fisheye view combines detailed local information and a global context, treemaps lend themselves naturally to showing the information encapsulated in the clustering tree. When viewing a graph at some level of abstraction, the viewer is really looking at nodes belonging to some level in the cluster tree. A treemap can display the whole structure of a cluster tree, thus allowing the user to place the current view in context.

In the standard treemap of Shneiderman [17] the nodes are represented as rectangles of various shapes. This makes a visual comparison of their importance (as determined by area) difficult, especially as the rectangles vary in orientation as well. In squarified treemaps [5] the *aspect ratio* (the ratio between the width and height of a rectangle) is taken into account when placing nodes in the treemap. The resulting treemaps contains squarish elements, making it easier to visually compare their areas.

Squarified treemaps with a modified visual appearance are implemented in the prototype of the compound-fisheye view visualization system. In traditional treemaps, only the leaf nodes of a tree are displayed as rectangular areas. It is often difficult to determine the nesting depth of the treemap structure as can be seen in Fig. 5(a). For the purpose of navigating the compound-fisheye view it is important to show depth information about the clustering tree, so the first step is to display the nesting information as shown in Fig. 5(b). In order to make the nested representation easier to view, progressively thinner borders for deeply nested elements are used. The thickness of the border around an element is inversely proportional to its level in the original tree. Thus, the root node has the thickest border and the leaf nodes have the thinnest borders as shown in Fig. 5(c).

5 Visualization Technique

Our visualization technique provides two views: one of the compound-fisheye view and another of the treemap defined on the cluster tree. On their own, neither of these approaches is powerful enough to represent and navigate a large graph. The treemap algorithm applies only to trees, and while it could be applied to a spanning tree of the graph, it does not show connectivity well. The compound-

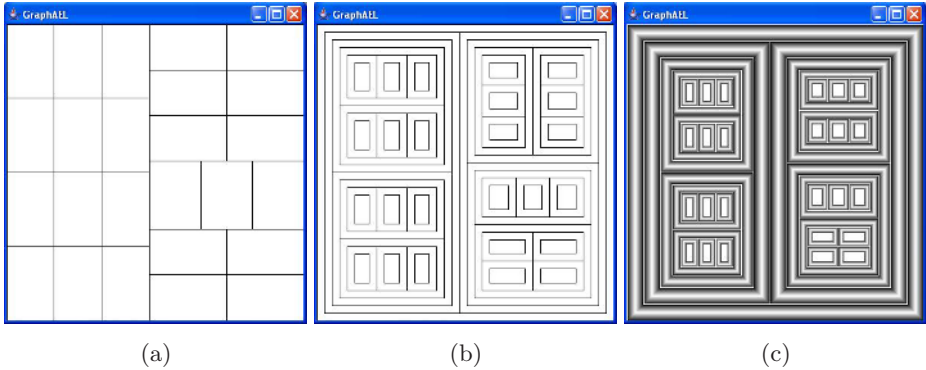


Fig. 5. (a) A squarified treemap representation of a 25-node tree; (b) the same tree with nested rectangles; the same tree with frames.

fish-eye view abstracts a great deal of the graph information, which can only be recovered by recursive expansion of clusters. While showing connectivity and local details well, it only shows an abstraction of the overall structure, making navigational decisions difficult.

Together, the compound-fisheye view of the graph and the treemap of the cluster tree offer a better approach to showing both local details and global context. The combined view is shown in Fig. 6. One of the main shortcomings of a compound-fisheye view is that nodes that are clustered become invisible in the display and the viewer cannot deduce information about the structure underneath the cluster. By using a combined view displaying both the compound-fisheye view and the treemap, information about the subtree rooted at the cluster-node can be better conveyed. In Fig. 6, the red node (rightmost node) has been selected by the user and its corresponding rectangle in the treemap is highlighted in blue (top left). It is easy to see that the selected cluster-node is

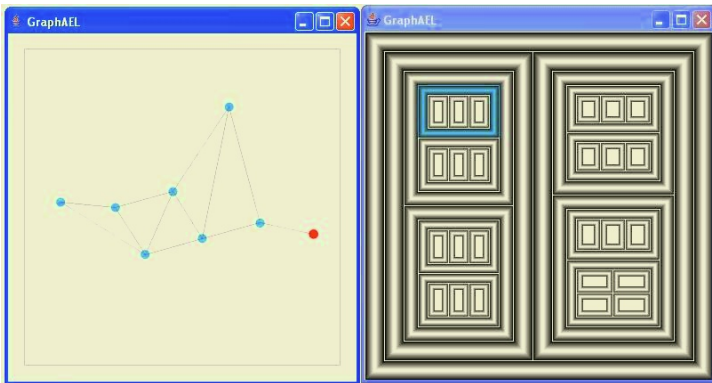


Fig. 6. Compound-fisheye view and treemap: the red cluster-node (rightmost) has been selected and its corresponding representation (top left) is highlighted in the treemap.

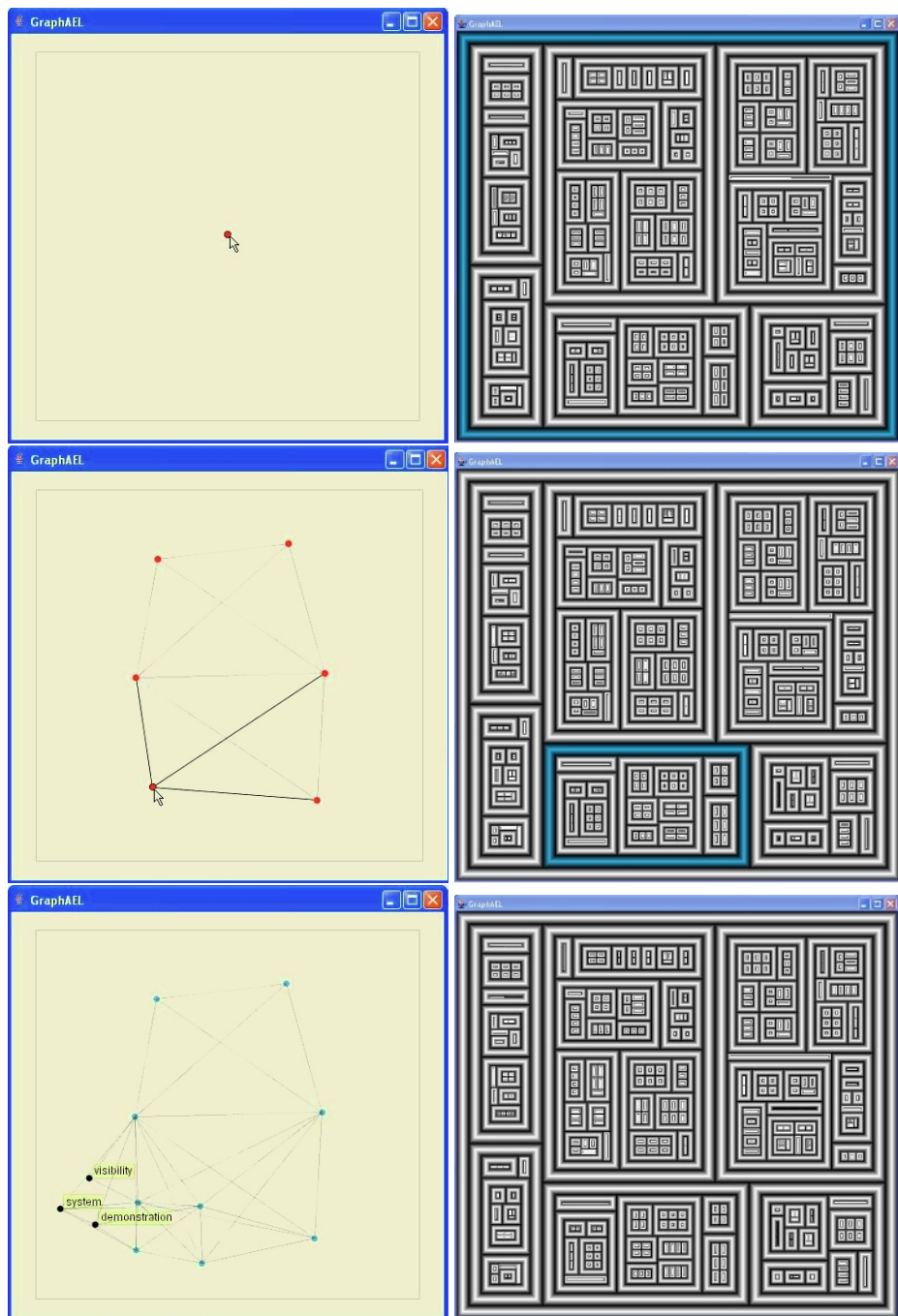


Fig. 7. Interaction with the system: steps 1, 2, and 3.

at level 4 in the clustering tree, contains 3 children that are also leaves, and if the user chooses to expand the cluster-node it will be replaced by 3 nodes in the current view.

When a user selects a node by placing the mouse pointer over it, it is highlighted along with all of its cluster tree siblings. This reveals the branching factor and permits the user to see the set of nodes that will be collapsed, should he choose to collapse the cluster (via a right mouse-click). During a collapse operation, the nodes being collapsed are highlighted in red, and whenever an expand operation is performed (via a left mouse-click), the cluster-node being expanded is marked in green. Once a cluster has been fully expanded, the resulting nodes are the nodes from the original graph and are colored in black. If all clusters are expanded, then the resulting graph is identical to the original graph and will contain only black nodes.

6 Conclusion and Future Work

The technique described in this paper has been implemented in a Java prototype. Fig. 7 shows snapshots of the exploration of the GD literature topic graph for the years 1994-2000, with 332 nodes and 1,338 edges. In step 1 the cluster tree root and its treemap are shown. In step 2 the root is expanded and one of the cluster-nodes has been selected (highlighting its corresponding treemap region). In step 3, the selected node has been expanded and labeled leaves of the tree appear in the view. For this example, the combined computation time needed for the initial layout, clustering, treemap, and rendering take under 5 seconds. Interaction with the compound-fisheye view is in real-time.

While the system can deal with larger graphs (with up to 10,000 nodes) the computation times are not nearly as good. Incorporating this prototype into a fully functional graph visualization system that can handle large graphs will be a difficult challenge, but one worth pursuing. Efficient data structures and algorithms, to support expand/collapse operations will become important if one requires real-time interaction with graphs with hundreds of thousands of nodes.

The current system uses cluster-node positions that are set by the clustering algorithm and are never modified. A natural alternative is to apply a layout algorithm to the compound-fisheye view and reapply it after an expand/collapse operation is performed. Since the view changes as nodes are being added/removed through expand and collapse operations, the layout algorithm must preserve the mental map between consecutive layouts and make smooth transitions between such layouts.

References

1. J. Abello, I. Finocchi, and J. Korn. Graph sketches. In *IEEE Proc. Information Visualization*, pages 67–71, 2001.
2. J. Abello and J. Korn. Mgv: A system to visualize massive multi-digraphs. *IEEE Transactions on Computer Graphics and Visualization*, pages 1–16, 2002.

3. B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Transactions on Graphics*, 21(4):833–854, 2002.
4. U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. In *11th Euro. Symp. on Algorithms (ESA)*, pages 568–579, 2003.
5. M. Bruls, K. Huizing, and J. J. van Wijk. Squarified treemaps. In *Proc. Joint Eurographics/IEEE TVCG Symp. Visualization, VisSym*, pages 33–42, 2000.
6. A. L. Buchsbaum, M. T. Goodrich, and J. R. Westbrook. Range searching over tree cross products. In *8th Euro. Symp. on Algorithms (ESA)*, pages 120–131, 2000.
7. A. L. Buchsbaum and J. R. Westbrook. Maintaining hierarchical graph views. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 566–575, 2000.
8. C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Balanced aspect ratio trees and their use for drawing large graphs. *Journal of Graph Algorithms and Applications*, 4:19–46, 2000.
9. P. Eades and Q. Feng. Multilevel visualization of clustered graphs. In *Proceedings of the 4th Symposium on Graph Drawing (GD)*, pages 101–112, 1996.
10. P. Eades, Q. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In *Proceedings of the 4th Symposium on Graph Drawing (GD)*, pages 113–128, 1996.
11. G. W. Furnas. Generalized fisheye views. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '86)*, pages 16–23, 1986.
12. P. Gajer and S. G. Kobourov. GRIP: Graph dRrawing with Intelligent Placement. *Journal of Graph Algorithms and Applications*, 6(3):203–224, 2002.
13. R. Hadany and D. Harel. A multi-scale algorithm for drawing graphs nicely. *Discrete Applied Mathematics*, 113(1):3–21, 2001.
14. D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *Journal of graph algorithms and applications*, 6:179–202, 2002.
15. R. Sablowski and A. Frick. Automatic graph clustering. In *Proceedings of the 4th Symposium on Graph Drawing (GD)*, pages 395–400, 1996.
16. M. Sarkar and M. H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–84, 1994.
17. B. Shneiderman. Tree visualization with treemaps: a 2-d space-filling approach. Technical report, HCI Lab, University of Maryland, Mar. 1991.
18. K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892, 1991.
19. S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.
20. C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *Proceedings of the 8th Symposium on Graph Drawing (GD)*, pages 171–182, 2000.