Journal of Graph Algorithms and Applications

http://www.cs.brown.edu/publications/jgaa/

vol. 4, no. 3, pp. 157–181 (2000)

# Navigating Clustered Graphs using Force-Directed Methods

*Peter Eades*

Basser Department of Computer Science
University of Sydney
http://www.cs.usyd.edu.au/
peter@cs.usyd.edu.au

*Mao Lin Huang*

Department of Computer Systems
University of Technology, Sydney
http://www.socs.uts.edu.au/
maolin@soco.uts.edu.au

**Abstract**

Graphs which arise in Information Visualization applications are typically very large: thousands, or perhaps millions of nodes. Current graph drawing methods successfully deal with (at best) a few hundred nodes.

This paper describes a strategy for the visualization and navigation of graphs. The strategy has three elements:

1. A layered architecture, called `CGA`, for handling *clustered graphs*: these are graphs with a hierarchical node clustering superimposed.
2. An online force-directed graph drawing method.
3. Animation methods.

Using this strategy, a user may view an *abridgment* of a graph, that is, a small part of the graph that is currently of interest. By changing the abridgment, the user may travel through the graph. The changes use animation to smoothly transform one view to the next.

The strategy has been implemented in a prototype system called `DA-TU`.

Communicated by G. Liotta and S. H. Whitesides: submitted September 1998; revised July 2000.

## 1 Introduction

Graphs which arise in Information Visualization applications are typically very large: thousands, or perhaps millions of nodes. Recent graph drawing competitions [5] have shown that visualization systems for classical graphs are limited to (at best) a few hundred nodes.

Attempts to overcome this problem have proceeded in two main directions:

*Clustering.* Groups of related nodes are "clustered" into super-nodes. The user sees a "summary" of the graph: the super-nodes and super-edges between the super-nodes. Some clusters may be shown in more detail than others. An example is in Figure 1. Note that "New South Wales" is shown in more detail than "Victoria". The clustering approach has been taken by a number of graph drawing researchers [2, 6, 13, 15], and is related to the "overview diagrams" used by some web navigation facilities [12].

*Navigation.* The user sees only a small subset of the nodes and edges at any one time, and facilities are provided to navigate through the graph. This approach was taken by the OFDAV system [9].

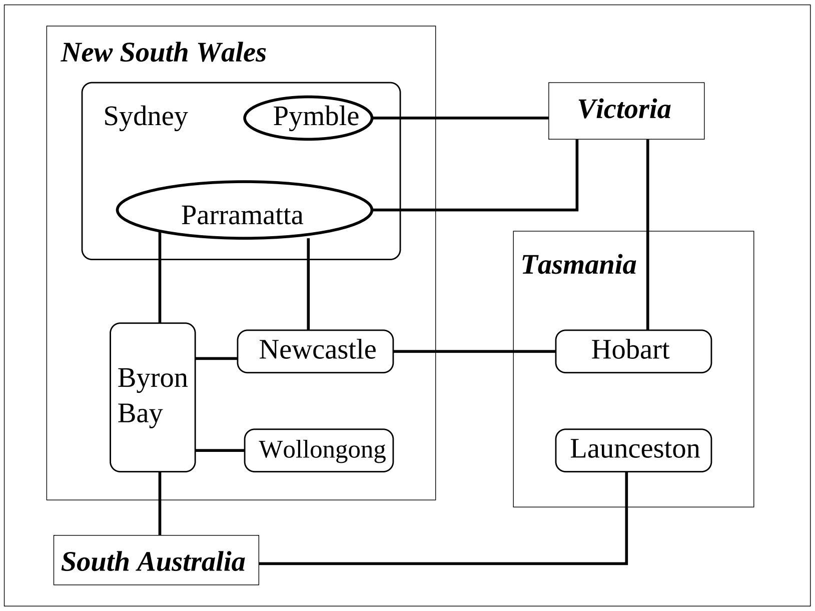

Figure 1: *A clustered graph.*

This paper introduces a strategy for combining the two approaches. The strategy has three elements:

1. A **layered architecture** for handling *clustered graphs*: these are graphs with a hierarchical node clustering superimposed (see [6]). The architecture, called `CGA`, is illustrated in Figure 2. This architecture supports *abridgments* of clustered graphs. These abridgments are logical views of parts of the clustered graph. Users may change their focus of interest by changing the abridgment. These changes are reflected in the *picture* of the abridgment. `CGA` is described in Section 2.

2. An online **force-directed graph drawing method**. This method operates at the *picture* layer of the architecture. It is a simple extension of the force-directed method from [9], described in Section 3.1.

3. **Animation methods.** Multiple animations are used to "preserve the mental map"[4], that is, to smooth the transition between pictures as the user changes focus. The animation methods are described in Section 3.2.

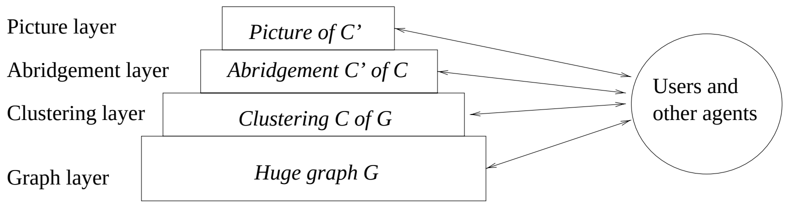

Figure 2: *The* `CGA` *architecture.*

Our strategy has been instantiated in a prototype system called `DA-TU`. Details of `DA-TU` as well as a static storyboard are in Section 4. An animated web storyboard is online at:
*http://www-staff.socs.uts.edu.au/˜ maolin/jgaa_demo/jgaa_demo.html*.

The main purpose of this paper is to demonstrate the *feasibility* of visualizing huge graphs (with more nodes than can fit on a screen) by a combination of clustering and navigation methods. We propose that the architecture described below provides a suitable framework, and that the force-directed drawing and animation methods are suitable tools. A thorough test of this hypothesis will take many years; this paper reports on the progress that we have made to date.

## 2 The Architecture

The architecture `CGA` (*Clustered Graph Architecture*) is a design for systems in which the user manipulates data in four layers, as illustrated in Figure 2. We describe the data and methods of these four layers below.

The main aim of `CGA` is to separate concerns in such a way that:

- The host system need not know the whole graph. In this way, the graph can be huge (for example, it could be the whole World-Wide-Web).
- Outside agents, such as clustering algorithms and graph drawing algorithms, can be employed.
- Expertise in different areas may be confined to different layers.

## 2.1 The graph layer

A graph in `CGA` is a classical undirected graph, consisting of nodes and edges. In applications it may be a very large graph, containing many thousands of nodes. The graph may be dynamic, that is, the node and edge set may be changing; these changes may be a result of user interaction through an interface, or they may be changed by an outside agent. Further, the nodes and edges may have application-specific *attributes*, such as labels and semantics.

The changes to a graph use basic operations as follows.

`G_new_node()`: adds a new node to the graph, and returns an identifier for that node to the sender.

`G_new_edge(`$u$`, `$v$`)`: adds a new edge (between existing nodes $u$ and $v$) to the graph, and returns an identifier for the new edge.

`G_delete_node(`$u$`)`: deletes node $u$ from the graph.

`G_delete_edge(`$e$`)`: deletes edge $e$ from the graph.

Further, an agent can request a neighborhood of a node:

`G_neighborhood(`$u$`)`: given a node $u$, this returns a list of neighbors of $u$.

Some more operations may be available to manage attributes of nodes and edges. For example, an elementary operation on the attributes of a node $u$ is:

`G_change_attribute(`$u$`, `*attribute_id*`, `*attribute_value*`)`: changes the attribute *attribute_id* to *attribute_value*.

The messages that invoke these operations may be sent and executed asynchronously, and thus, one can conceptually regard the graph as a database. If the whole graph is known, then it may be implemented by storing the graph in a database. However, in many applications the whole graph is not known (such as with web graphs), and a "graph server" implementation is appropriate.

## 2.2 The clustering layer

A *clustered graph* $C = (G, T)$ consists of an undirected graph $G = (V, E)$ and a rooted tree $T$ such that the leaves of $T$ are exactly the vertices of $G$. Each node $\nu$ of $T$ represents a *cluster* of vertices of $G$ consisting of the leaves of the subtree rooted at $\nu$. The tree $T$ describes an inclusion relation between clusters; it is the *cluster tree* of $C$.

Figure 1 shows a clustered graph $C = (G, T)$; here the cluster tree $T$ gives a geographical relation, and the graph $G$ shows communication lines. Figure 3 shows the underlying graph $G$ of $C$. Figure 4 shows the cluster tree $T$ of $C$.

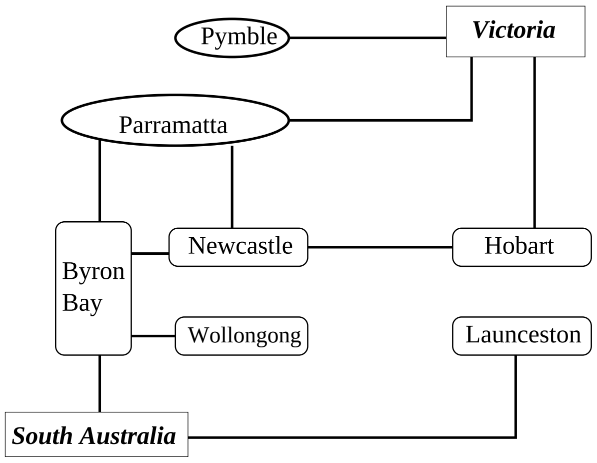

Figure 3: *Underlying graph of the clustered graph in Figure 1.*

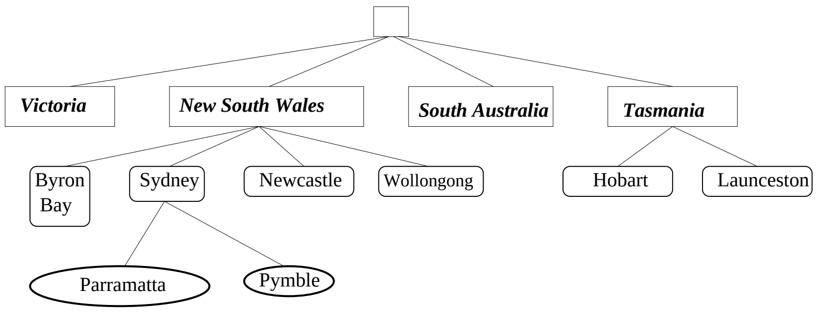

Figure 4: *Cluster tree of the clustered graph in Figure 1.*

Clustered graphs were introduced by Feng [6] as a model for relational structures with a node hierarchy. There are many closely related models; for example, the *compound graphs* of Sugiyama and Misue [16, 14] are more general; the *higraphs* of Harel [8] are far more general.

We need two elementary operations for clustered graphs.

`C_create_cluster(`$S$`: set of nodes)`: creates a new cluster of the set $S$ of nodes, and returns an identifier for the new cluster. As a precondition, all nodes in $S$ must be siblings in the cluster tree. The parent of the new cluster is the shared parent of $S$.

`C_destroy_cluster(`$u$`: node)`: The children of $u$ in the cluster tree become children of their grandparent, and $u$ is deleted. This operation may not be applied to the root, and applying the operation of a leaf $u$ merely deletes $u$.

These operations are illustrated in Figure 5.

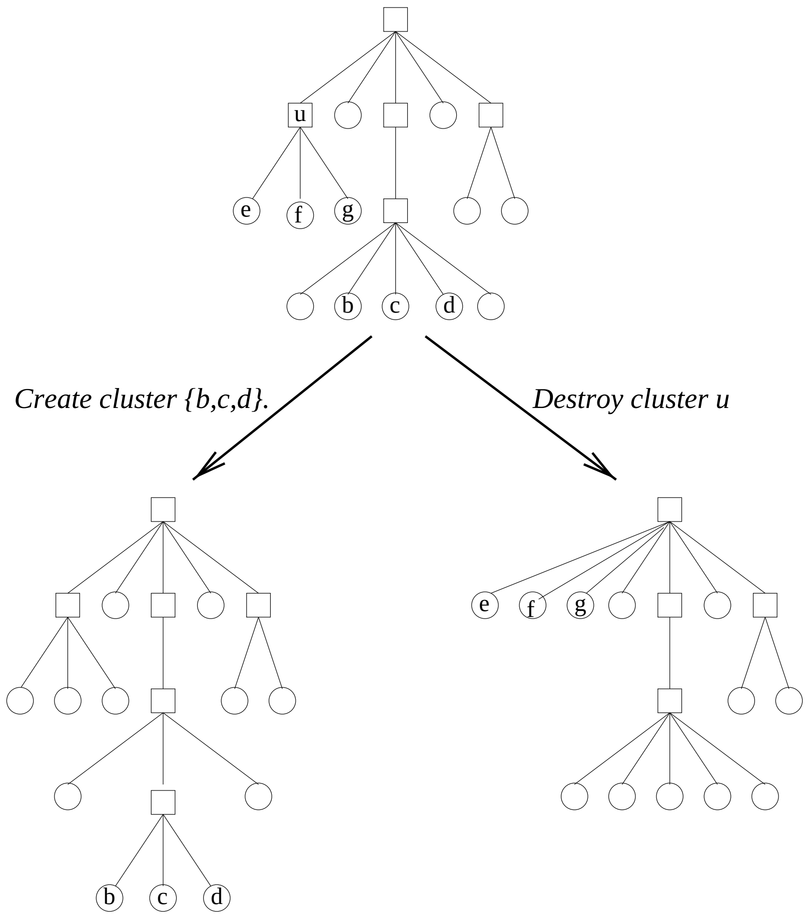

Figure 5: *Some operations at the cluster layer.*

Further, operations such as `G_new_node` at the graph layer have an effect on the cluster layer. When a new node is created at the graph layer, a message is sent to the cluster layer which makes that new node into a child of the root.

A special operation at the cluster layer is to *dismiss_clusters*; this destroys all non-leaf clusters except the root.

## 2.3 The abridgment layer

In applications, the whole clustered graph is too large to show on the screen; further, it is too large for the user to comprehend. This motivates the *abridgment layer*. An abridgment of the clustered graph in Figure 1 is shown in Figure 6.

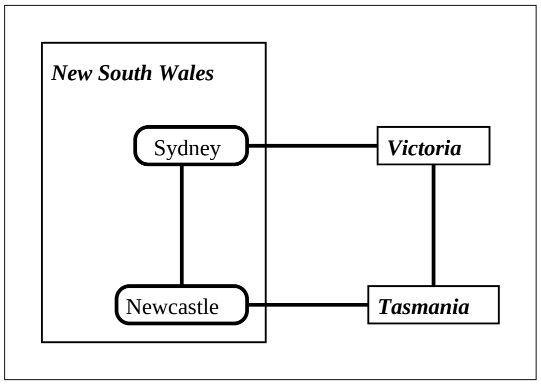

Figure 6: *An abridgment.*

We now give a formal definition of “abridgment”. Suppose that $U$ is a set of nodes of the cluster tree $T$. The subtree of $T$ consisting of all nodes and edges on paths between elements of $U$ and the root is called the *ancestor tree* of $U$. The set $U$ is called the *basis* of the ancestor tree. An example of an ancestor tree is in Figure 7.

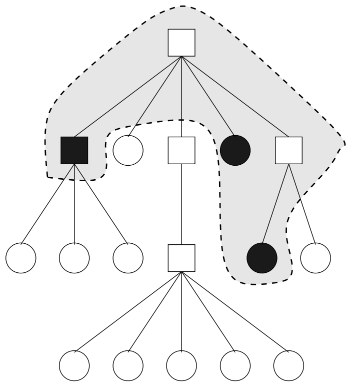

Figure 7: *The light shaded area is the ancestor tree of the dark shaded nodes.*

A clustered graph $C' = (G', T')$ is an *abridgment* of the clustered graph $C = (G, T)$ if $T'$ is an ancestor tree of $T$ with respect to a set $U$ of nodes of $T$, and there is an edge between two distinct nodes $u$ and $v$ of $G'$ if and only if there is an edge in $G$ between a descendent of $u$ and a descendent of $v$. The vertex set of $G'$ is the basis of the ancestor tree $T'$.

Figure 6 shows the abridgment of Figure 1 with basis {*Sydney, Newcastle, Tasmania, Victoria*}.

`CGA` has three elementary operations on abridgments; these change the basis of the abridgment.

> `open_cluster(`$u$`)`: This is defined whenever $u$ is in the basis $U$ of the current abridgment. It replaces $u$ by the children of $U$. The effect of this operation on the cluster tree is illustrated in Figure 8.
>
> `close_cluster(`$u$`)`: All children of $u$ in the basis of the current abridgment are deleted (from the basis), and $u$ is added to the basis. The effect of this operation on the cluster tree is illustrated in Figure 9.
>
> `hide(`$u$`)`: If $u$ is in the basis, then it is deleted from the basis. The effect of this operation on the cluster tree is illustrated in Figure 10.

Note that the `hide` operation is needed so that the children of a node can be selectively hidden (otherwise, if $u$ is in the basis then all siblings of $u$ would need to be in the basis).

As an example, the abridgment in Figure 6 may be obtained from Figure 1 by closing "Sydney" and "Tasmania", then hiding "Byron Bay", Wollongong" and "South Australia".

The operations at the lower layers have effects at the abridgment layer.

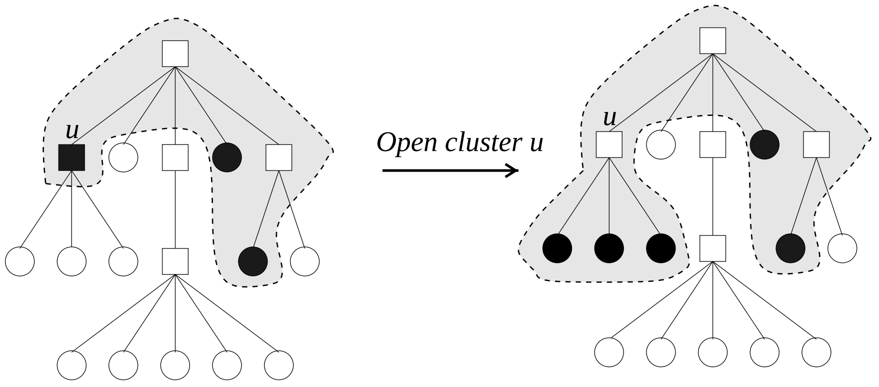

Figure 8: *The effect of the* `open_cluster` *operation on the cluster tree.*

## 2.4 The picture layer

The main purpose of `CGA` system is visualization; it shows animated pictures of abridgments of large clustered graphs. Note that the abridgment itself is

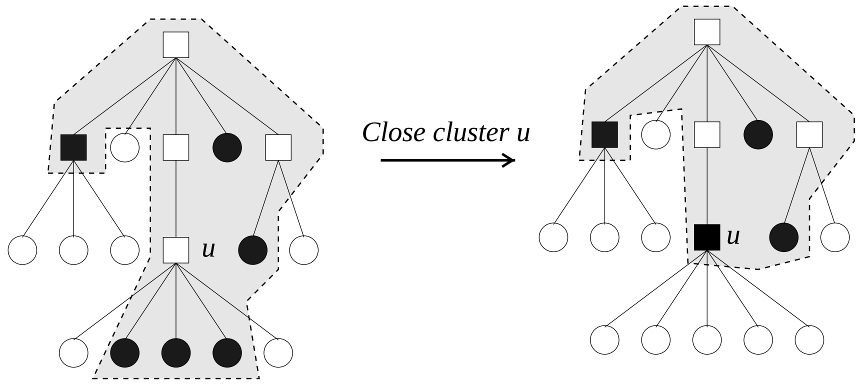

Figure 9: *The effect of the* `close_cluster` *operation on the cluster tree.*

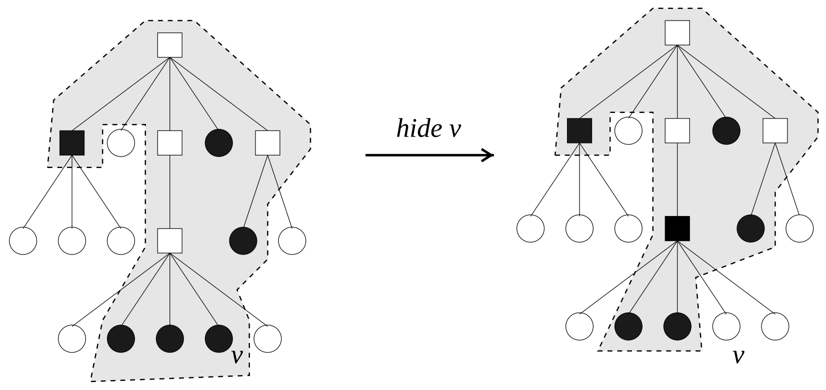

Figure 10: *The effect of the* `hide` *operation on the cluster tree.*

a clustered graph. Here we describe how `CGA` operates on static and dynamic pictures of clustered graphs.

Examples of static pictures of clustered graphs are shown in Figures 1 and 6; pictures of clustered graphs from the implementation `DA-TU` are in Section 4. The algorithms used by `DA-TU` to obtain these pictures are described in Section 3.

In `CGA`, a *static picture* of a clustered graph $C = (G, T)$ contains a location $p(v)$ for each vertex $v$ of $G$ and a route $c(u, v)$ for each edge $(u, v)$ of $G$, in the same way as drawings for classical graphs. Further, a static picture has a region $b(\nu)$ of the plane for each cluster $\nu$ of $T$, such that if $\nu$ is a leaf of $T$ then $b(\nu)$ is located at $p(\nu)$, and if $\mu$ is a child of $\nu$ in $T$ then $b(\mu)$ is contained in $b(\nu)$. This definition differs slightly from Feng's definition [6] of a drawing of a clustered graph: `CGA` allows overlaps between regions, Feng does not. In the implementation `DA-TU`, methods are used to avoid overlaps, but the architecture must allow for the existence of overlaps. The regions used by the implementation `DA-TU` are rectangles, and the edges are straight lines.

A *dynamic picture* of a clustered graph $C$ consists of a sequence $D_1, D_2, \ldots, D_k$ of static pictures of $C$. Each $D_i$ should differ from $D_{i-1}$ by very little, so the sequence appears as a smooth animation. Examples of this animation are in Section 4.

Changes at the graph layer, the cluster layer, and the abridgment layer all effect the dynamic picture in `CGA`. Whenever such a change occurs, `CGA` produces a new abridgment $C_{new}$, which differs from the previous abridgment $C_{old}$ by a small amount (a few nodes change). The picture layer of `CGA` reacts by producing a new static picture, called a *key frame*, of the new clustered graph $C_{new}$. Then, without user (or agent) interaction, `CGA` displays a dynamic picture $D_1, D_2, \ldots, D_k$ of $C_{new}$. This dynamic picture plays the role of "in-betweening" in animation; it continues until another change comes from a lower layer.

The picture layer has the following operations:

- `Move(u: node)`: this is the usual operation of manually moving a node in the picture.
- `Gathering`. This operation moves sibling nodes closer together. In practice, this tends to make the cluster boundaries disjoint. Examples of the gathering operation are in Section 4.
- `Scaling.` This operation can be used to increase the size of the picture to allow more detail to be seen, or reduce the size of the picture, to fit the screen.
- `Layout operations.` These are classical layout functions, which produce a picture $P'$ of a clustered graph $C'$ from a picture $P$ of a clustered graph $C$. The layout operation is used in two circumstances:
  1. When a change from $C$ to $C'$ occurs at the abridgment layer, the `layout` operation responds at the picture layer by producing a picture $P'$ of $C'$, computed from $C'$ and $P$.
  2. The layout operation is used to produce the dynamic pictures described above. That is, it produces picture $D_{i+1}$ from $D_i$.

  Details of the implementation of the layout operation in `DA-TU` are in the next section.

## 3 Layout and animation

There are two requirements for algorithms to produce pictures for `CGA`.

- We need to produce diagrams that are easy to read and understand. A great deal of research in *graph drawing algorithms* (see, for example, [1]) has resulted in a variety of such algorithms; these take a combinatorial description of a graph and attempt to produce a drawing that is as readable as possible.

- CGA aims to navigate through graphs. The logical operations at the lower layers must be reflected in changes to the drawing at the picture layer. These changes must be "smooth" at the picture layer; that is, each static picture $D_i$ should be very similar to $D_{i-1}$, and successive key frames should be similar.

For DA-TU we have used a force-directed method to achieve both requirements. Our methods are similar to those described in [9]. In Section 3.1, we describe the force model that drives this technique. Section 3.2 describes how animation is used to smooth the changes at the picture layer.

Note that other layout operations to satisfy the requirements above are possible; for an example, see [11].

## 3.1 The Force Model

In this section, we briefly outline the force model. It has been developed from several previous force models used in graph drawing [1, 3, 7, 10].

We use three types of spring forces. These operate between the vertices of the graph $G$ in the abridgment $C = (G, T)$.

- *Internal-spring:* A spring force between a pair of nodes in $G$ which are siblings in $T$.
- *External-spring:* A spring force between a pair of nodes in $G$ which are not siblings in $T$.
- *Virtual-spring:* Each cluster (non-leaf node of $T$) $\nu$ has a *virtual node* $\nu'$ associated with it. In the picture, $\nu'$ is a point within the rectangle representing $\nu$. Further, for each node $u$ in $G$, if the parent of $u$ in $T$ is $\nu$, then there is a *virtual edge* between $u$ and $\nu'$. A *virtual-spring* is a spring force between $u$ $\nu'$ along this virtual edge.

It is best to describe these spring forces with an example; see Figure 11. The internal-spring forces on vertex $c$ are along the edges $(c, a)$ and $(c, b)$; the external-spring forces on $c$ are along the edges $(c, f)$ and $(c, g)$.

Virtual-springs can be described using a virtual node in each cluster. In the clusters $X$, $Y$, and $Z$, virtual nodes $x'$, $y'$, and $z'$ are shown; each virtual node is connected to each node in its cluster by a virtual edge. Virtual-springs exert forces along these edges. Note that virtual nodes and edges are not shown in the actual picture of the clustered graph (unless the user wants to see them).

The internal-spring forces are stronger than the external-spring forces. This tends to make siblings come close together. This effect is increased by the virtual-spring forces. The *gathering* operation strengthens the virtual-spring

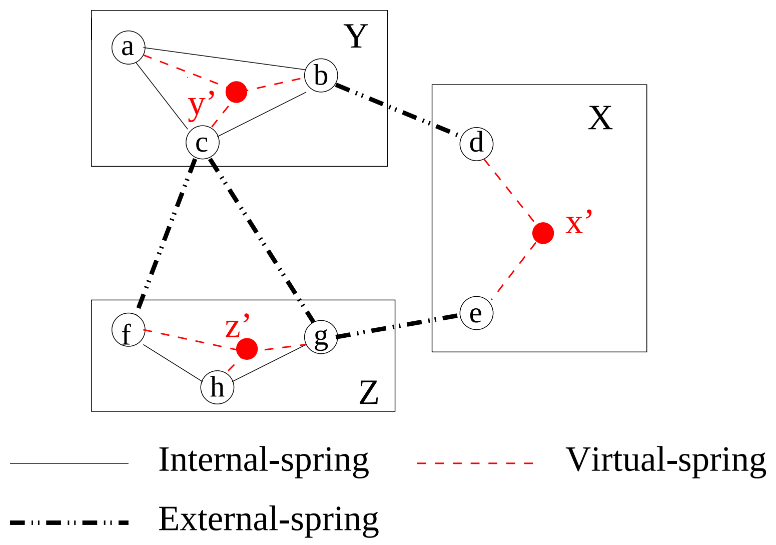

Figure 11: *Spring forces.*

forces and the internal-spring forces; this makes sibling nodes move closer together.

As well as spring forces, there are gravitational repulsion forces between all nodes.

The forces are applied additively to give an aesthetically pleasing layout of the graph. The sum of forces on each node is continually computed, and the nodes move according to the strength and direction of these forces. If the force on each node is zero, then the system has reached a locally minimal energy state, and the movement stops.

## 3.2 Animations

In `DA-TU`, the whole visualization is fully animated. Every change at one of the lower layers, whether triggered by the user or by another agent, is animated at the picture layer.

This animation may reduce the cognitive effort of the user in recognizing change; we aim to preserve the user's "mental map" [4].

There are several types of animation that are implemented in our system. These are detailed below.

- *Force animation.* The force model above is used in the animation of all changes to picture. The forces move the nodes toward a minimum energy state, and this movement gives the sequence $D_1, D_2, \ldots, D_k$ of drawings to smoothly change the layout from abridgment to the next. Overall, the animation driven by the forces is the most important mechanism for the smooth transition between changes at the layers lower than the picture layer.

- *Animated addition and deletion* of nodes and edges. When a node is deleted or added to the abridgment, we use animated *shrinking* or *growing* to help the user identify nodes that are disappearing or appearing. Disappearing nodes lose their connecting edges, then shrink until they disappear. Appearing nodes are displayed in a reduced size with their edges, then grow to be full-sized nodes. This avoids the sudden disappearing/appearing of node images on the screen, which can disturb the user's mental map.

- *Animated cluster regions.* Nodes move according to the force model, and the rectangular regions representing clusters follow the nodes. At all times, each cluster is the minimum enclosing rectangle of its children. These rectangles move smoothly as the positions of the children change. This is especially important for the gathering operation; the clusters separate smoothly.

- *Animated scaling.* Animated shrinking scales down a picture if it gets too large for the screen. Further, animated enlarging is used to increase the size of the picture; this enables the user to see details and makes direct manipulation operations of the picture easier. In fact, animated scaling is implemented by simply changing the forces in the springs; for example, to shrink the whole picture, every spring is increased in strength.

- *Animated cluster closing/opening:* When closing a cluster, we firstly use animation to reduce the size of closed region of this cluster. As soon as the size reaches a certain threshold layer we smoothly replace the representation of the cluster from its opened form (a red line bounded rectangle) to its closed form (a small black line bounded rectangle).

  When opening a cluster, we firstly smoothly replace the representation of the cluster from its closed form to its opened form. The children and relevant links of the cluster are smoothly added into the layout. Then we smoothly enlarge the size of this cluster to its normal size.

- *Camera animation.* Camera animation moves the whole drawing. It is optional. It can be used, for example, to move specific nodes of interest to the center of the screen.

All these animations operate in parallel.

## 4 Implementation

The strategy described in Sections 2 and 3 has been implemented in a prototype system called `DA-TU`[1]. A screenshot from `DA-TU` is in Figure 12; from `DA-TU`, the

[1] "Big map" in Mandarin.

user can call many of the operations described in Section 2. Note that operations at the graph layer are not implemented.

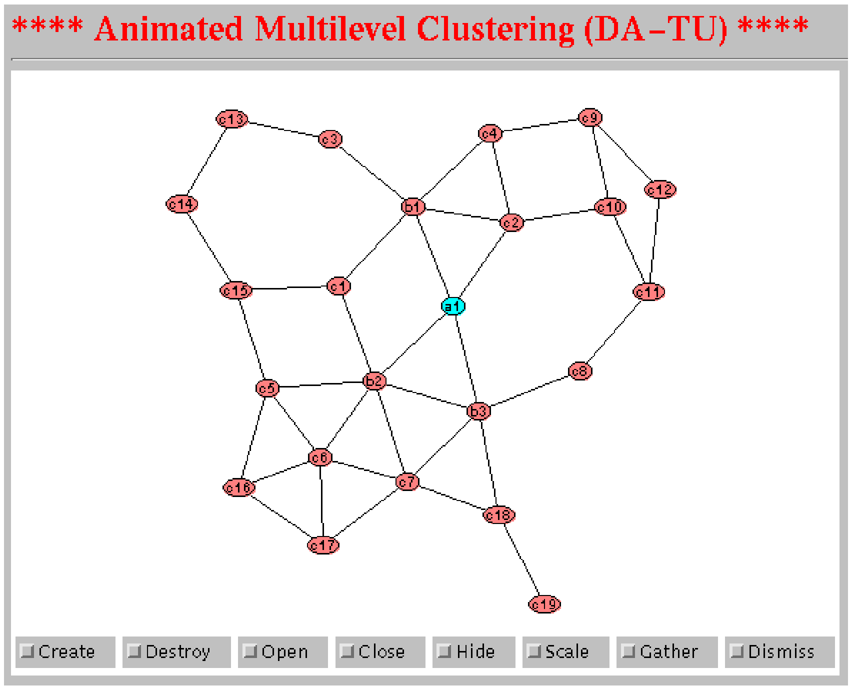

Figure 12: A screen from `DA-TU`.

This section records interactive sessions with `DA-TU`. They illustrate the basic operations of the system and how it works to achieve a better quality of the layout of clustered graphs.

## 4.1 Session one

The first session is a storyboard applet at:
*http://www-staff.socs.uts.edu.au/˜ maolin/jgaa_demo/jgaa_demo.html*.
using this applet, the reader may step through several types of operation. This session has nine steps:

1. A graph in DA-TU. The animated layout is computed by a spring algorithm. Note node $a1$ is fixed in the middle of the screen; it is the current focus.

2. The user creates five clusters. However, there are some overlaps among the clusters.

3. The user applies multiple forces in the 'gathering' operation to eliminate the overlaps. This may improve the readability of the layout. Note the 'virtual edges' and 'virtual node' in the cluster to the far left. This holds the nodes within the cluster together.

4. The same layout as shown in last one; however, the virtual edges and node are invisible. However, the 'virtual spring' applied between non-adjacent vertices $c8$ and $c19$ still operates.

5. Two clusters, $Cluster\_1$ and $Cluster\_4$ are 'closed' by clicking on regions occupied by these two clusters, in the 'close' mode. The user also applies a little 'gathering' to keep the clusters disjoint.

6. Open the cluster $Cluster\_4$ by clicking on the cluster in the 'open' mode. Some more 'gathering' is applied to make the rectangles disjoint.

7. Close the cluster $< b2, c5, c6, c7, c16, 17 >$ by clicking on its region, in the 'close' mode.

8. Open the clusters $Cluster\_1$ and then $Cluster\_3$. This is done by clicking on each of the clusters, one by one, in the 'open' mode. Again, some 'gathering' is applied.

9. Dismissing the whole cluster tree from the graph. This is done by clicking on the 'dismiss' button in control panel.

### 4.2 Session two

The second session consists of Figures 13 to 20. This session shows operations on the graph in Figure 12.

The first few figures show the results of operations *create_cluster* and *gathering*, followed by an illustration of the virtual springs, and the *close_cluster* operation.

### 4.3 Session three

The next session begins with Figure 16. The *gathering* operation, some *close_cluster* operations, and an *open_cluster* operation are shown. Finally, a special operation *dismiss_clusters* is shown.

## 5 Conclusions

CGA provides an architecture for handling graphs visually. The architecture is aimed at handling huge graphs, which may be too large to fit onto the screen. The user begins with a picture of an abridgment of the graph. This abridgment may be changed by user interaction or by an outside agent. Navigation through the graph is basically moving between abridgments.

The architecture has been implemented in a prototype system `DA-TU`. This provides a simple interface to the functions of `CGA`. It also uses a force-directed animated layout algorithm to ensure that the picture is aesthetically pleasing, and that transitions between pictures do not destroy the "mental map" [4].

Although `DA-TU` provides a proof of the main concepts of `CGA`, it is far from complete. Our future plans include the implementation of a graph drawing system within an application domain (such as web graphs) in which huge graphs commonly occur.

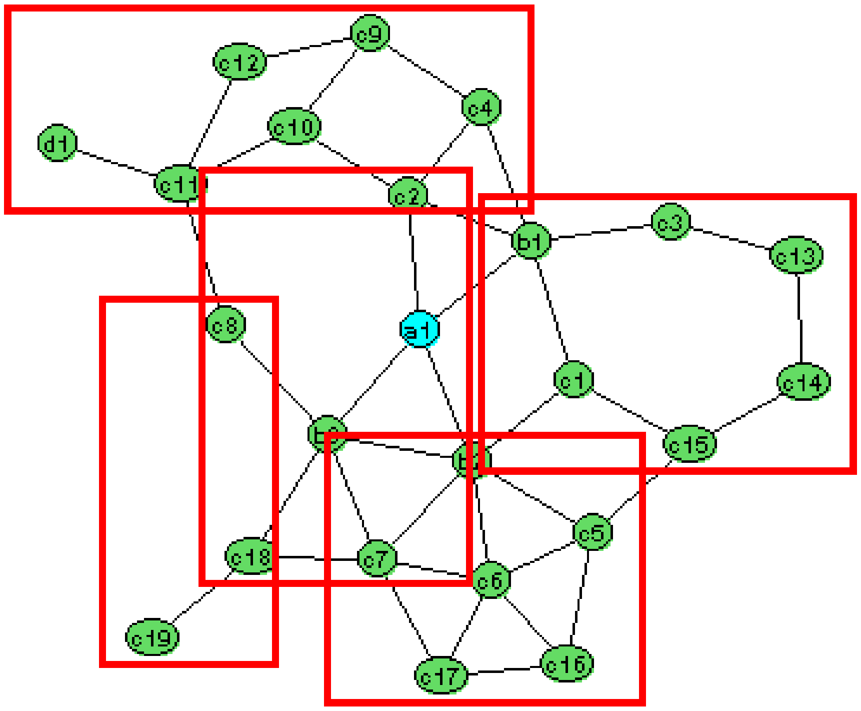

Figure 13: The user creates five clusters on the graph in Figure 12. However, this layout has five overlaps among the clusters.

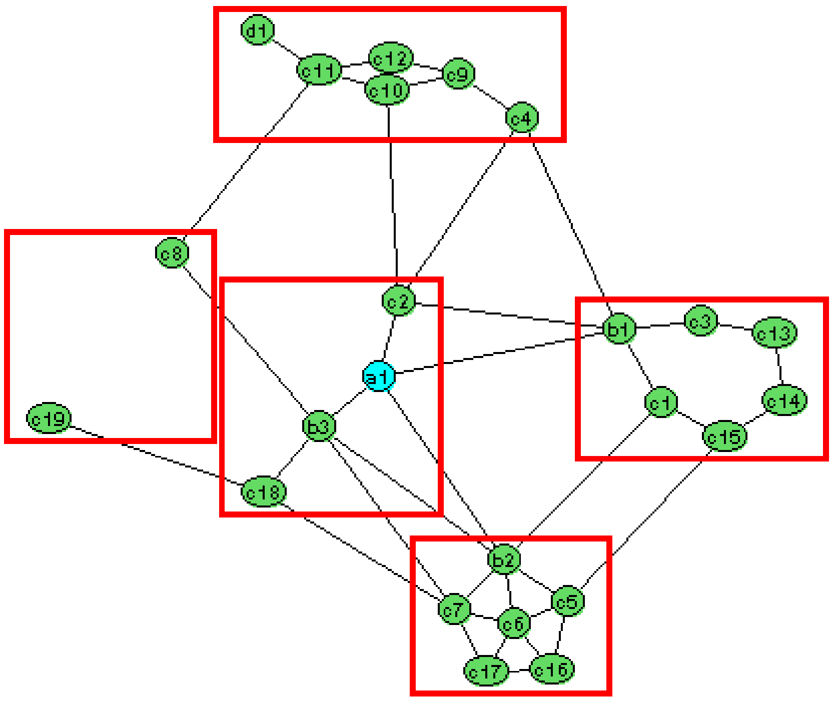

Figure 14: The user applies the spring forces in the "gathering" operation to eliminate the overlaps in Figure 13. This greatly improves the readability of the layout for the user.

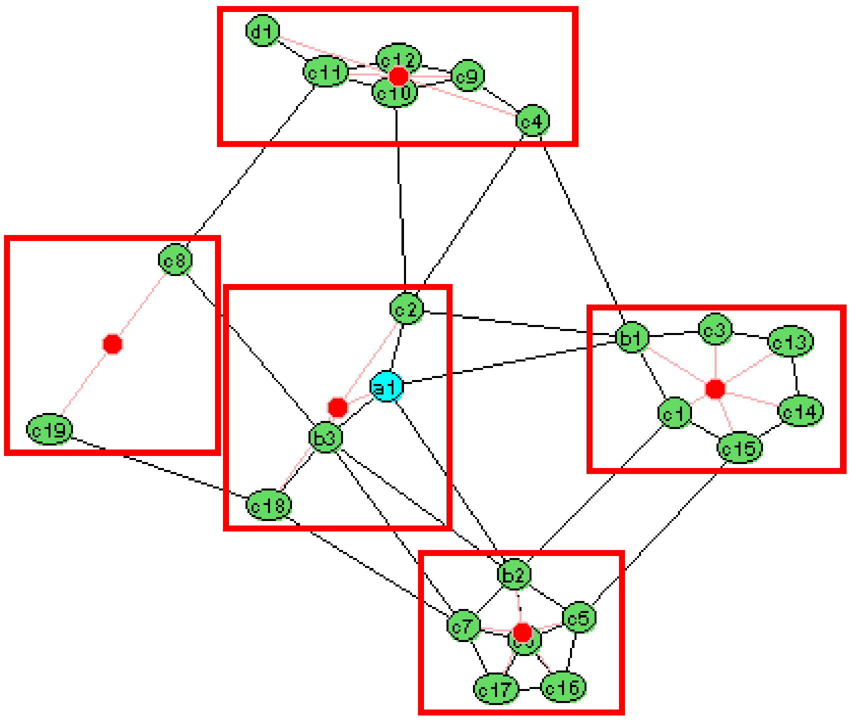

Figure 15: The same layout as shown in Figure 14; however, the virtual nodes and edges are shown. Note the "virtual springs", that is, an attractive force applied between non-adjacent vertices $c8$ and $c19$.

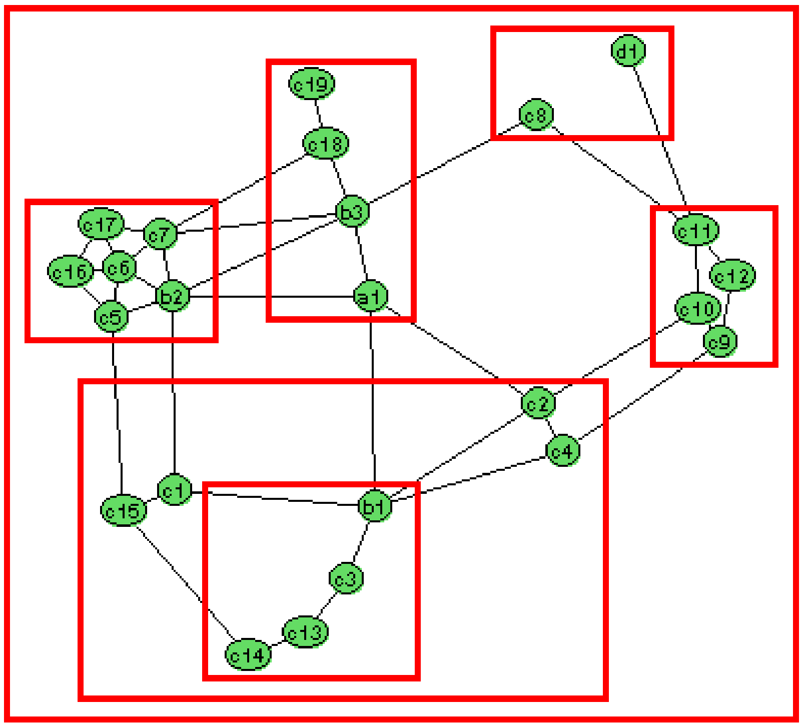

Figure 16: A clustered graph $C$ with 4 layers, after the application of spring forces in the “gathering” mode.

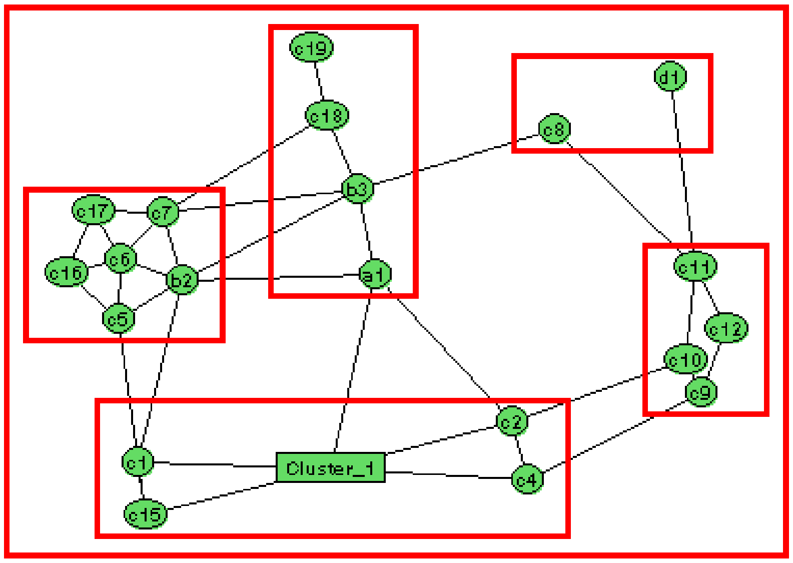

Figure 17: Closing one cluster, $\nu = \{b1, c3, c13, c14\}$, in Figure 16 by clicking on $\nu$ in the "close" mode. The representation of the closed cluster $\nu$ is a small rectangle.

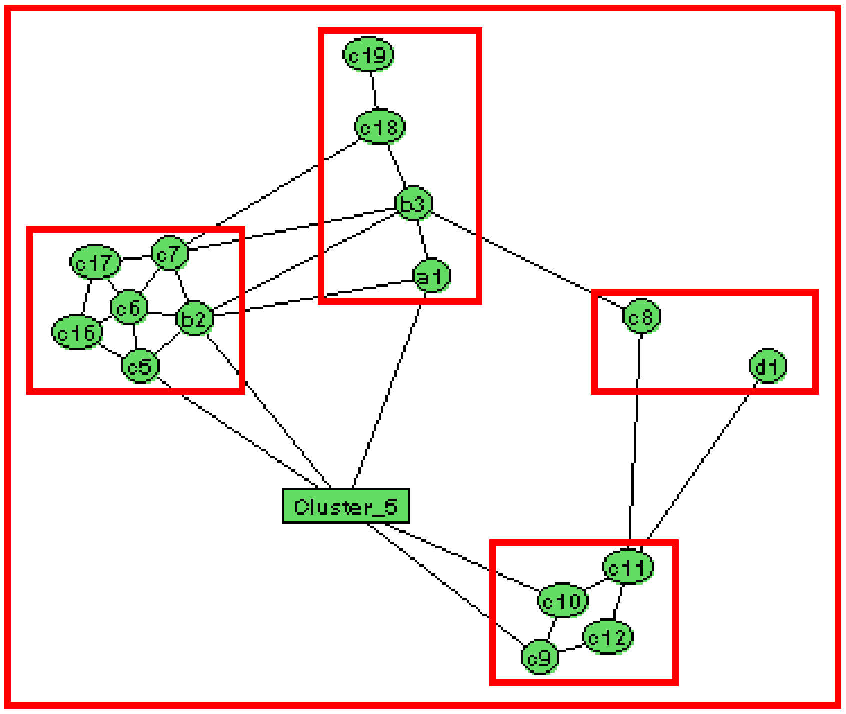

Figure 18: Continuing, closing the cluster $\{c1, c2, c4, c15, Cluster\_1\}$ in Figure 17.

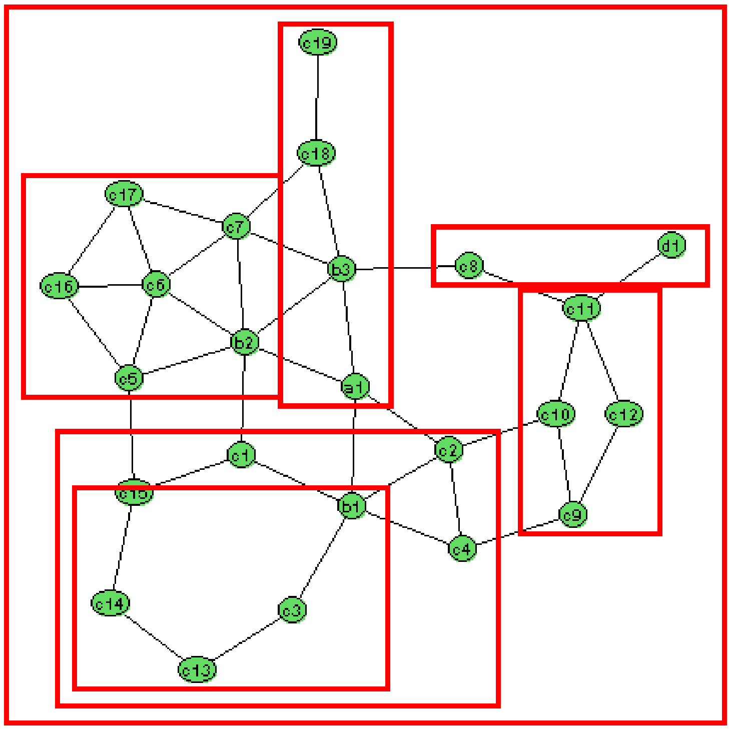

Figure 19: Opening the cluster *Cluster*_5 in Figure 18, and then opening the cluster *Cluster*_1. This is done by clicking on both visual rectangles of two clusters one by one under in the "open" mode.

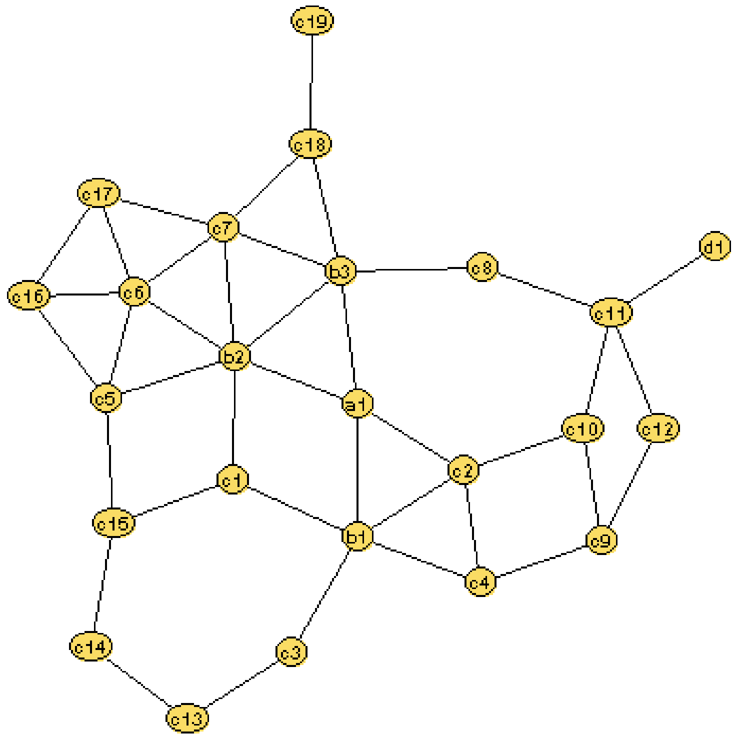

Figure 20: Dismissing the whole cluster tree from the layout as shown in Figure 19, and returning to the graph with no clusters. This could be done by destroying the clusters one by one; however, a "dismiss" button is provide to destroy all clusters at once.

## References

[1] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. Graph drawing - algorithms for the visualization of graphs. *Prentice-Hall, Englewood cliffs, NJ*, 1999.

[2] F. Bertault and M. Miller. Graph drawing 99. In *An algorithm for drawing compound graphs*, 1999. (to appear).

[3] P. Eades. A heuristic for graph drawing. *Congr. Numer.*, 42:149–160, 1984.

[4] P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. In *Proceedings of Compugraphics 91*, pages 24–33, 1991.

[5] P. Eades and J. Marks. Graph drawing contest report. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 143–146. Springer-Verlag, 1995.

[6] Q. Feng. Algorithms for drawing clustered graphs. In *PhD thesis, Department of Computer Science and Software Engineering, The University of Newcastle, Australia.*, 1997.

[7] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.

[8] D. Harel. On visual formalisms. *Commun. ACM*, 31(5):514–530, 1988.

[9] M. L. Huang, P. Eades, and J. Wang. On-line animated visualization of huge graphs using a modified spring algorithm. *Journal of Visual Languages and Computing*, 9(4), No. vl980094:623–645, 1998.

[10] T. Kamada. Symmetric graph drawing by a spring algorithm and its applications to radial drawing, 1989.

[11] S. Moen. Drawing dynamic trees. *IEEE Software*, 7:21–8, 1990.

[12] S. Mukherjea and J. Foley. Navigational view builder: A tool for building navigational views of information spaces. In *ACM SIGCHI '94 Conference Companion, pages 289-290*, 1994.

[13] S. C. North. Drawing ranked digraphs with recursive clusters. *preprint, Software Systems and Research Center, AT&T Laboratories.*, 1993.

[14] G. Sander. Layout of compound directed graphs. In *Technical Report A/03/96, University of the Saarlands*, 1996.

[15] K. Sugiyama. A cognitive approach for graph drawing. *Cybernetics and Systems: An International Journal*, 18:447–488, 1987.

[16] K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man and Cybernetics*, 21(4):876–896, 1991.