A Space Efficient Clustered Visualization of Large Graphs

Mao Lin Huang and Quang Vinh Nguyen
Faculty of Information Technology, University of Technology, Sydney, Australia
maolin@it.uts.edu.au, quvnguye@it.uts.edu.au

Abstract

This paper proposes a new technique for visualizing large graphs of several ten thousands of vertices and edges. To achieve the graph abstraction, a hierarchical clustered graph is extracted from a general large graph based on the community structures which are discovered in the graph. An enclosure geometrical partitioning algorithm is then applied to achieve the space optimization. For graph drawing, we technically use the combination of a spring-embbeder algorithm and circular drawings that archives the goal of optimization of display space and aesthetical niceness. We also discuss an associated interaction mechanism accompanied with the layout solution. Our interaction not only allows users to navigate hierarchically up and down through the entire clustered graph, but also provides a way to navigate multiple clusters concurrently. Animation is also implemented to preserve users' mental maps during the interaction.

1 Introduction

Graph visualization has been widely used in human-computer interaction. A graph commonly includes a node set and an edge set to represent entities and relationships between entities respectively. In real-world applications, graphs could be very large with thousands or perhaps millions of nodes, such as citation and collaboration networks and the World Wide Web (WWW). As the result of rapid increasing of the size in networks, the large scale visualization has become one of the hottest topics in Information Visualization. The question about how to comprehensively display large graphs on the screen becomes the key issue in graph visualization. However, the display of large graphs can decrease significantly the performance of a visualization technique which normally performs well on small or medium size of datasets. Large graph visualization is usually suffered from poor running time and limitation of display space. In addition, the issue of "view-ability" and usability also arises because it will be almost impossible to discern between nodes and edges when a dataset of thousands of items are displayed [13].

It seems that classical graph models with a simple node-link diagram tend to be inadequate for large scale visualization with several thousands of items. The lack of formal hierarchical structures in real world applications could limit the conveying and perception of the complicated information. Figure 1 shows an example of the graph visualization of a WWW site which illustrates two typical major problems:

1. Too many nodes (pages) to be displayed and the layout of such a large geometrical area could not be fitted in one single screen
2. The layout of the graph has inefficient utilization of display space with many unused areas in the display.

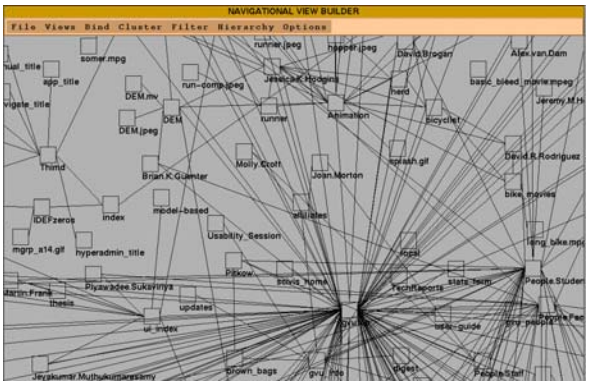

Figure 1. An example of a large graph visualization using the classic virtual-page technique.

To solve the first problem, a well established new graph model to accommodate with the visualization of large graphs is required. We believe that one way to deal with the display of large graphs is to provide users with a certain degree of Graph Abstract. That is to filter out some details

DOI 10.1109/ICIG.2007.10

of the graph drawing which the users are not interested at a time, while the overall structure of the graph drawing is maintained for navigation.

Among several available graph visualization approaches, we believe that the use of clustered graph is a better option for graph abstraction. It is believed that a good visualization system for very large graphs should be a combination three components including graph drawing, graph clustering and interaction [15] [4]. Therefore, visualization of clustered graphs such as [7] [9] is one of excellent approaches to deal with large graphs through the graph abstraction. A clustered graph can be extracted from a general graphs by partitioning recursively the graph into hierarchy of sub- graphs for clustered visualization, which simplifies the complex structures of large graphs through the global abstraction for easy interpretation, perception and navigation of large information spaces.

To solve the second problem mentioned above, we need to optimize the layout algorithms so that the utilization of display screen could be maximized. As a result it allows more nodes to be displayed. The research from Ware [21] shows that more information can be displayed on very high-resolution and large screen, but it does not necessarily provide very much more information into the brain. This is because the conventional monitor covers only 5-10% of visual field in the normal condition, but it uses as much as 50% of brain pixels [22]. The study also shows that the uniquely stimulated brain pixels peak at the width of a normal monitor view, and it is effective (but not critical) to increase the number of pixels for the normal desktop to reach the limit of the brain pixels. Therefore, the investigation of optimized visual abstraction (clustering) techniques that could provide viewers with more comprehensive views of the large graphs is important.

2 Related Work

Large graphs visualization has been recently received a lot of attention of researchers in both information visualization and graph drawing communities. Although some newly available techniques are quite capable of visualizing large graphs of thousands to hundred thousands of nodes and edges, named a few [3] [12] [20] [1] [11] and [10], the visualization of large graphs is still one of the hot topics in information visualization.

Harel and Koren [12] described a technique to draw a graph that used high-dimensional embedding and then projected it onto a 2D plane. Although this technique is very fast and capable of exhibit graph in various dimensions with some good navigational ability, it is more suitable for visualizing mesh-graphs rather than tree-like graphs or clustered graphs. One of the good approaches for handling large graphs is to use multi-scale visualization [11] [3] and [5]. These techniques typically apply a force-directed algorithm to draw large graphs using multi-scale scheme, in which they try to beautify the coarsest-scale representation. These technique aim to improve the processing speed while maintaining the graph niceness. Good visualization of large graphs can also be archived by using multilevel techniques [20] [2]. In short, techniques in this approach aim to improve the visual appearance of the visualization by defining different levels for a structure so that they can present the graphs using an optimal algorithm at each level.

Although the above techniques are quite capable of visualizing large graphs, the space-efficiency is not considered in these techniques, in which could limit the amount of information to be visualized on the screen at a time. Fekete et al. [8] presented a space-efficient visualization of graph using a modification of the well-known *Tree-maps* [14]. Technically, the authors used *Tree-maps* to display the tree structure of graph and used explicit link curves to present the other links. This technique is optimal in term of using display space and it is quite useful for visualizing structures that the underlying trees have some meaning. However, the technique does not perform well in general graphs and clustered graphs because the link curves may cause the unnatural look of the graphs. Some preliminary work has been carried out and from which two tree visualization techniques *Space-Optimized tree* (or SO-tree) [17] and *EncCon tree* [18] have been developed that can quickly display large trees with maximized utilization of display space. However, these techniques are only suitable for trees (hierarchical structures).

This paper proposes a new technique for visualizing very large general graphs. Our technique is similar to the framework of *Tulip* [3] which consists of three components: graph clustering, graph layout and interaction. We first use a new clustering algorithm to partition the complete graph into abstract clusters for achieving the view abstraction; that greatly reduces the visual complexity of the graph layout, and enhance the comprehension and understanding of the graph.

The clustered graph is then visualized by using a new space-efficient layout technique which is a combination of layout algorithms. This geometrical optimization of graph layout allows more data items (and clusters) to be displayed within a limited screen resolution. Our visualization provides viewers with not only an abstract view of the entire graph but also an interaction technique for navigating through the large graph. The navigation method allows users to navigate hierarchically through the clustered graph and navigate across a number of selected clusters. All the interactions are accommodated with animation to preserve users' mental maps during the navigation.

3 The archtecture of our Visualization

Our model for visualizing large graphs includes several processes which are illustrated at Figure 2. There are two major phases involved in this model including the clustering analysis and the user interface phases. These two phases operate independently.

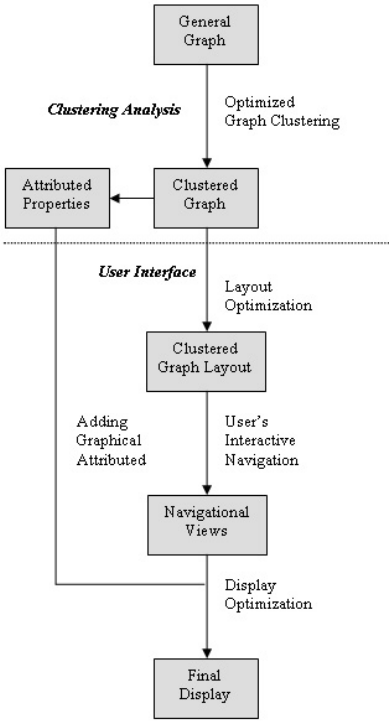

Figure 2. An architecture model for visualizing large graphs.

The *clustering analysis phase* is responsible for analyzing a large graph dataset and partitioning it into an optimized clustered graph based on discovered internal communities. In short, the clustering algorithm recursively divides the graphs into smaller and smaller sub-graphs based on the density of connection within and between subgroups. Although this process does not require a very fast algorithm and it can run independently on a high-speed workstation, the computational cost of clustering algorithms should be controlled with the worst-case running time O(n^2) on a sparse graph or faster to ensure its capability of handling hundreds thousands of items within a few hours using a ordinary personal computer. The clustering process also extracts the attributed properties for nodes, edges and relations between sub-graphs.

The *user interface phase* is responsible for visualization and navigation of the clustered graph, including the processes of layout optimization, interactive viewing and display optimization. A combination of a number of layout algorithms is employed which aims to optimize the geometrical space so that the large graph can be drawn at a normal screen size.

During the navigation of clustered graphs, we allow users to interactively adjust the views to reach an optimized representation of the graph; from which users can obtain the best understanding of the data and structures they are currently interested in. This visualization is involved with real time human-computer interaction. Therefore, very fast graph layout and navigation algorithms are required for handling hundred thousands of items within minutes or seconds using a personal computer with limited display space and computational power.

The final display is created through the view navigation and graphical attributing. We use rich graphic attributes to assist viewers to quickly identify the domain specific properties associated with data items. We next describe briefly of the clustering and the technical detail of our visualization technique.

4 Graph Clustering

We use a graph clustering method which can quickly discover the community structure embedded in large graphs and divide the graph into densely connected sub-graphs. Our clustering algorithm is a modification of Newman's algorithm [16] which considers the equality of size among clusters during the partitioning. The proposed algorithm can not only run fast in time *O((m+n)n)* similarly to [16], but also achieve a consistent partitioning result in which a graph is divided into a set of clusters of the similar size. The balanced size of clusters could provide users with a clearer view of the clustered graph and thus it makes easier for users to visualize and navigate large graphs. Our balanced clustering technique also achieves the layout optimization at both global and local levels of the display through the use of the *enclosure+connection* layout technique. This allows more visual items to be displayed within limited screen resolutions with comprehensive views. The combination of our clustering method and a space-efficient layout technique would enable the visualization of very large general graphs with several thousands of elements.

5 Graph visualization

We use a new space-efficient visualization technique similarly to *EncCon* [18] to optimize the geometrical space for visualizing large clustered graphs with several thousands of nodes and edges. This technique consists of two components, the space-efficient layout and the interactive navigation. The layout of clustered graph is generated by using

a combination of an extended fast enclosure partitioning algorithm, called *Clenccon*, and a number of traditional graph drawing algorithms, including a spring-embedder algorithm [6], a circular drawing, and simple layout algorithm, to archive the objectives of space-efficiency, aesthetical niceness and fast computation. Although some available techniques can use any layout algorithm at each clusters, such as using a spring-force algorithm [20] [11], in our belief, the choice of a space-efficient enclosure-partitioning algorithm at high levels will be more efficient because it provides more space for displaying information at the limited display.

The *Clenccon* layout algorithm is only applied to those non-leaf sub-graphs, in which the space utilization and computational cost are crucial. In other cases, the other layout algorithms, such as *Spring-embedder* and *circular drawing*, are applied to the calculation of the position for those leaf sub-graphs, which contain a small number of nodes in which the space utilization issue becomes less important and, therefore, the aesthetic niceness and flexibility issues need to be more considered. The use of a particular layout algorithm depends on the nature of the leaf subgraphs. Our system also displays a high-level node-link diagram to present the overall clustering structure explicitly (see examples at Figure 4 and Figure 5).

Technically, the algorithm inherits essentially the advantage of space-filling techniques [14] [18] that maximize the utilization of display space by using area division for the partitioning of sub-trees and nodes. Note that the issue of space utilization becomes significantly important when visualizing large graphs with thousands or even hundred thousands of nodes and edges because of the limitation of screen pixels. It is similar to *EncCon* [18] that use a rectangular division method for recursively positioning the nodes hierarchically. This property aims to provide users with a more straightforward way to perceive the visualization and it ensures the efficient use of display space. Our new technique is applied for clustered graphs rather than simple tree structures; therefore, the algorithm takes the connectivity property between sibling nodes into its partitioning process. We now describe the technical detail of our layout and navigation algorithms.

5.1 Layout Algorithm

Our layout algorithm is responsible for positioning of all nodes in a given clustered graph in a two-dimensional geometrical space, including a vertex subset $\{v_1, v_2, ..., v_n\}$ in *V* and a cluster subset $\{v'_1, v'_2, ..., v'_n\}$ in *V'*. A clustered graph *C* = {*G, T*} is derived by a general graph *G* = {*V, E*} and a cluster tree *T* whose leaves are in *V*. Each cluster $v'_i = C'$ is a sub cluster graph, contains a subset of *V* given by the leaves of the sub-tree *T'* rooted at *r'*. The root *r'* of the sub-tree *T'* is also called a super-node. The super nodes are not displayed in our visualization but they are used for partitioning process of calculating the local region for sub cluster graph. For the partitioning of clustered graph *C*, We define a virtual tree consisting of a set of super-nodes for area division. We define a super-node $r(v'_i)$ for each cluster v'_i. Further description of the clustered graph can be found in [9].

The layout algorithm is a combination of two algorithms: 1) *Clenccon* - a fast area division algorithm and 2) graph drawing algorithms including a *spring-embedder* algorithm, a *circular drawing* and a simple algorithm to lay out a very small number of nodes.

Each cluster v'_i is bounded by a rectangular local region $R(v'_i)$ centered at super-node $r(v'_i)$. The drawing of the corresponding sub-clustered-graph $G(v'_i)$ is restricted to be inside the geometrical area of $R(v'_i)$. Therefore, the local region $R(v'_i)$ of cluster v'_i is the sum of the rectangular areas assigned to its children. The position of the super-node $r(v'_i)$ of v'_i is at the centre of the rectangle defined by $R(v'_i)$. The position of leaf nodes is defined by either the spring embedder, the circular drawing or the simple layout algorithms.

We first assign the entire rectangular display area as the local region to the clustered graph C. We then recursively partition the local regions for every sub-clusters until all the clusters are reached.

We assign a weight *w(v')* to each vertex *v'* for the calculation of the local region *R(v')* of the vertex. Although the weight of each vertex can be associated with its property, all the leaf vertices in our experiments have the same weight. Suppose the rectangular local region *R(v')* for cluster *v* is drawn, we then need to calculate the local regions $\{R(v'_{l+1}), R(v'_{l+2}), ..., R(v'_{l+k})\}$ for its sub-clusters $\{v'_{l+1}, v'_{l+2}, ..., v'_{l+k}\}$. The partitioning ensures that the area of each rectangle $R(v'_{l+i})$ is proportional to the weight $w(v'_{l+i})$ of the cluster v'_{l+i}. The calculation of *w(v')* of a cluster *v'* is done recursively from leaves of the cluster tree to a the root of the cluster tree. The calculation is done by the following formula:

$$w(v) = w_0 + S \sum_{i=1}^{k} w(v_{l+i}) \quad (1)$$

where w_o is the internal weight of cluster *v'*. Although the internal weight of a cluster can be defined by the cluster's attributed property, we define the internal weight of all clusters to be 1 for all experiments. *S* is a constant *(0 ¡ S ¡ 1)*, and $w(v'_{l+i})$ is the weight assigned to the ith child of cluster *v'*. The constant *S* determines the size difference of local regions of all clusters based on the number of descendants of those vertices.

The process of recursive partitioning *R(v')* into subregions $\{R(v'_{l+1}), R(v'_{l+2}), ..., R(v'_{l+k})\}$ for all its children

clusters $\{v'_{l+1}, v'_{l+2}, ..., v'_{l+k}\}$ is illustrated as the procedure below:

```
procedure partitioning (Node N) {
  if all child-nodes of N are leaf-nodes then
  {
    lay out the child-nodes using a graph algorithm;
    scale the layout to fit with rectangular local region;
  }
  else
  {
    lay out the child-nodes using Clenccon algorithm;
    for each non-leaf child-node of N
    {
      partitioning(child-node);
    }
  }
}

procedure Clenccon-layout (Node [] Nodes) {
  group linked-nodes into subgroups;
  sort the subgroups based on connection and size;
  lay out subgroups using EncCon algorithm;
  for each subgroup
  {
    lay out nodes in subgroup using EncCon algorithm;
  }
}

procedure childnode-layout (Node [] Nodes) {
  if number of child-nodes < K1 then
  {
    lay out the child-nodes using simple algorithm;
  }
  else if number of child-nodes > K1 and
    no. edges / no. child-nodes > K2 then
  {
    lay out the child-nodes using circular drawing;
  }
  else
  {
    lay out the child-nodes using Spring algorithm;
  }
}
```

where *K1 = 6* indicates the number of nodes that is suitable for each algorithm. If there are just a few number of nodes, i.e. less than 10 nodes, a simple node location algorithm can perform well. *K2 = 5* indicates the ratio of the number of edges over the number of nodes. Because the force directed algorithms do not usually perform well for a graph whose the number of edges is much larger than the number of nodes, we use the circular drawing in this situation.

The detail description of the area partitioning *EncCon* can be found at [18]. Although there are several improved force-directed layout algorithm, the traditional *Spring-Embedder* [6] layout algorithm is chosen in our implementation. This is because the algorithm is simple, easy to implement, flexible and it performs well in general for a small number of nodes, in which a more complicated algorithm is not necessary. The circular drawing is a simple technique but is very effective to show the pattern of relationship for a graph whose the number of edges is much larger than number of nodes. Technically, we place all nodes equally on a circle where relational nodes are located close together so that the pattern of connections can be display more clearly (see Figure 5 and Figure 6).

Figure 3(a) shows an example of partitioning and drawing a small clustered graph using our algorithm. We can see that the algorithm uses the Clenccon algorithm to layout three cluster nodes and their inter-relationships to ensure the efficient utilization of space and uses the spring-embedder algorithm to draw the sub-graphs within each cluster to achieve the aesthetic niceness and flexibility. Note that the inter-relationships among clusters here are represented by using abstract links. Figure 3(b) shows the same example of the partitioning, but the inter-relationships among clusters are represented by using the original structure of relationships. Figure 4, Figure 5 and Figure 6 illustrate the applications of our layout algorithm on various very large datasets. These pictures show clearly the structure of clustered graphs, in which sub-graphs are efficiently partitioned and drawn inside their local regions. All of these pictures use abstract links to represent the inter-relational structures among clusters.

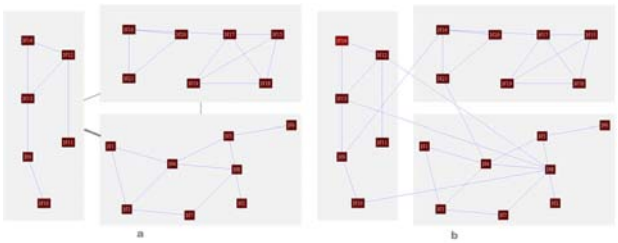

Figure 3. An example of partitioning and drawing a small clustered graph using our algorithm, a) use the abstract links among clusters; b) use the original structure of relationships.

5.2 Navigational Views

There is no pure visualization technique that could assist data retrieval without providing users with an associate navigation mechanism in graphic user interface design. In our user interface component, during the navigation, we enable users to interactively adjust the views to reach an optimized representation of the graph; from which users can obtain the best understanding of the data and its relational structures they are currently interesting in.

In our prototype, we use a multiple-views technique [19] to achieve the focus+context view navigation of large clustered graphs. This technique allows the exploration of data hierarchically as well as across multiple selected clusters to quickly focus on the interest parts of data. Therefore, users can semantically zoom in one or many areas of interest or sub-clusters (see Figure 9) while retaining simplified context views. It effectively uses both focus and context views

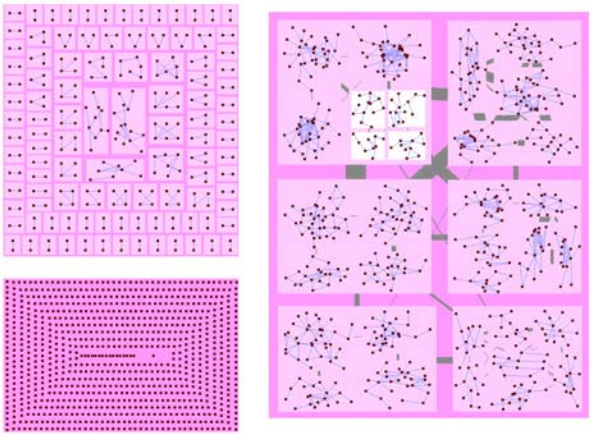

Figure 4. An example of clustering on a large citation dataset with over 2000 vertices and 4200 edges of the social network papers.

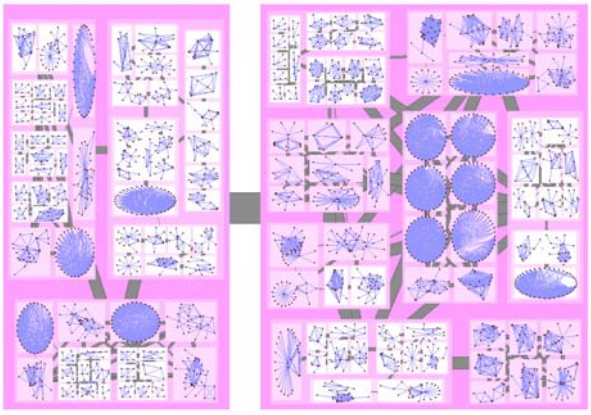

Figure 5. An example of clustering on a protein dataset with very strong relations produced by our algorithm.

to enable the interactive investigation of very large data. In our visualization, we provide three views to assist viewers to obtain the best understanding of the dataset and its relational structure they are currently exploring. These views are: 1) a *full-context view*, 2) a *current-context views* and 3) a *main view* (see Figure 7 and Figure 8).

The *full-context view* is displayed as a small panel located at the top-left side of the visualization. This view displays the entire context of a large clustered graph in high-level abstraction with little details. This enables users to always maintain an overall view of the information. This view only shows three levels of the cluster tree from the root. The view updates automatically according to changes occurred in the database. It highlights the context of the current sub-graph the user is interacting, so that users can always identify where they are and where they have been in a large information space. For example, in Figure 7 we can easily identify the current-context view which is on the top-left region of the *full-context view* and the *main-view* which is the right sub-graph inside the top-left region of the *full-context view*.

The *current-context view* is displayed as a small panel locating under the *full-context view* at the left-hand side. This view displays the immediate context or the up-level view of the *main view*. Similar to the *full-context view*, this view displays up to three levels of clustered graph from the root in an abstraction manner. The current-context panel also updates automatically and highlights a focused sub-graph which to be viewed in the main view in details. This enables users to easily identify their focused region at the current context (see Figure 8).

The *main-view* is displayed in a large area of the visualization. It displays the detail of a focused sub-graph and the structure of the clustered graph in an enclosure manner with color brightness. This view also shows the connectivity strength between clusters using the widths of edges. Furthermore, the sizes of nodes in the *main-view* are automatically rescale based on the number of visible nodes.

5.3 Final Display

Some graphical attributes are employed in the final display of clustered graphs to assist viewers to quickly identify the domain specific properties of data and the hierarchical structure of the clustered graph. Colors are employed to assist viewers to quickly identify the hierarchical structure of the clustered graph. In our prototype, the local regions of nodes at different levels are painted with a same color but at different brightness. This drawing property aims to provide a pleased view while retaining the clarity between sub-graphs. Although the use of brightness of colors for background can theoretically apply to several levels of hierarchies, we also apply this drawing property to first four hierarchical levels to avoid the confusion with too many colors.

Width or thickness of the edge is employed to represents the weight of the edge (or the number of connections between two nodes). For example, in Figure 3(b), there are 3 links between the left cluster and bottom-right cluster. However, Figure 3(a) displays only one thick abstract link between these clusters. We can see that edges among leaf nodes are drawn with light-green color and edges among non-leaf nodes (or between two clusters) are drawn with light-gray color. Furthermore, to avoid the overlapping due to the over-thickness of edges, the thickness of an edge is limited. Therefore, if the weight of an edge between two

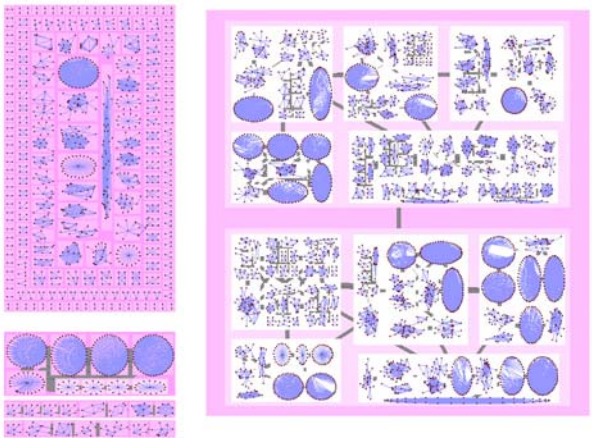

Figure 6. An example of clustering on a protein dataset with our algorithm cluster a protein network with over 15,000 nodes and 40,000 edges.

Figure 7. A clustered visualization of a very large protein dataset with three views: 1) a full-context, 2) a current-context view and 3) a main view.

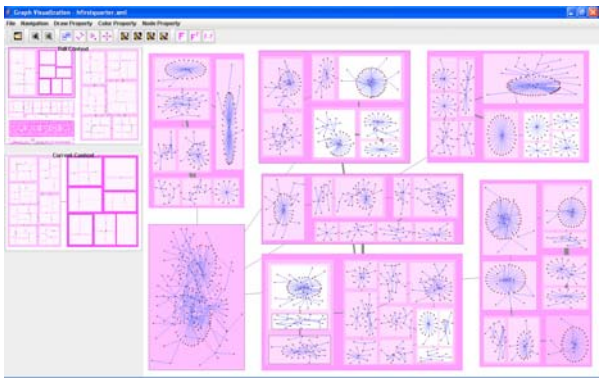

Figure 8. A navigational view of a selected sub-graph of the same dataset used in Figure reffig:fig7 produced by our visualization system.

clusters is greater than a limited number in our implementation, the edge is drawn at its limited maximum-width with darker boundary.

Although the area partitioning algorithm can theoretically optimize display space of the entire graph layout, a small portion of display space is reserved as gaps among clusters for showing abstract links between clusters and these abstract links which indicate the connectivity strength between clusters

5.4 Complexity of the layout algorithm

Our layout algorithm is a combination of the *Clenccon* and other simple layout algorithms, including *Spring-embedder*, *circular layout* and a simple node placement layout for very small number of vertices. The Clenccon is applied to those non-leaf vertices while either the *Spring-embbeder*, *circular layout* and a simple layout algorithm is applied to those leaf vertices. The *Clenccon* extends the *EncCon* [18] layout algorithm for hierarchically partitioning the clustered graph. Therefore, the computational complexity of both *EncCon* and *Clenccon* algorithms is linear or O(n).

The cost for running *Spring-embedder* algorithm is quite expensive, especially for laying out large graphs. Fortunately, this algorithm is only applied to clusters with a small number of nodes. The size of each group of nodes in our experiments varies from a few to a few tens of nodes. As a result, the computational complexity of the Spring Embedder algorithm on the leaf nodes are $kO(c^3)$ in worse case where *k* is the number of leaf groups and c is the number of vertices within a group which is almost constant. For example, a partitioning divides a large sparse graph of 1000 vertices into 20 leaf groups of roughly 50 vertices; and, the Spring Embedder algorithm is applied independently to 20 groups of 50 vertices. Therefore, the worse case of the computational cost is about $20 \cdot 50^3$) which is only a small fraction of the total computational cost 1000^3 for the layout of the entire graph. The computational cost for calculating the position of nodes on a circle is linear and the simple algorithm for placing a very small number of nodes (i.e. less than 6) is obviously constant.

Conclusions

We have presented a new visualization technique that could handle the visualization of very large graphs with up

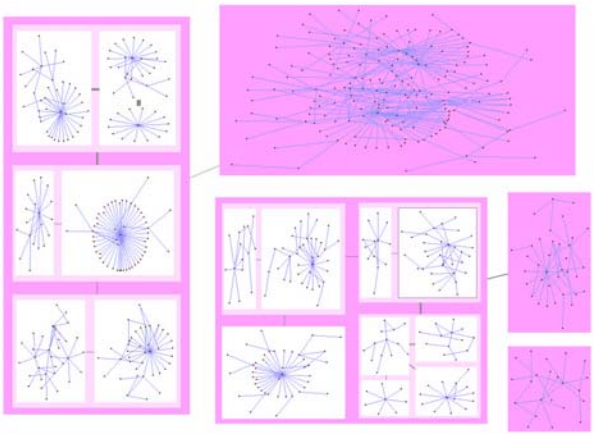

Figure 9. A navigational view of a multiple sub-clusters from Figure 8, including the top-center cluster, middle-center cluster, bottom-left cluster and two bottom-left sub-clusters from bottom-center cluster.

to tens thousands or even hundred thousands of elements on an ordinary Personal Computer. Our method includes two independent steps: clustering and visualization. The clustering step aims to reduce the visual complexity and enhance the comprehension of large graph layouts through the use of visual abstraction. We first discover an optimized community structure in a graph and divide it into densely connected clusters. We then use three levels of visual abstraction: 1) the *full context view*, 2) the *current context view* and 3) the *main view*, to display the large graph. The visualization uses a new space-efficient layout and navigation technique that can visualize effectively the large clustered graphs with several thousands of elements on a limited display space. We use a multiple view technique to archive the focus+context view navigation. The interactive animation is also employed to preserve the users' mental maps during the interaction.

Acknowledgements

This project is supported by Australia Research Council (Discovery Research Grant: DP0665463).

References

[1] J. Abello, F. van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), 2006.

[2] D. Archambault, T. Munzner, and D. Auber. Topolayout: Multi-level graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 12, 2006.

[3] D. Auber. Tulip: A huge graph visualisation framework. In *Proc. Graph Drawing Software, Mathematics and Visualization*, pages 105–126, 2003.

[4] D. Auber, Y. Chiricota, F. Jourdan, and G. Malancon. Multiscale visualization of small world networks. In *Proc. IEEE Symposium on Information Visualization*, pages 75–81, 2003.

[5] D. Auber and F. Jourdan. Interactive refinement of multiscale network clusterings. In *Proc. IV 2005*, pages 703–709, 2005.

[6] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[7] P. Eades and Q. Feng. Multilevel visualization of clustered graphs. In *Proc. Graph Drawing (GD'96)*, pages 101–112, 1996.

[8] J.-D. Fekete, D. Wang, N. Dang, A. Aris, and C. Plaisant. Overlaying graph links on treemaps. In *Proc. Information Visualization 2003 Symposium Poster Compendium*, pages 82–83, 2003.

[9] Q. Feng. *Algorithms for Drawing Clustered Graphs*. PhD Thesis, The University of Newcastle, 1997.

[10] S. Hachul and M. Junger. An experimental comparison of fast algorithms for drawing general large graphs. In *Proc. Graph Drawing (GD'05)*, pages 235–250, 2005.

[11] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. In *Proc. Graph Drawing (GD'00)*, pages 183–196, 2000.

[12] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. *Journal of Graph Algorithms and Applications*, 8(2):195–214, 2004.

[13] I. Herman, G. Melancon, and M. S. Marshall. Graph visualization in information visualization: a survey. *IEEE Transactions on Visualization and Computer Graphics*, 6:24–44, 2000.

[14] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proc. 1991 IEEE Visualization*, pages 284–291, 1991.

[15] S. Marshall. *Methods and tools for the visualization and navigation of graphs*. PhD Thesis, University Bordeaux I, 2001.

[16] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69:066133, 2004.

[17] Q. V. Nguyen and M. L. Huang. Space-optimized tree: a connection+enclosure approach for the visualization of large hierarchies. *Information Visualization Journal*, 2(1):3–15, 2003.

[18] Q. V. Nguyen and M. L. Huang. Enccon: An approach to constructing interactive visualization of large hierarchical data. *Information Visualization Journal*, 4(1):1–21, 2005.

[19] J. C. Robert. On encouraging multiple views for visualisation. In *Proc. International Conference on Information Visualisation (IV 1998)*, pages 8–14, 1998.

[20] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *Mathematics Research Report 00/IM/60, University of Greenwich*, pages 1–22, 2000.

[21] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 2004.

[22] A. J. Wilkins. *Visual Stress*. Oxford University Press, 1995.