A CIRCULAR LAYOUT ALGORITHM FOR CLUSTERED GRAPHS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Mehmet Esat Belviranlı

August, 2009

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Uğur Doğrusöz (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. İsmail Hakkı Toroslu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Ali Aydın Selçuk

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

A CIRCULAR LAYOUT ALGORITHM FOR CLUSTERED GRAPHS

Mehmet Esat Belviranlı
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Uğur Doğrusöz
August, 2009

Visualization of information is essential for comprehension and analysis of the acquired data in any field of study. Graph layout is an important problem in information visualization and plays a crucial role in the drawing of graph-based data. There are many styles and ways to draw a graph depending on the type of the data. Clustered graph visualization is one popular aspect of the graph layout problem and there have been many studies on it. However, only a few of them focus on using circular layout to represent clusters. We present a new, elegant algorithm for layout of clustered graphs using a circular style. The algorithm is based on traditional force-directed layout scheme and uses circles to draw each cluster in the graph. In addition it can handle non-uniform node dimensions. It is the first algorithm to properly address layout of the quotient graph while considering inter-cluster relations as well as intra-cluster edge crossings. Experimental results show that the execution time and quality of the produced drawings with respect to commonly accepted layout criteria are quite satisfactory. The algorithm has been successfully implemented as part of Chisio, version 1.1. Chisio is an open source general purpose graph editor developed by i-Vis (information visualization) Research Group of Bilkent University.

Keywords: Information Visualization, Graph Visualization, Graph Drawing, Force Directed Graph Layout, Clustered Graphs, Circular Graph Layout.

ÖZET

KÜMELENMİŞ ÇİZGELER İÇİN ÇEMBERSEL YERLEŞİM ALGORİTMASI

Mehmet Esat Belviranlı
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Uğur Doğrusöz
Ağustos, 2009

Bilgi görselleme çeşitli çalışma alanlarından elde edilen verilerin anlaşılması ve analizi açısından oldukça önemlidir. Çizge yerleşimi ise bilgi görsellemede önemli bir problemdir ve çizge tabanlı bilgilerin görsellenmesinde önemli rol oynar. Bilginin türüne bağlı olarak çizgeyi çizmenin pek çok tarz ve yöntemi vardır. Kümelenmiş bilgi görselleme, çizge yerleşim probleminin popüler bir alanıdır ve konu üzerinde pek çok çalışmalar olmuştur. Fakat bu çalışmalardan çok azı kümeleri ifade etmek için dairesel yerleşim üzerine yoğunlaşmışdır. Bu çalışmada, kümelenmiş çizgelerin dairesel tarzda yerleşimi için yeni bır algoritma sunulmaktadır. Algoritma, geleneksel güce-dayalı yerlesim şablonunu esas almakta ve her bir kümeyi çizmek için daireler kullanmaktadır. Ayrıca değişebilir düğüm büyüklüklerini desteklemektedir. Kümeler arası ve aynı zamanda da küme içi kenar kesişimlerini göz önünde tutarak bölüm çizgesinin (küme düğümlerinin oluşturduğu çizge) yerleşimini ele alan ilk algoritmadır. Deneysel sonuçlar, hesaplama zamanı ve genelde kabul edilen yerleşim niteliği açısından algoritmanın son derece başarılı olduğunu ortaya koymaktadır. Algoritma Chisio'nun (sürüm 1.1) bir parçası olarak başarıyla uygulanmıştır. Chisio, Bilkent Üniversitesi i-Vis (bilgi görselleme) Araştırma Gurubu tarafından geliştirilmiş açık kaynak kodlu ve genel amaçlı bir çizge düzenleyicidir.

Anahtar sözcükler: Görselleme, Çizge Görselleme, Çizge Çizimi, Çizge Yerleşimi, Güce-dayalı Çizge Yerleşimi, Kümelenmiş Çizgeler, Çembersel Çizge Yerleşimi.

Acknowledgement

I would like to express my gratitude to my supervisor Assoc. Prof. Dr. Uğur Doğrusöz for his efforts in the supervision of the thesis.

I would like to express my special thanks to Prof. Dr. İsmail Hakkı Toroslu and Asst. Prof. Dr. Ali Aydın Selçuk for showing keen interest to the subject matter and accepting to read and review the thesis.

I would also like to express my gratitude to The Scientific and Technological Research Council of Turkey, TUBITAK for their extensive support during my two years of M.S. study.

Contents

List of Figures

Chapter 1

Introduction

A graph is an abstract structure that is used to model relational information. Many information visualization systems require graphs to be drawn so that information being modeled becomes human interpretable [13].

There are various graphical representations for graphs. Usually, vertices are represented by symbols such as points, boxes or ellipses and edges are represented by curves connecting the symbols that represent the associated vertices [6]. However, graphical representations vary greatly according to the application domain. Even within a graphical representation schema, there are infinitely many ways to draw a graph, by simply changing coordinates of nodes in the plane [13].

When drawing a graph, we would like to take into account a variety of aesthetic criteria. For example, planarity and the display of symmetries are often highly desirable in visualization applications [6]. In general, in order to improve the readability of drawings, it is important to keep the number of crossings and bends low. Also, to avoid wasting of space on screen or page, it is important to keep area of the drawing as small as possible. Trade-offs are often necessary as these are conflicting objectives [13].

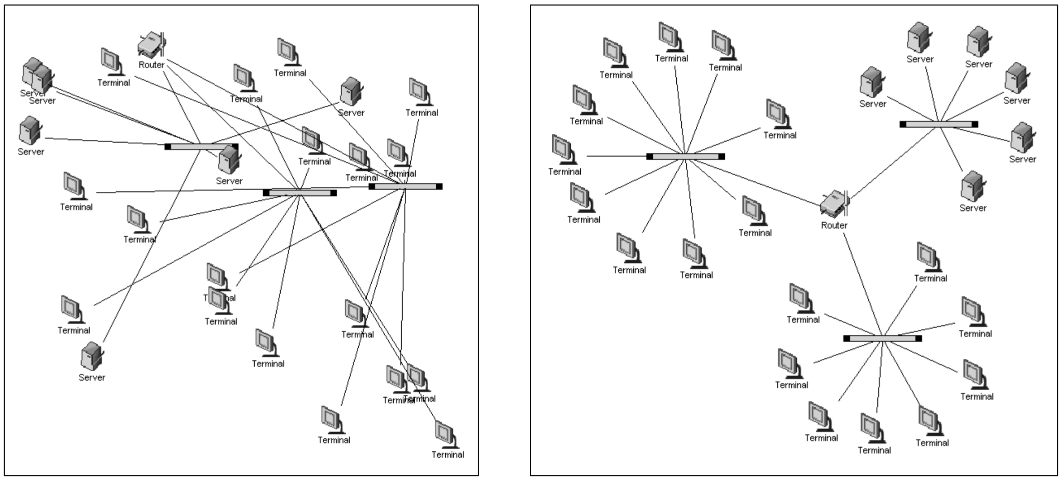

Figure 1.1: Two drawings of the same computer network system [13].

1.1 Aesthetics

Aesthetics is a subjective term, however it is possible to formalize it in our context. An aesthetic property specifies a geometric asset of underlying graph that we would like to highlight as much as possible. Commonly adopted aesthetics are [6]:

- *Crossings:* Minimization of edge-edge crossings is one of the most important aesthetic criteria. Ideally we would like to have crossing free drawings, however non-planar graphs do not admit one. Node-edge crossings should also be minimized, although they are not as important as edge-edge crossings.
- *Overlaps:* Minimization of node-node overlaps is another important aesthetic criterion.
- *Uniform Edge Length:* Minimization of the variance of the lengths of the edges.
- *Symmetry:* Maximize displayed symmetries in the graph.
- *Area:* Minimization of the total area of the drawing. The ability to generate drawings that use screen area efficiently is very important as screen

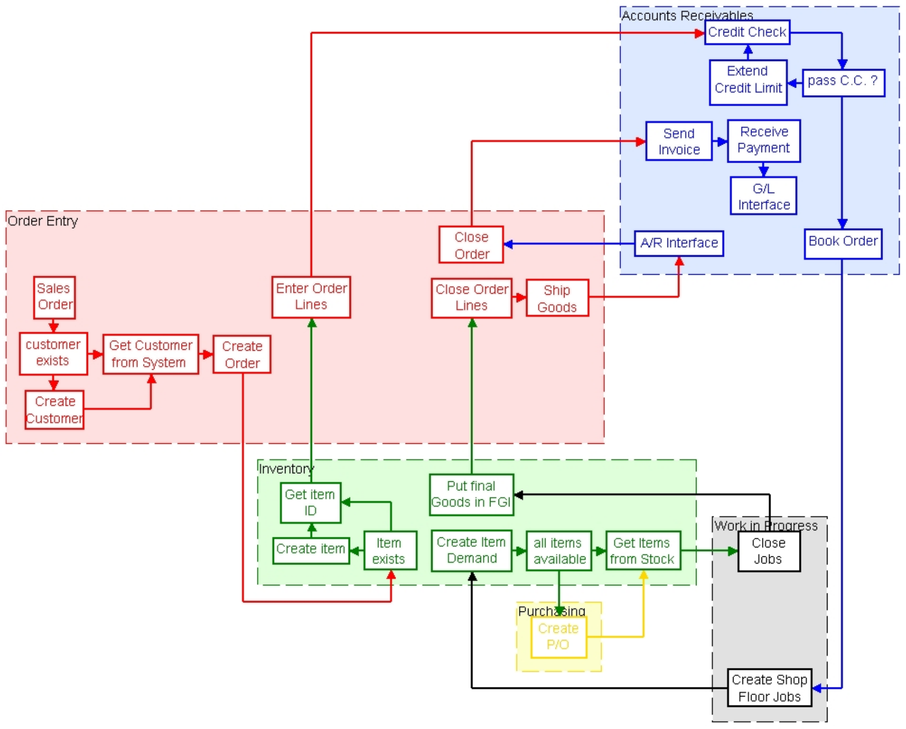

Figure 1.2: A drawing that displays the processes running inside an online shopping system, http://www.oreas.com.

space is an important and generally very limited resource for visualization applications.

- *Separation of Clusters:* In clustered graphs, clusters should be clearly separated from each other. Similarly, nodes inside a cluster should be close to each other as much as possible in order to emphasize the grouping between them.

Trying to satisfy all of these criteria is generally infeasible if not impossible, as they are inherently conflicting. So one has to prioritize according to needs of a particular application.

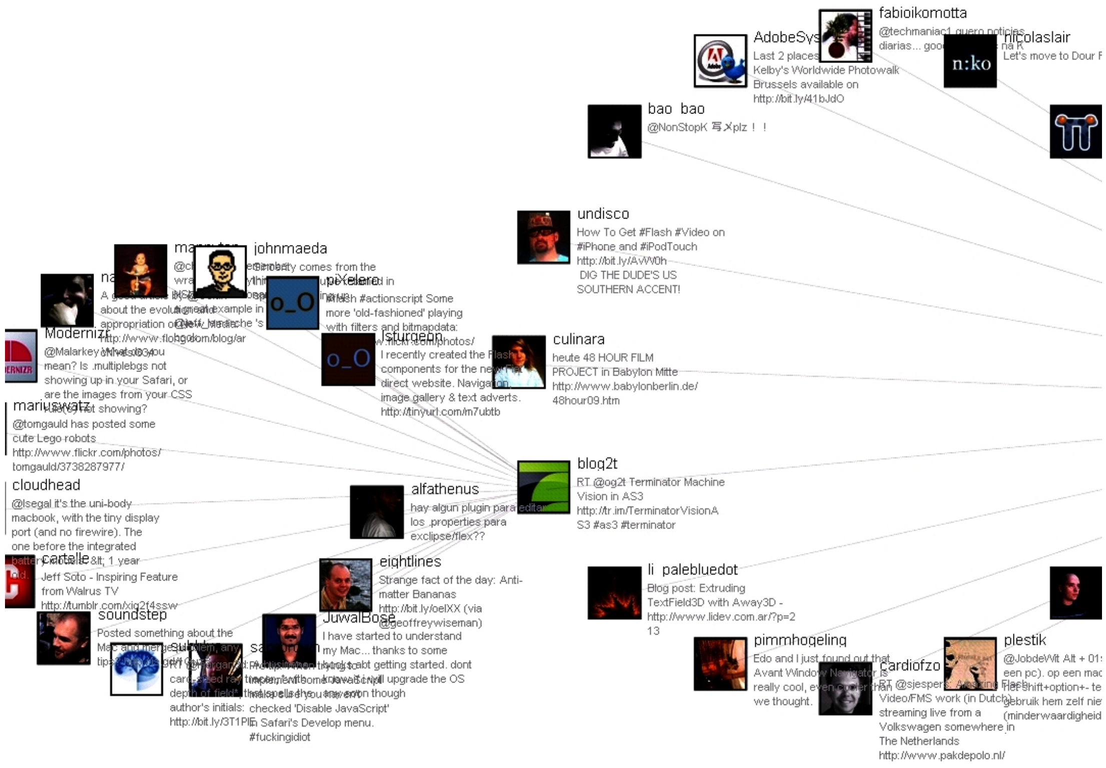

Figure 1.3: A circular drawing, which displays relationship between people, produced by a social network visualization tool, http://www.neuroproductions.be/twitter_friends_network_browser.

1.2 Visualization of clustered data

Clustering is the assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar in some sense. Clustering of data is a commonly used practice in many fields including bioinformatics, computer networks, machine learning, data mining, statistics, VLSI design, image analysis and social networks (Figures 1.2 through 1.5) [31]. Since clustering divides related information into groups, it provides a good means of handling size complexity for those fields which are obliged to deal with large amounts of data.

Clustering is used as a method of complexity management also in graph visualization. Since the data is split into smaller parts, it is easier to visualize the data in small partitions instead of a large chunk, therefore reducing the complexity of the drawing process. On the other hand clustering puts further constraints on the visualization requirements. The drawing should keep the clusters together and tight as well as neatly displaying the relations between each of them, thus increasing the readability of the graph.

In the past years, there have been many studies on clustered layouts [24]. However, most of these studies focus more on cluster generation using techniques such as geometric clustering [9, 20, 21, 27, 29] and graph theoretic clustering [5, 9, 19, 20, 21, 25, 27]. Only a few of them actually consider the layout of an already clustered graph by using orthogonal and straight line drawings [10, 11]. However these methods are not sufficient to separate clusters hence cannot display clustered data clearly. On the other hand, in industry, various tools which handle cluster graph layout exist [1, 2, 17, 30, 33]. Besides the algorithms specifically targeted at clustering, compound graph layout algorithms can also be specialized to visualize clusters by drawing each cluster into separate compounds with a fixed depth level of one [23, 24]. Comparing all of these algorithms, clustered data can be represented with circular style best in terms of aesthetics criteria.

Effective analysis of the underlying data in graph visualization is only possible with sound automatic layout capabilities of such systems. In this thesis, we present a new algorithm for automatic layout of clustered graphs in circular fashion. The algorithm is unique in the sense that it properly addresses layout of the quotient graph (the graph composed of clusters and their relations) while considering inter-cluster relations as well as intra-cluster edge crossings. This is achieved with a novel approach that applies force directed schemes on circular drawings. The algorithm is successfully implemented as a part of Chisio, version 1.1. Chisio is developed and released by i-Vis (information visualization) Research Group of Bilkent University.

The rest of the thesis is organized as follows: Definitions, Related Work, Layout Algorithm, Implementation, Experimental Results and Conclusion.

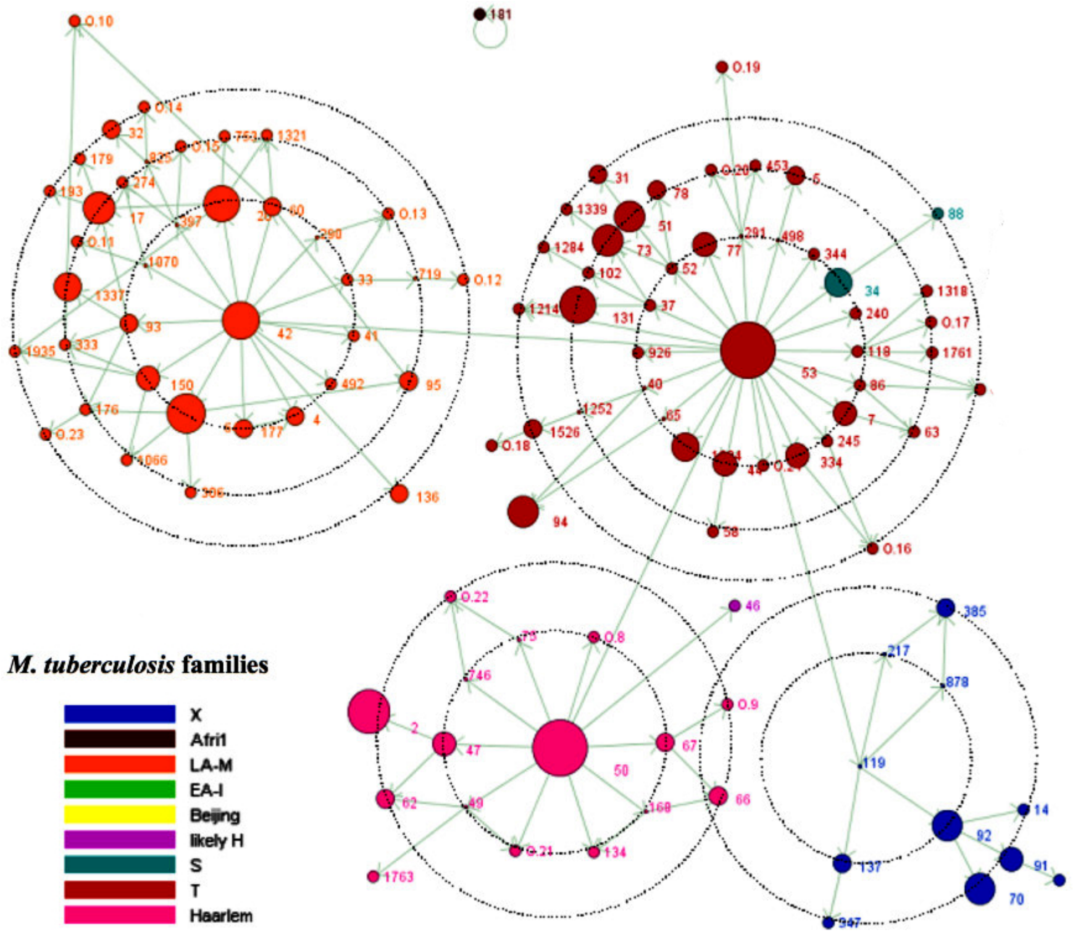

Figure 1.4: Cluster graph drawn from spoligotyping data of 344 TB cases censused in French Guiana over the 1996 - 2003 period [14].

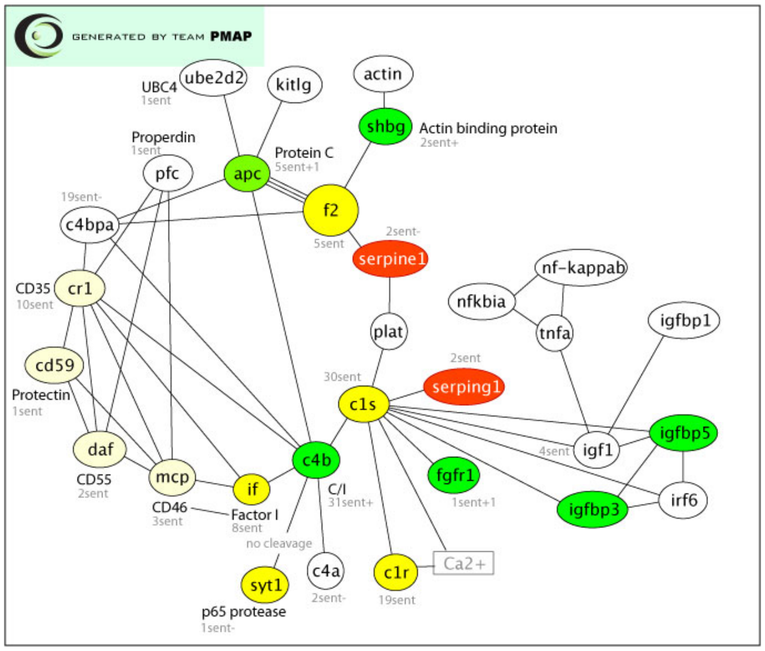

Figure 1.5: Complement activation and regulation network drawn by yFiles circular layout [33], http://www.proteolysis.org/proteases/m_goto_network/net4_0908

Chapter 2

Definitions

A *graph* G is defined by two finite sets V and E, where the elements of V are the *nodes* of G, and the elements of E are the *edges* of G. A *clustered graph* is a graph $G = (V, E)$ with a partition $C = \{C_1, C_2, \cdots, C_k\}$ on the node set, where each $C_i, i = 1, \cdots, k$ corresponds to a *cluster*, $C_i \cap C_j = \emptyset$ for all $i, j = 1, \cdots, k$, $k \geq 1$, and $V = \sum_{i=1}^{k} C \cup C_{k+1}$, and C_{k+1} denotes potentially empty *unclustered* node set.

An edge is called an *intra-cluster* edge if both its ends belong to the same cluster; an *inter-cluster* edge, otherwise.

Given a clustered graph G, its *quotient graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by merging each cluster into a single node, where:

$$\mathcal{V} = C \cup C_{k+1} \text{ and}$$
$$(v_i, v_j) \in \mathcal{E} \Leftrightarrow i \neq j \wedge (\exists\, v \in C_i, w \in C_j \quad (v, w) \in E)$$

Unclustered nodes are assumed to belong to the distinguished cluster C_{k+1}. We call the nodes of the quotient graph corresponding to clusters, *circle* or *cluster nodes*. Similarly, a node of a clustered graph that belongs to a cluster is called *on-circle* or *in-cluster* node. An on-circle node is a *child* of the circle that it is placed on. On-circle nodes with neighbors outside the cluster are called *out-nodes*. If a node is not on-circle, then it is called *non-on-circle* node. (i.e. it is either a circle or an unclustered node)

Given a cluster graph G, the following terminology will be used to refer to node lists in the rest of the paper:

- *all nodes*: all nodes in G and its quotient graph

$$\mathbf{V(G)} = V(G) \cup \mathcal{V}(\mathcal{G}),$$

- *circle nodes*: all nodes in quotient graph corresponding to clusters

$$\mathbf{V_c(G)} = \{v_i \mid v_i \in \mathcal{V}(\mathcal{G}) \ \wedge \ 1 \leq i \leq k\},$$

- *on-circle nodes*: all clustered nodes in G

$$\mathbf{V_o(G)} = \{u \mid u \in V(G) \ \wedge \ u \in \sum_{i=1}^{k} C_i\},$$

- *non-on-circle nodes*: all but on-circle nodes

$$\overline{\mathbf{V}}_\mathbf{o}\mathbf{(G)} = \mathbf{V(G)} - \mathbf{V_o(G)} = C_{k+1} \cup \ \mathcal{V}(\mathcal{G}).$$

For instance, for the sample clustered graph in Figure 2.1, we have

$$\begin{aligned}
\mathbf{V(G)} &= \{a, b, c, d, e, f, g, 1, 2\} \\
\mathbf{V_c(G)} &= \{1, 2\} \\
\mathbf{V_o(G)} &= \{a, b, e, f, g\} \\
\overline{\mathbf{V}}_\mathbf{o}\mathbf{(G)} &= \{c, d, 1, 2\}
\end{aligned}$$

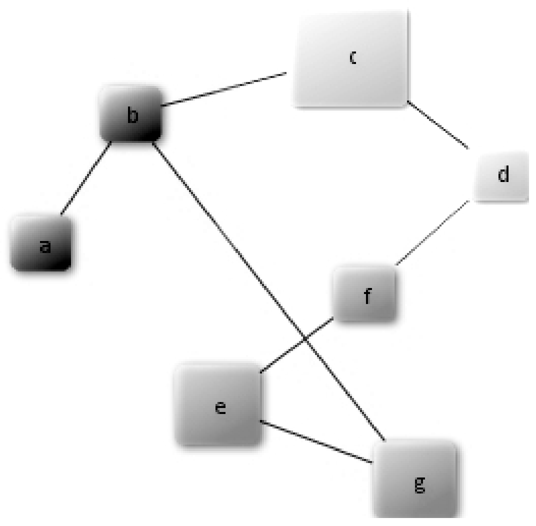

Figure 2.1: A sample clustered graph with 2 clusters $\{a, b\}$ and $\{e, f, g\}$, and unclustered nodes $\{c, d\}$.

Chapter 3

Related Work

As we stated in the previous section, good drawings of graphs involve some sort of prioritization of a set of aesthetic criteria. There is no universal algorithm that will generate beautiful drawings for every kind of application-graph. Therefore there are many algorithms in the literature that try to generate good automatic drawings of family of graphs.

There has been a great deal of work done on general graph layout [6] and it is possible to classify these algorithms under various titles. However, only the two types of layouts described below is related to this study.

3.1 Force directed graph layout

In force directed layout, graph to be laid out is represented as a physical model and a simulation of this model is done with a feasible accuracy. That is graph layout problem is solved via simulating a physical system.

In the basic model, nodes are represented as charged particles that repel each other and edges are represented with springs. The energy level of a node is determined from the forces acting on it. The spring embedder tries to minimize the global energy level by moving the nodes in the direction of the forces. Global energy level, which is the sum of all energy levels of the nodes, is computed after each iteration of the system to determine if the total energy is below a certain amount.

The accuracy and reality of this basic system is a trade off between performance and quality. Generally, for performance reasons only one node is displaced at a time [18].

It is always possible to include additional physical factors in the model to have more realistic hence better resulting systems, like [13]:

- *Magnetic forces:* These are generally used to enforce a flow in to the drawing. For example; in directed graphs it is possible to emphasis the flow if all edges are interpreted as compasses that align themselves according to a magnetic field [28].
- *Gravitational forces:* These are generally used to produce more compact drawings. As spring forces are only effective within components and repulsive forces can make the drawing only bigger. Basic model should be extended to be able to minimize inter component space. Hence gravitational forces are introduced. All nodes are attracted to the mass center of all the other nodes [2].
- *Acceleration:* It is also possible to add mass related factor momentum into the model. This factors adds the previous velocity of a node to the movement being calculated for an iteration. Acceleration is generally added to improve running time performance as well as quality.
- *Temperature:* The basic model with its extensions described until now can settle with a local minima. To overcome this problem it is possible to use controlled amount of randomness. Researchers in optimization theory use a technique from statistical mechanics called *simulated annealing* allowing for changes into states with higher energy. With this addition calculated node movement is disturbed with a relatively minor random vector to avoid being trapped at a local energy minimum. At the beginning the magnitude

of this random vector is bigger, as the simulation matures the system is cooled down meaning magnitude of the force is reduced in order to stabilize the final layout [12].

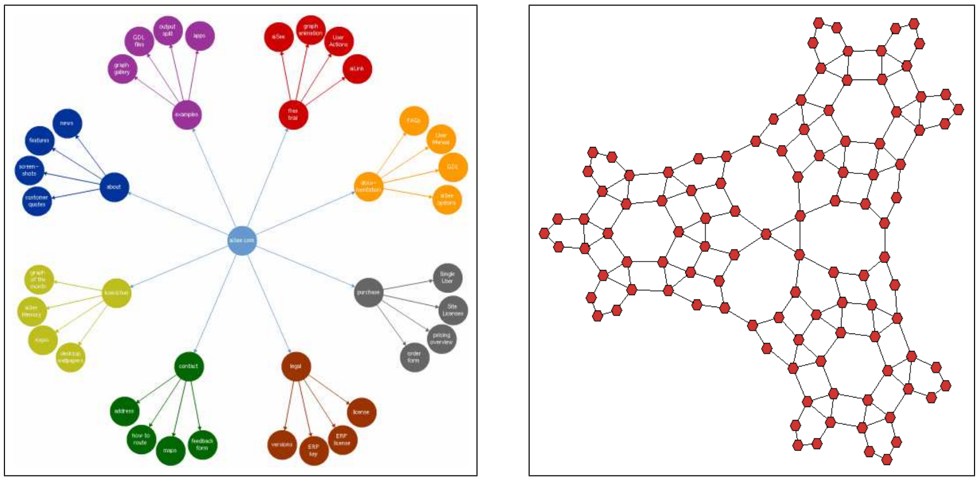

Figure 3.1: Two symmetric drawing samples ([2, 30] respectively).

Force directed layout algorithms are very popular and successful. They reveal structural properties like symmetries, cycles and trees nicely. They have reasonably fast implementations utilizing Barnes-Hut trees and similar data structures. However force directed layout algorithms do not guarantee anything about the final drawing and unit edge length assumption may introduce serious problems for some graphs. An excellent analysis for different approaches to force directed layout is given in [6].

3.2 Circular graph layout

Circular drawing has been an interesting and popular area of research on layout algorithms. Circular graph layout algorithms aim to produce an outer-planar drawing of a given graph where vertices lie on a fixed circle, connected with non intersecting edges lying inside the circle [32]. Since finding an outer-planar embedding of a graph is an NP-hard problem [22], several heuristics developed to

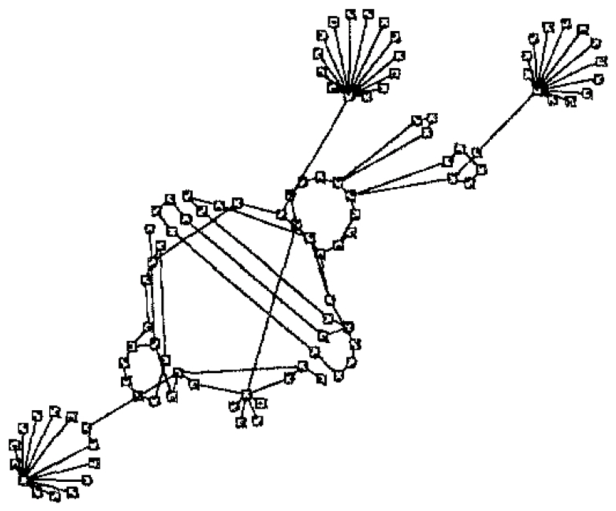

Figure 3.2: A graph drawn by the algorithm in [7].

find approximate solutions to the problem [4, 8, 16, 22]. Most of the algorithms are based on incremental crossing-aware placement of vertices around the circle. Once vertices are initially placed, some post processing applied in order to decrease the total number of edge crossings.

Majority of the circular layout algorithms place the given nodes on a single circle, hence ignoring visualization of any clustered data. Only two of them take clusters into account by using the given cluster information [7, 26]. These algorithms position the nodes on the same circle only if they are assigned to the same cluster therefore resulting in the drawing of one circle per cluster. The study in [26] partially supports circular layout of clustered data when the quotient graph is a tree. The only study that completely addresses layout of arbitrary quotient graph using circles is [7]. However, this algorithm has a major drawback that it places non-tree parts of the quotient graph around a single large "backbone" circle when the cluster graph is cyclic. Such a placement of clusters around a big circle results in additional inter-cluster edge crossings (see Figure 3.2).

Circular drawing of clustered data differs in several ways from the drawing of unclustered data. In the former, the major concern is to position the circles and the nodes inside the circles, so that number of edge crossings is kept minimal.

Since multi-cluster layout introduces two more crossing types, inter-cluster - intra-cluster and inter-cluster - inter-cluster edges, nodes around the circle and also the circle itself should be properly positioned in order to reduce these types of crossings as well as keeping the circular layout inside the clusters optimal (see Figure 3.3).

Multi-cluster layout divides the problem of drawing a large single piece of data into smaller problems, thus reducing the complexity of visualization requirements. However, on the other hand, clustered layout introduces the positioning problem of clusters and nodes as described above. Recognizing the trade off, this study addresses an algorithm “aware” of inter-cluster edges, in order to aesthetically and efficiently display clustered graph information using a circle for each cluster.

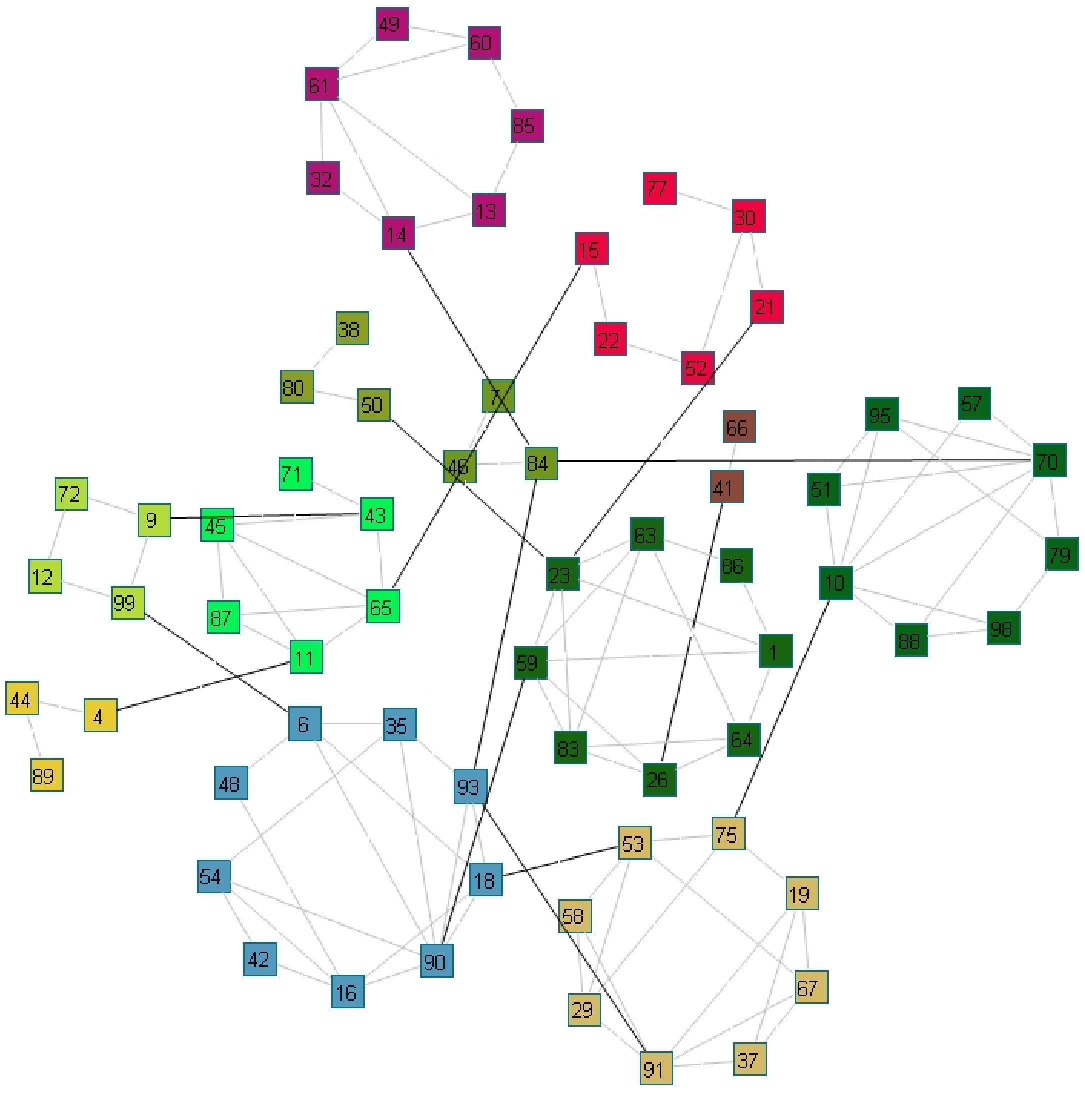

Figure 3.3: A sample drawing where individual clusters are nicely laid out but the overall layout is bad (i.e. there are many inter-cluster edge crossings)

Chapter 4

Layout Algorithm

4.1 Underlying physical model

A basic force-directed layout algorithm with certain extensions to satisfy the clustering conventions in circular drawings has been chosen. Basic idea of the layout algorithm is to simulate a physical system in which nodes are assumed to be physical objects with certain “electrical charge”, connected via “springs” of a pre-specified desired length. Objects pull or repel each other depending on current lengths of any connected springs. In addition, relatively minor repulsion forces act on any pair of objects that are “too close” to each other to avoid node-to-node overlaps. Furthermore, we assume “gravitational forces” to keep graph components together.

In order to handle varying node sizes (especially larger cluster nodes) and avoid overlaps with neighboring nodes, calculation of distances are based on the borders of nodes, as opposed to their centers [15]. Thus the optimal layout is regarded as the state of this system, in which total energy is minimal. The use of extra constraints is implemented by introducing extra properties to the physical model used by the spring embedder, trying to obey the basic (mainly Newtonian) laws of physics:

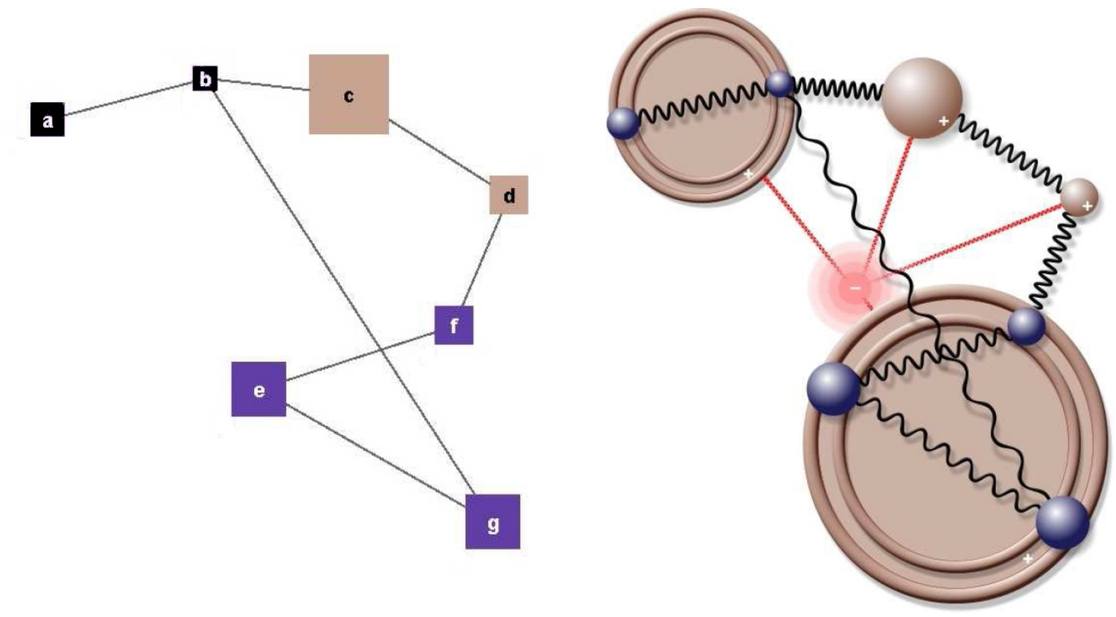

Figure 4.1: A sample clustered graph with 2 clusters $\{a, b\}$ and $\{e, f, g\}$, and unclustered nodes $\{c, d\}$ (left) and the corresponding physical model used by our algorithm (right).

- Each cluster/circle is represented by a "meta-node" of circular shape, on which sits a round shaped track on its periphery. The physical entities for each member node of a cluster is assumed to be either *fixed* (pinned down to its owner circle) or *flexible* to move around (via swapping with their neighbors) the track on which they sit as needed by the different steps of the algorithm. In any case, the on-circle nodes move with their owner circle nodes. This fulfills the requirement of member nodes being on the periphery of the owner circle.

- A center of gravity in the middle of the bounding rectangle of the current drawing is assumed. All unclustered nodes and cluster nodes (all nodes except member nodes of a cluster) are attracted towards this center. This should keep disconnected parts of a graph together.

Figure 4.1 illustrates the basics of our physical model with an example.

4.2 Algorithm

We assume that the graph to be laid out is a clustered graph $G = (V, E)$ with clusters $C = \{C_1, C_2, \cdots, C_k\}$, unclustered nodes C_{k+1}, and quotient graph $\mathcal{G}$, all using adjacency list representations. Layout specific data and functionality are kept in these structures as well. In addition, we assume special mechanisms for efficient iteration over necessary graph objects exist.

Our algorithm named CiSE (Circular Spring Embedder) is composed of four major steps preceded by an initialization phase:

Initialization: This is where the necessary structures for layout along with quotient graph of the graph to be laid out are constructed.

- **Step 1:** In this step each cluster is laid out independently using a circular layout algorithm of your choice (e.g. [16]).

- **Step 2:** Here we determine the "skeleton" of the layout by laying out the quotient graph. The specific algorithm to use for this step depends on the structure of the quotient graph. If it is always a tree, a radial layout is ideal. For the general case, best choice seems to be a regular spring embedder. Note however that the dimensions of nodes of this graph will be non-uniform, requiring extra attention.

- **Step 3:** In this step, our aim is to reposition/rotate circles according to the location of their out-nodes and inter-cluster edges incident on these nodes. However, nodes on the circles are not allowed to move individually. They are assumed to be "pinned down" to their owner circles. After this step, a draft layout of the whole graph is obtained.

- **Step 4:** The difference between this step and the previous one is that, we allow nodes on circles to move with respect to their parent circle (as

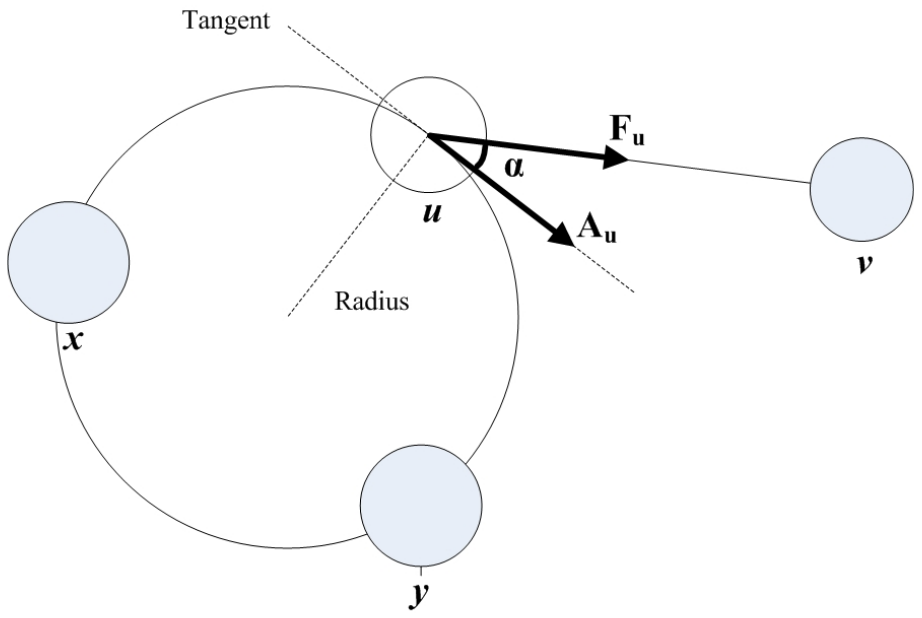

Figure 4.2: Extraction of rotational force A_u from the total force F_u acting on a circular node.

well as moving with them) by swapping them with their neighbors. Two neighboring on-circle nodes are swapped only if the operation does not increase the edge crossing count.

Steps 3 and 4 make up the core of our algorithm. In the following, we describe how we calculate and make use of different kinds of forces (Figure 4.2, 4.3) as part of a modified spring embedder implemented with these steps.

The formula for calculating the spring force for edge $e = (u, v)$ is

$$F_s = \frac{(\lambda - ||p_u - p_v||)^2}{\eta} \; \vec{p_u p_v},$$

where λ is the ideal edge length, η is the elasticity constant of the edge, and p_u and p_v are positions of nodes u and v, respectively. Ideal edge length of an inter-cluster edge should be chosen to be a reasonable factor larger than an intra-cluster one to better separate the clusters. Non-uniform node dimensions require force calculations to be based on clipping points rather than node centers. The

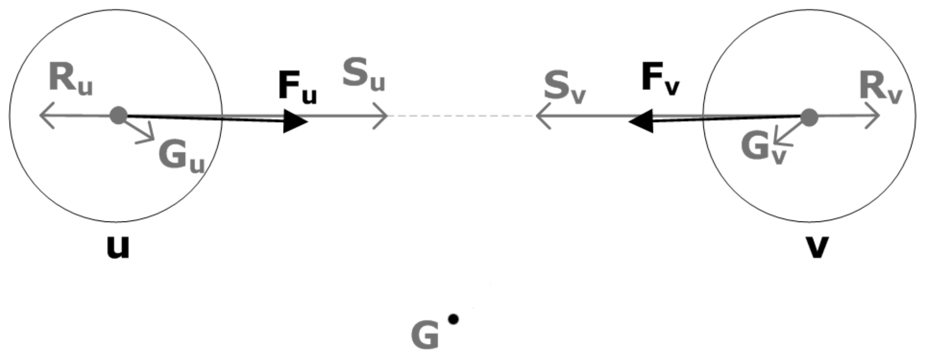

Figure 4.3: Calculation of total force F using spring, repulsion and gravity forces (S, R and G respectively) acting on a node.

following method is used for calculating spring forces acting on each edge's ends:

```
algorithm CalcSpringForces(Graph $G$, int $step$)
1) for each $e = (u, v) \in E(G)$ do
2)   $idealLength := \lambda$
3)   if $step = 4$ or $e$ is an inter-cluster edge then
4)     $c_u := u.boundRect \cap$
         LineSegment($u.center$, $v.center$)
5)     $c_v := v.boundRect \cap$
         LineSegment($u.center$, $v.center$)
6)     $S := (idealLength - ||c_u - c_v||)^2/\eta \cdot \vec{c_u c_v}$
7)     $S_u$ += $S$
8)     $S_v$ −= $S$
```

Notice that spring forces for intra-cluster edges are ignored during step 3 as we assume the nodes to be fixed (i.e. those forces would have canceled each other if they were to be transferred to their owner circles).

The overall time complexity of this method is $\Theta(|E^M|)$ as all steps inside the for-loop can be processed in $\Theta(1)$ steps.

Node-to-node repulsion forces are calculated using the formula

$$F_r = \frac{\alpha}{||p_u - p_v||^2}\ \vec{p_u p_v},$$

where α is the repulsion constant. Similar to spring forces, repulsion forces require us to make clipping point calculations for nodes of non-uniform size based on the line passing through nodes' centers:

```
algorithm CalcRepulsionForces(Graph $G$,
    int $step$)
1) for each pair of nodes $u, v \in V(G)$ do
2)   if ($u, v \in \mathbf{V_o(G)}$ and $u.owner = v.owner$ then
3)     $c_u := u.boundRect \cap$
         LineSegment($u.center$, $v.center$)
4)     $c_v := v.boundRect \cap$
         LineSegment($u.center$, $v.center$)
5)     if $||c_u - c_v|| <$ REPULSION_RANGE then
6)       $R := \alpha / ||c_u - c_v||^2$
7)       $R_u$ += $R$
8)       $R_v$ −= $R$
```

Steps 6-10 are handled in $\Theta(1)$ steps, which are executed a total of maximum $O(|V^M|^2)$ times, making the overall complexity of the method $O(|V^M|^2)$. However, since a node pair affect each other only when they are below a certain geometric distance and within the same graph, the average complexity is expected to be asymptotically lower than this.

Gravitation forces have fixed magnitude and they are always towards the center of the bounding rectangle of the owner graph:

```
algorithm CalcGravitationForces(Graph $G$)
1) for each $u \in \overline{\mathbf{V}}_\mathbf{o}\mathbf{(G)}$ do
2)   $center := G.boundRect$
3)   calculate gravitation force $G$ towards $center$
4)   $G_u$ += $G$
```

The overall time complexity of this method is $\Theta(|V^M|)$ as all steps inside the for-loop can be processed in $\Theta(1)$ time.

In each iteration, once all kinds of forces are calculated, we add them up to determine the total force on each node as follows:

```
algorithm CalcTotalForces(Graph $G$, int $step$)
1) for each $u \in \mathbf{V(G)}$ do
2)    $F_u := S_u + R_u + G_u$
3) for each $u \in \mathbf{V_o(G)}$ do
4)    if in swap preparation phase then
5)       $D_u += \text{Horizontal}(F_u)$
6)       $o := u.owner$
7)       $F_o += (F_u)$
8)       $A_o += \text{Horizontal}(F_u)$
9)       $F_u := 0$
```

The main steps 3 and 4 using earlier ones make up the core of our algorithm:

```
algorithm PerformStep3-4(Graph $G$, int $step$)
1) Initialize($G$)
2) $iter := maxIterCount[step]$
3) $error := 0$
4) while ($iter > 0$ and
         $error > errorThreshold[step]$) do
5)    CalcSpringForces($G$, $step$)
6)    CalcRepulsionForces($G$, $step$)
7)    CalcGravitationForces($G$)
8)    CalcTotalForces($G$, $step$)
9)    MoveNodes($G$, $step$)
10)   $iter := iter - 1$
```

```
 algorithm MoveNodes(Graph $G$, int $step$)
1)  for each $u \in \overline{\mathbf{V}}_{\mathbf{o}}(\mathbf{G})$ do
2)    if $u \in \mathbf{V}_{\mathbf{c}}(\mathbf{G})$ then
3)      move it using $F_u$ / # nodes in $u$
4)    else
5)      move it using $F_u$
6)    if $u \in \mathbf{V}_{\mathbf{c}}(\mathbf{G})$ then
7)      rotate it using $A_u$ / # nodes in $u$
8)  if $step = 4$ and in swap phase then
9)    for each $u \in \mathbf{V}_{\mathbf{o}}(\mathbf{G})$ do
10)     determine safe and unsafe node pairs to swap
11)     perform swap
12)     for each $u \in \mathbf{V}_{\mathbf{o}}(\mathbf{G})$ do
13)       $D_u := 0$
```

Once all forces have been calculated during an iteration, we move each node with respect to the total force acting upon it. Movement occurs in three ways: translation, rotation and swap. Prior to all types of movements, a factor of the current temperature is applied on the total force as part of the global cooling schema.

Translation is for only non-on-circle nodes. Circle nodes translate together with its on-circle nodes, hence total forces acting on a circle node should be divided by the children count. Additionally, circle nodes are rotated in the average rotation amount contributed by their children.

Swaps are performed during step 4 only inside swap phases. Swap operation is divided into two types: safe and unsafe. Participants of safe swaps are on-circle nodes which are not classified as out nodes. Such pairs are safe to swap because no inter-cluster edge crossings would be introduced after a swap. However, unsafe swaps might cause inter-cluster edge crossings, since an unsafe pair consists of at least one out-node. If no crossings would be introduced, then an unsafe swap is also allowed.

A quick analysis of the algorithm reveals that the running time of layout of a compound graph is $O(k \cdot |V^M|^2)$ where k is the number of iterations required to reach an energy minimal state.

Chapter 5

Implementation

The proposed layout algorithm has been developed and tested within Chisio 1.1, an open source software released in 2009 by Information Visualization Research Group at Bilkent University [3]. The development environment is Sun's Java SDK 1.5 and Microsoft Windows XP operating system on an ordinary personal computer (Pentium D 2.8 GHz CPU and 3 GB memory).

Chisio is a general purpose graph editing and layout tool supporting standard graph editing facilities like zoom, scroll, add/remove graph objects, move and resize. With the help of the extensible framework provided by Chisio for layout algorithm developers, we have easily conducted our study on this platform. Our algorithm has been added to supported layouts list of Chisio under the name "CiSE" (Circular Spring Embedder).

In order to adapt our algorithm to Chisio, we needed to extend the basis model provided by Chisio layout package (Figure 5.1). The basis model is a simple structure where algorithms keep geometry and topology information for the layout elements during the layout. It is called "L-level" and consists of simple graph objects like nodes and edges. The CiSE model extends this basis model by defining layout-specific objects like circles, clustered and un-clustered nodes, node pairs and edges.

Since L-level objects are special to layout operations, Chisio does not use them for actual rendering operation. Instead, Chisio has its own representation of graph objects, Chisio graph model (Figure 5.2), which needs to be synchronized with L-level objects before and after layout. More clearly, geometry and topology information is transferred from Chisio graph model to L-level objects prior to the layout. Similarly, after layout (and during layout as well if animation is enabled), the information is written back to the Chisio model, hence allowing Chisio to update the visual representation of the graph properly.

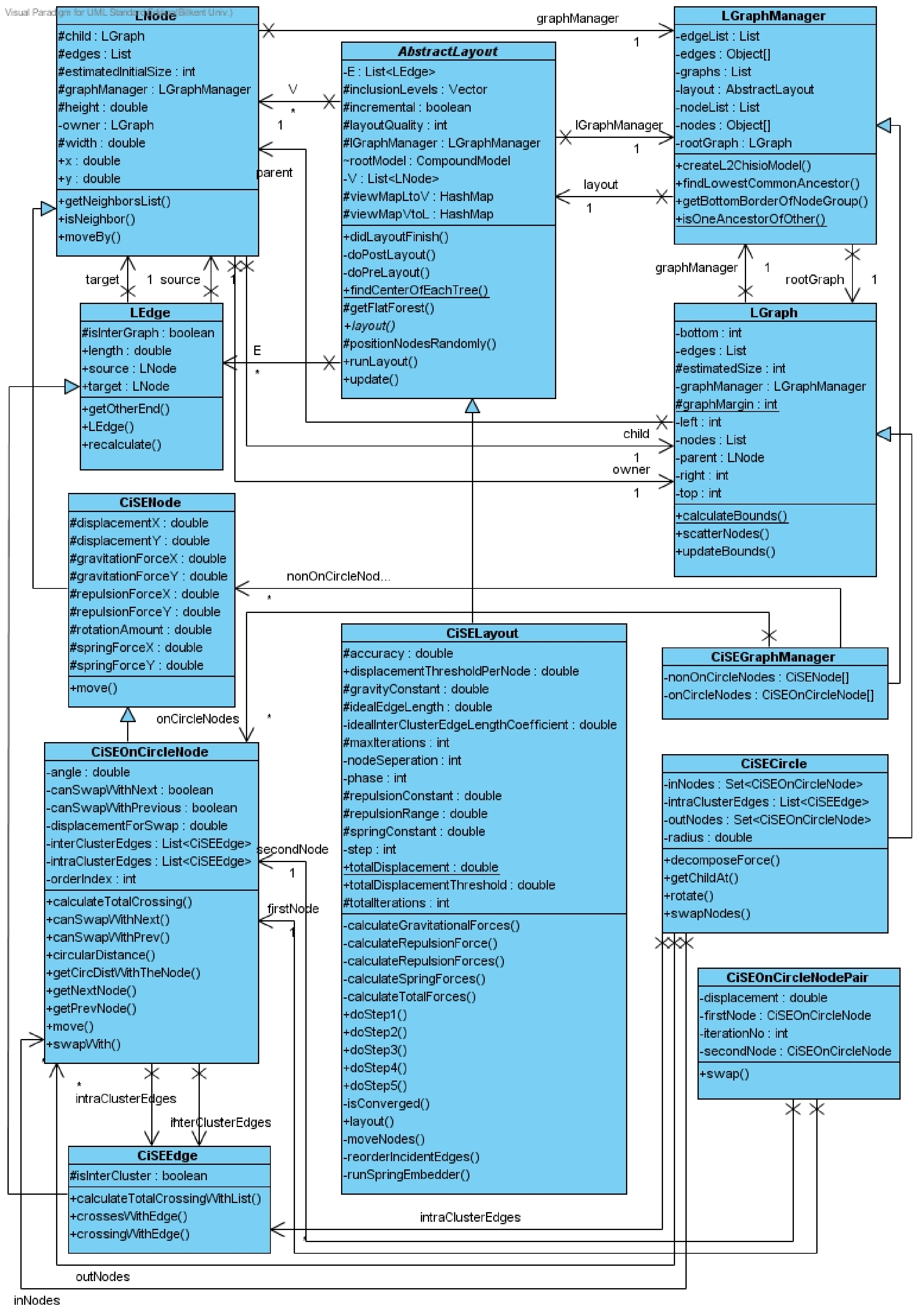

Figure 5.1: Class diagram for Chisio L-level package and CiSE extension.

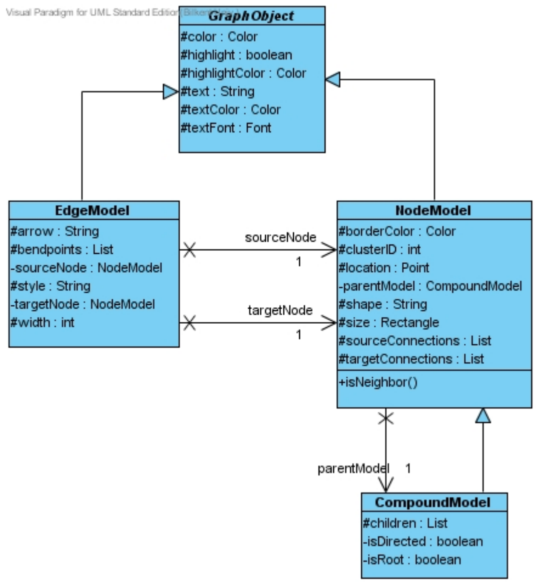

Figure 5.2: Class diagram for Chisio graph model

Chapter 6

Experimental Results

6.1 Running time performance

We have performed experiments on execution time of our layout algorithm on randomly generated graphs with one of several parameters changing for each set. For each test, a random graph has been generated with the provided parameters:

- n: total number of nodes,
- m/n: proportion of edges to number of nodes,
- m_{ic}/m: proportion of inter-cluster edges to number of all edges,
- d_{max}: maximum cluster size,
- d_{min}: minimum cluster size,

In order to create a random clustered graph, first we generate the nodes and distribute them to clusters while respecting maximum cluster size and number of on-circle nodes divided by number of all nodes ratio. Then we create edges between the nodes in different clusters with a total count of m_{ic} times and leaving rest for inter-cluster edges. Each test is executed 10 times and the average is taken. Figure 6.1 shows an example of a randomly generated clustered graph. ‘

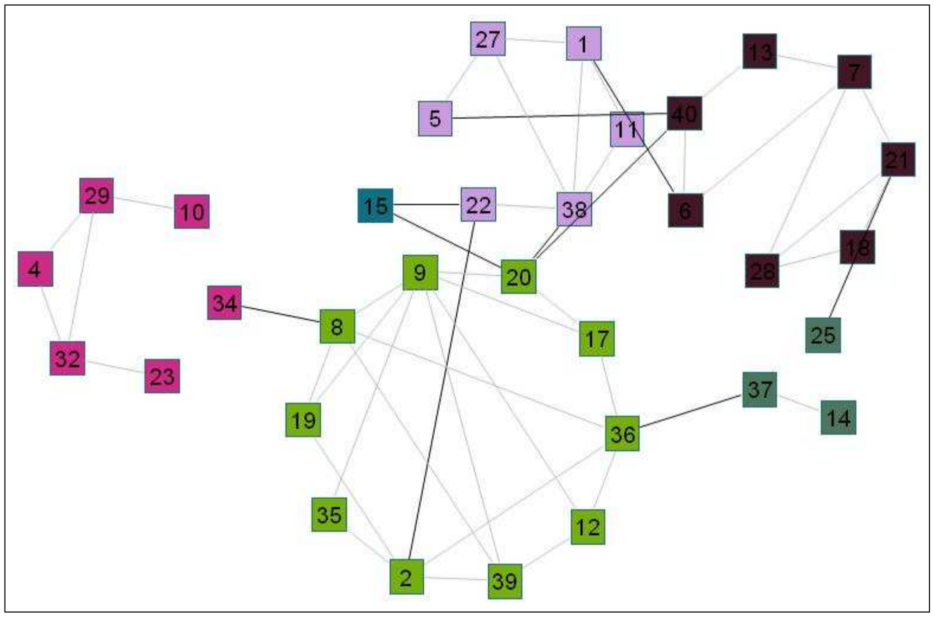

Figure 6.1: A randomly generated graph laid out by our algorithm. ($n = 40$, $m/n = 1.2$, $m_{ic}/m = 0.20$, $d_{max} = 10$ and $d_{min} = 2$)

From the theoretical analysis given earlier, a quadratic behavior of execution time is expected. The experiments validate this argument (Figures 6.2).

We have also performed a test set to see how the proportion of inter-cluster edges to all edges affects the execution time (Figure 6.3). Running time seems to be affected when the ratio is very low. This is due to the fact that, layout of the quotient graph converges very quickly when there are very few number of inter-cluster edges. However, early convergence does not exist when the ratio gets bigger, therefore resulting in varying timings independent from the ratio.

Another experiment we have conducted was on the effect of cluster sizes on execution time (Figure 6.4). When there are bigger clusters, quotient graph tends to be smaller which results in faster layout times for the cluster graph. On the other hand, layout of individual circles takes longer due to increasing size of clusters. However, since the complexity of the algorithm [16] used for inner layout of circles is linear, the time loss for this operation is compensated by the cluster graph layout, which is quadratic. As a result, an increase in cluster sizes results in a decrease in the overall running time.

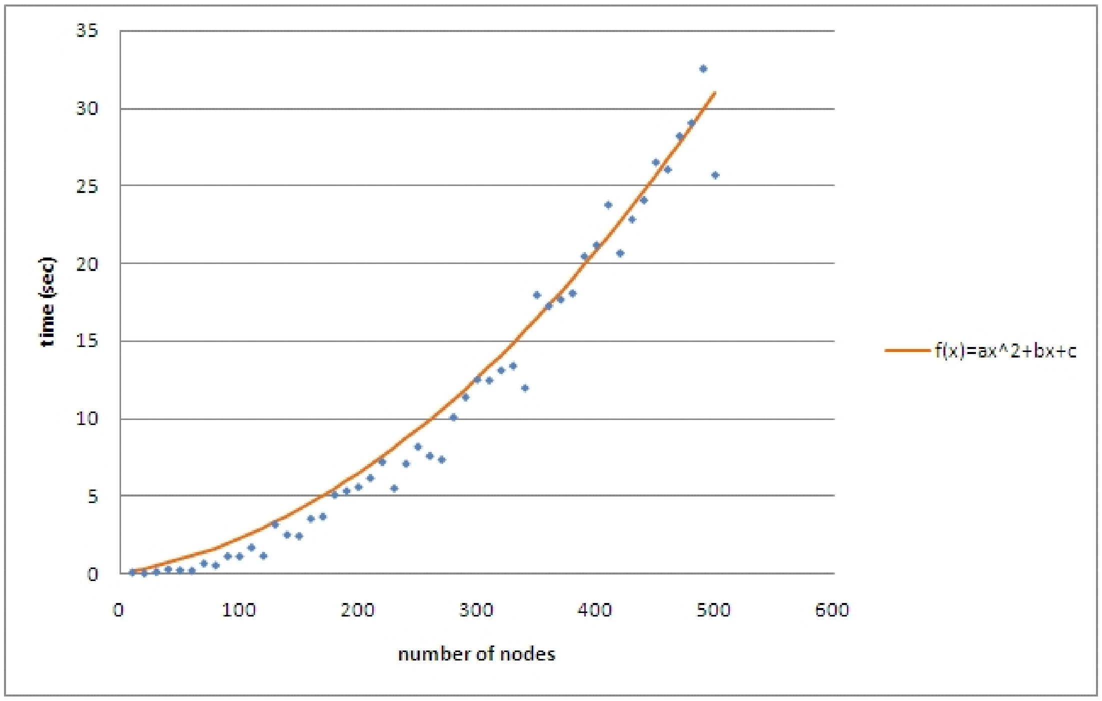

Figure 6.2: Number of nodes (n) vs. execution time of our algorithm. ($m/n = 1.5$, $m_{ic}/m = 0.10$, $d_{max} = 15$ and $d_{min} = 2$)

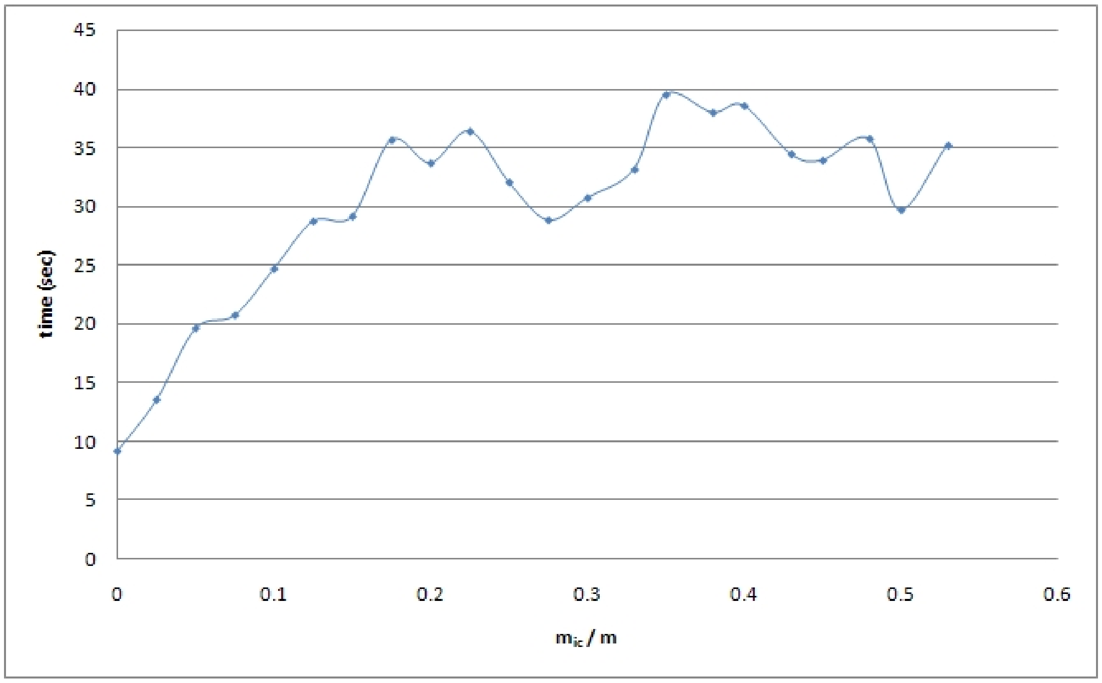

Figure 6.3: Inter-cluster edge ratio (m_{ic}/m) vs. execution time of our algorithm. ($n = 500$, $m/n = 1.5$, $d_{max} = 15$ and $d_{min} = 2$)

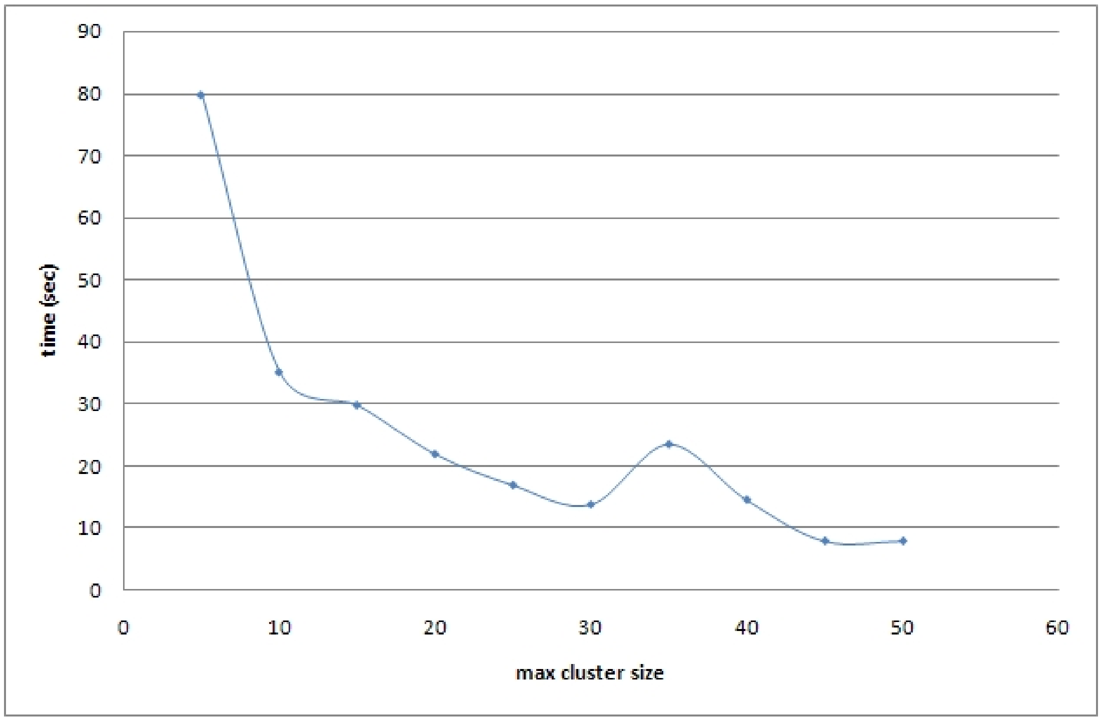

Figure 6.4: Maximum cluster size (d_{max}) vs. execution time of our algorithm. ($n = 500$, $m/n = 1.5$, $d_{min} = 2$ and $m_{ic}/m = 0.10$)

The last observation we made was on the effect of the difference between maximum and minimum cluster sizes on execution time (Figure 6.5). The purpose of this test is to observe how uniformity of cluster sizes affects the results. As it can be seen from the resulting chart, non-uniformity has a positive effect on the execution time. Differences between cluster sizes help the nodes move more freely during quotient graph layout. In other words, smaller clusters can more easily move around bigger circles, hence approaching to convergence faster by relaxing more edges on each iteration.

6.2 Quality

In our experiments, quality of the layout algorithm is also inspected according to the aesthetic criteria defined in Section 1.1. In general, the results produced by the algorithm are satisfactory in terms of node-node overlapping and drawing

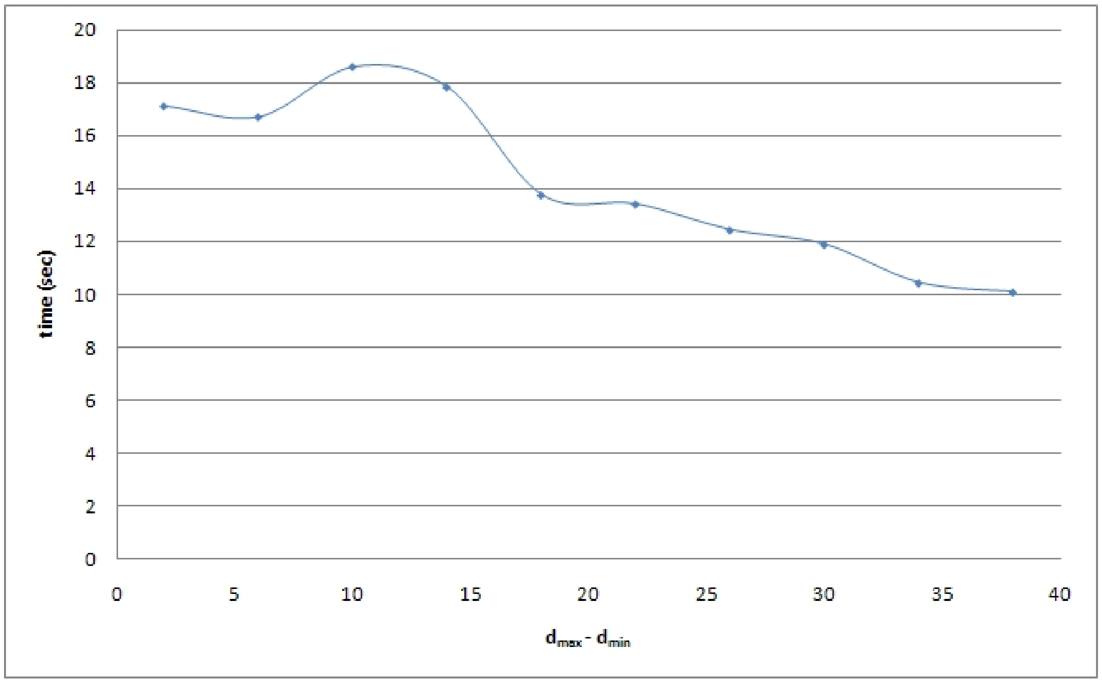

Figure 6.5: Maximum-minimum cluster size discrepancy (d_{max} - d_{min}) vs. execution time of our algorithm. ($n = 500$, $m/n = 1.5$, $d_{max} = 44$ and $m_{ic}/m = 0.10$)

area. Nodes almost never overlap due to repulsion forces. On the other hand, they stay sufficiently sufficiently close to each order, resulting in a tight and compact drawing. Drawing area is uniformly occupied by clusters preserving the symmetry of the visualization.

On the other hand, it is difficult to state edge lengths to be uniform. This is due to two main reasons:

- Intra-cluster edges usually have varying lengths because of the circular positioning of nodes. Since minimizing edge crossings is highest priority and nodes are placed at a fixed distance from each other, the edge lengths will inevitably vary according to the order of nodes around the circle.
- Inter-cluster edges might sometimes be arbitrarily long because of the unachievable swaps. Swaps are requested as a result of opposite spring forces caused by long incident edges acting on two on-circle nodes. Since highest priority is minimizing edge crossings, we never swap those two nodes if the

swap would introduce new edge crossings. However, if such swaps were performed, spring forces acting on these nodes would relax by the help of shortened inter-cluster edge lengths. In other words, such a precaution on swap operations prevents new edge crossings at the cost of longer edge lengths than the desired.

For a better evaluation of the layout quality, we compared our algorithm with the circular clustered layout algorithm in the Graph Layout Toolkit (GLT)[7]. Since the details of this algorithm implemented as part of a commercial tool are not available, we have compared our algorithm with theirs for one of the two similar graphs provided in their papers. (Figure 6.6). As you can see, the major drawback of their algorithm is that when the cluster graph is cyclic, non-tree parts of the cluster end up on a single large backbone circle in the middle, introducing many inter-cluster edge crossings. Also notice that inter-cluster edges are very long compared to their intra-cluster counterparts.

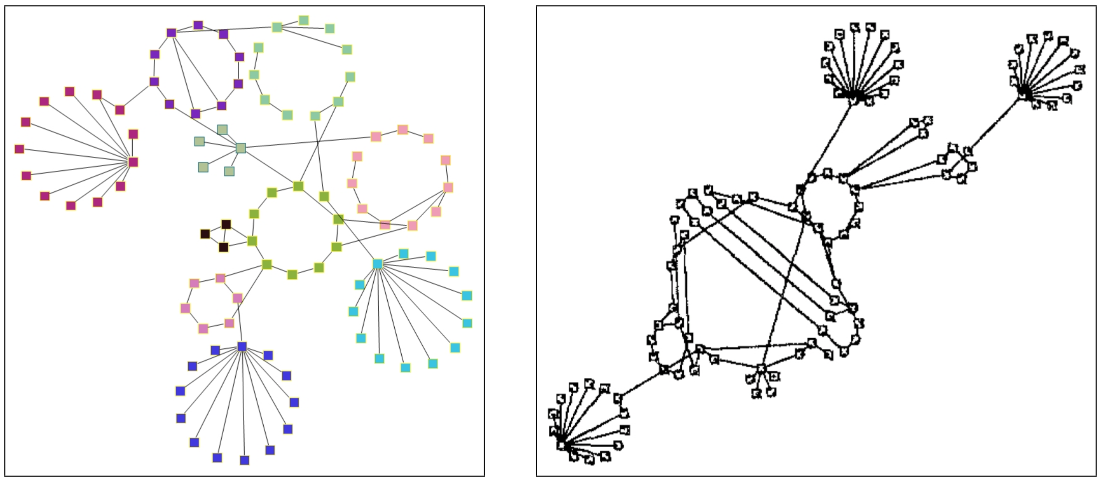

Figure 6.6: Two layouts of the same graph: CiSE on the left and circular layout of GLT on the right [13].

The algorithm in [26], on the other hand, is able to handle clustered layout only when the quotient graph is a tree, therefore we did not include this algorithm in the comparison.

Chapter 7

Conclusion

In this study, we have presented a new algorithm for layout of clustered graphs in a circular fashion. To our knowledge, this is the first layout algorithm that uses a force directed scheme in order to create circular drawings of clustered graphs. It addresses layout of the quotient graph and can handle non-uniform node dimensions. The main novelty of our work is the use of a modified spring embedder that treats clusters and edges between them as part of a physical system while keeping circular layout for each cluster at an optimal quality level. The algorithm produces successful results in terms of both time complexity and aesthetics criteria.

On the other hand, there is still room for improvement. In order to reach convergence earlier, oscillations caused by repetitive swaps between same nodes in consecutive iterations should be carefully studied. Although there are several checks for preventing them, more advanced handling of the cases where such oscillations might occur should be performed. Moreover, any improvements on clipping point/rectangle calculations would end up in a considerable decrease in overall running time because spring and repulsion force computations heavily depend on clipping point/rectangle detection. Another improvement for the current system would be to support multi-level or recursively structured clustered graphs.

Appendix A

Sample Drawings Produced by CiSE

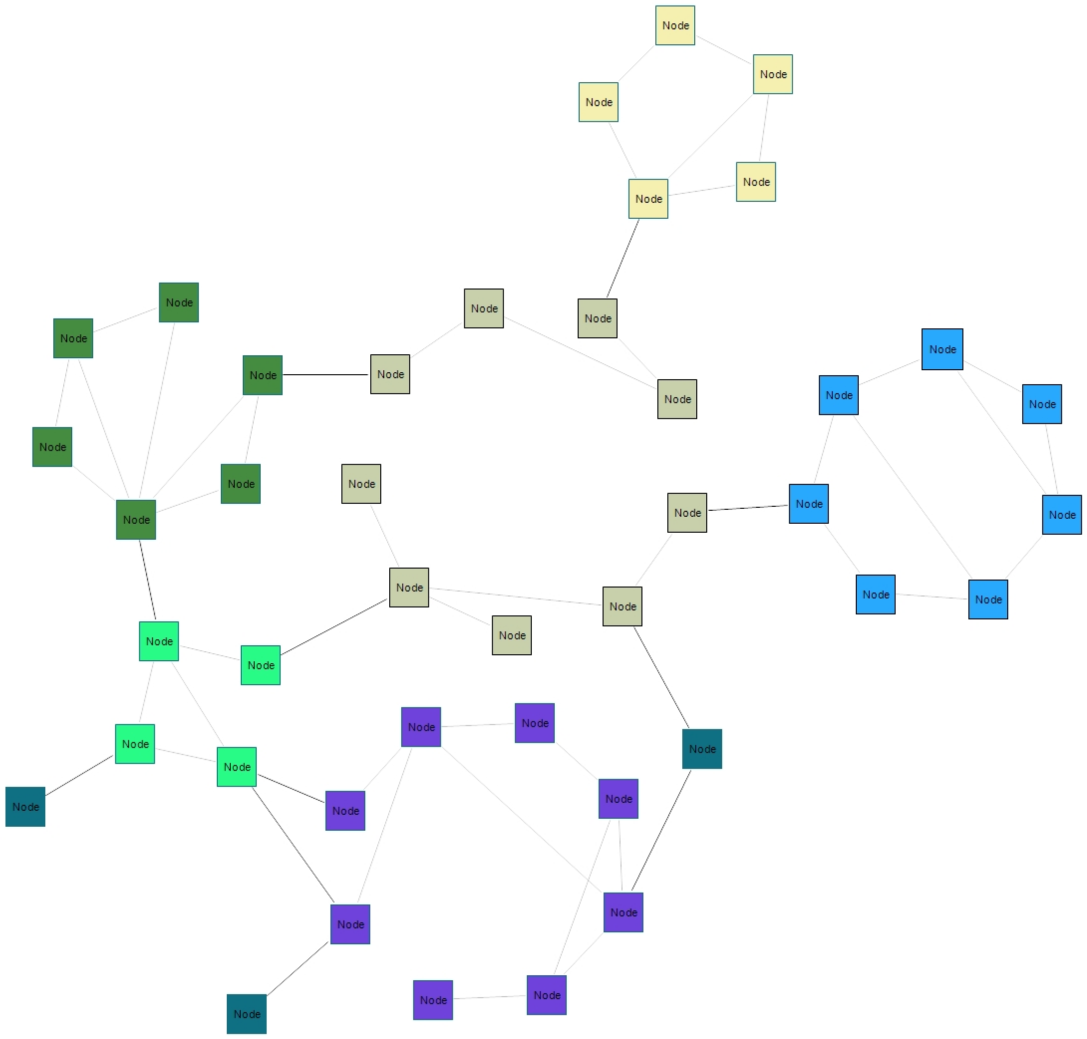

Figure A.1: A drawing created by CiSE

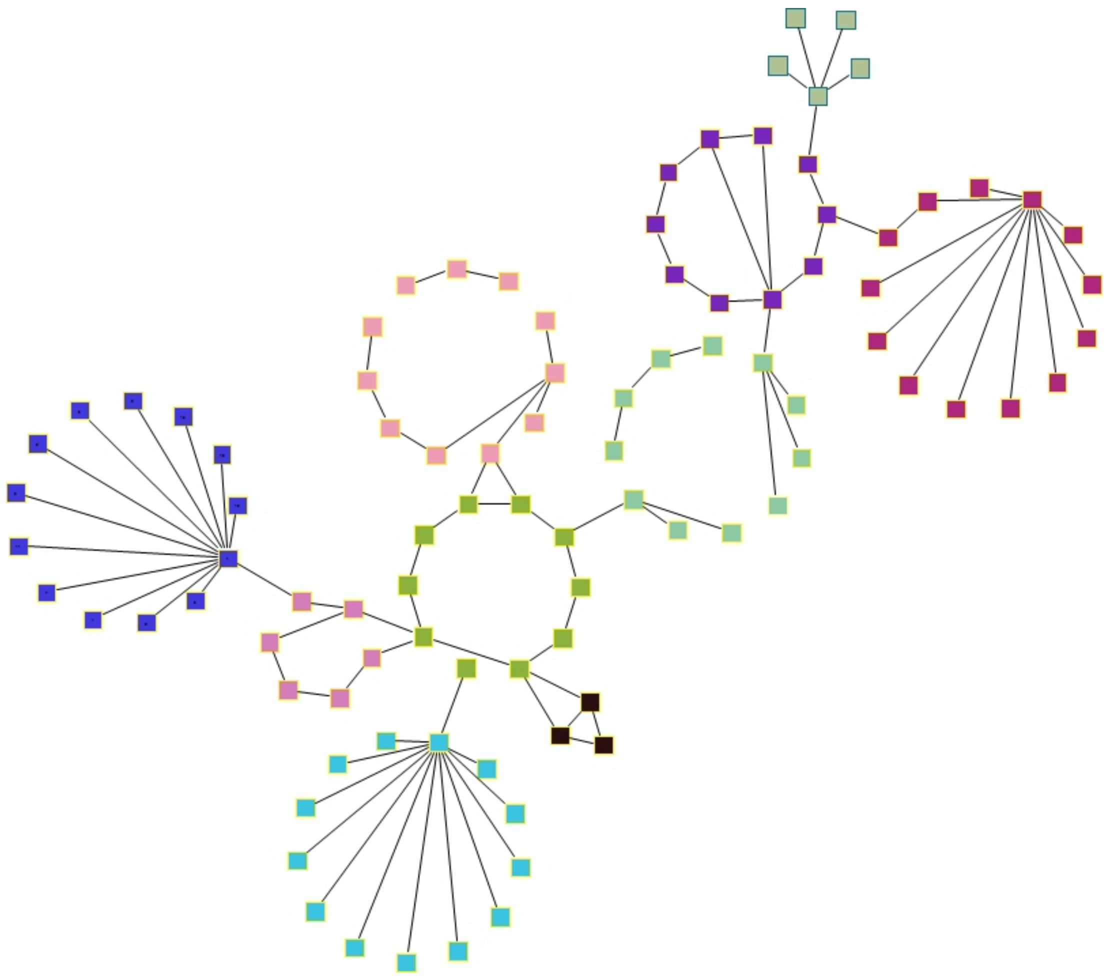

Figure A.2: A drawing created by CiSE

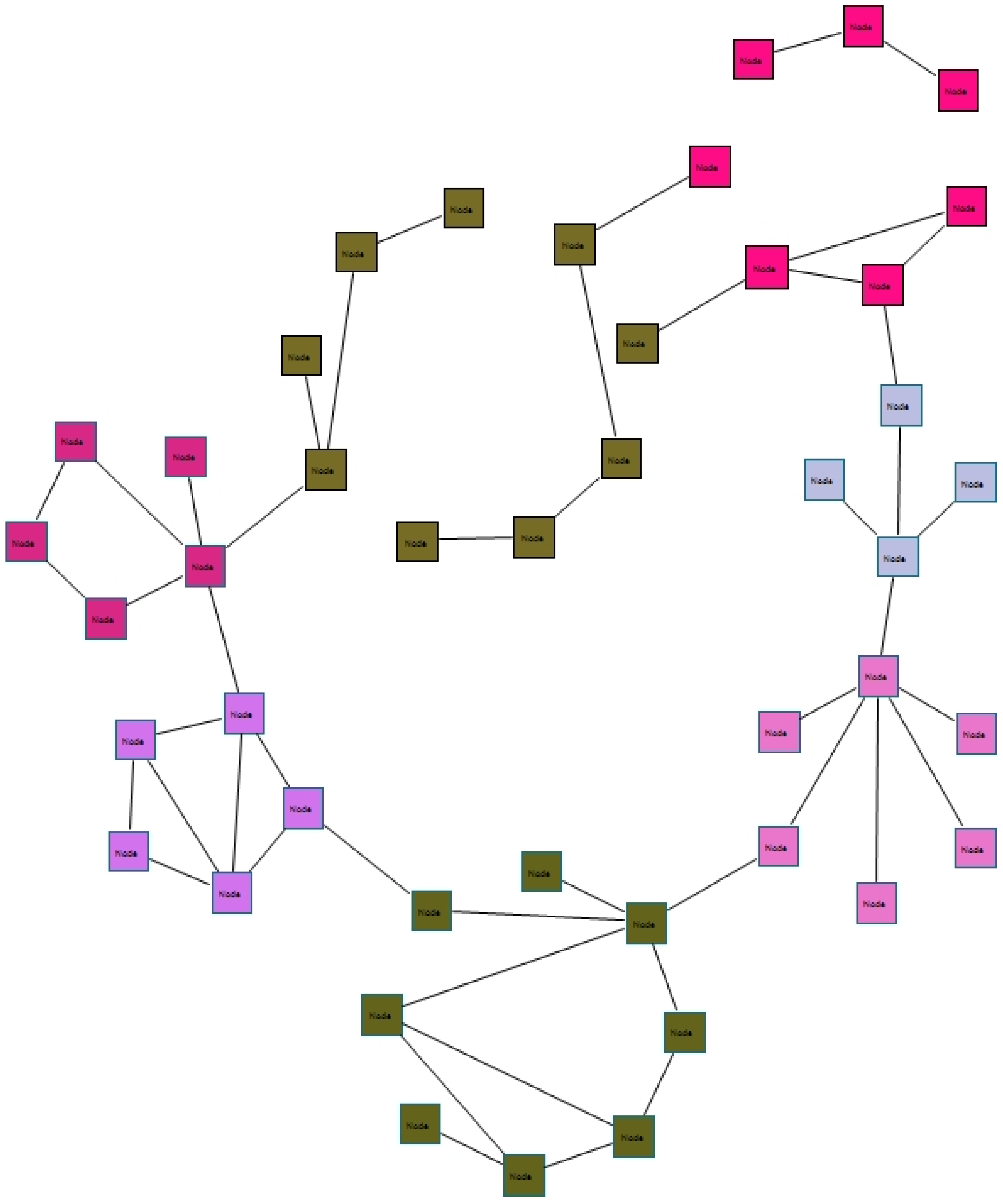

Figure A.3: A drawing created by CiSE

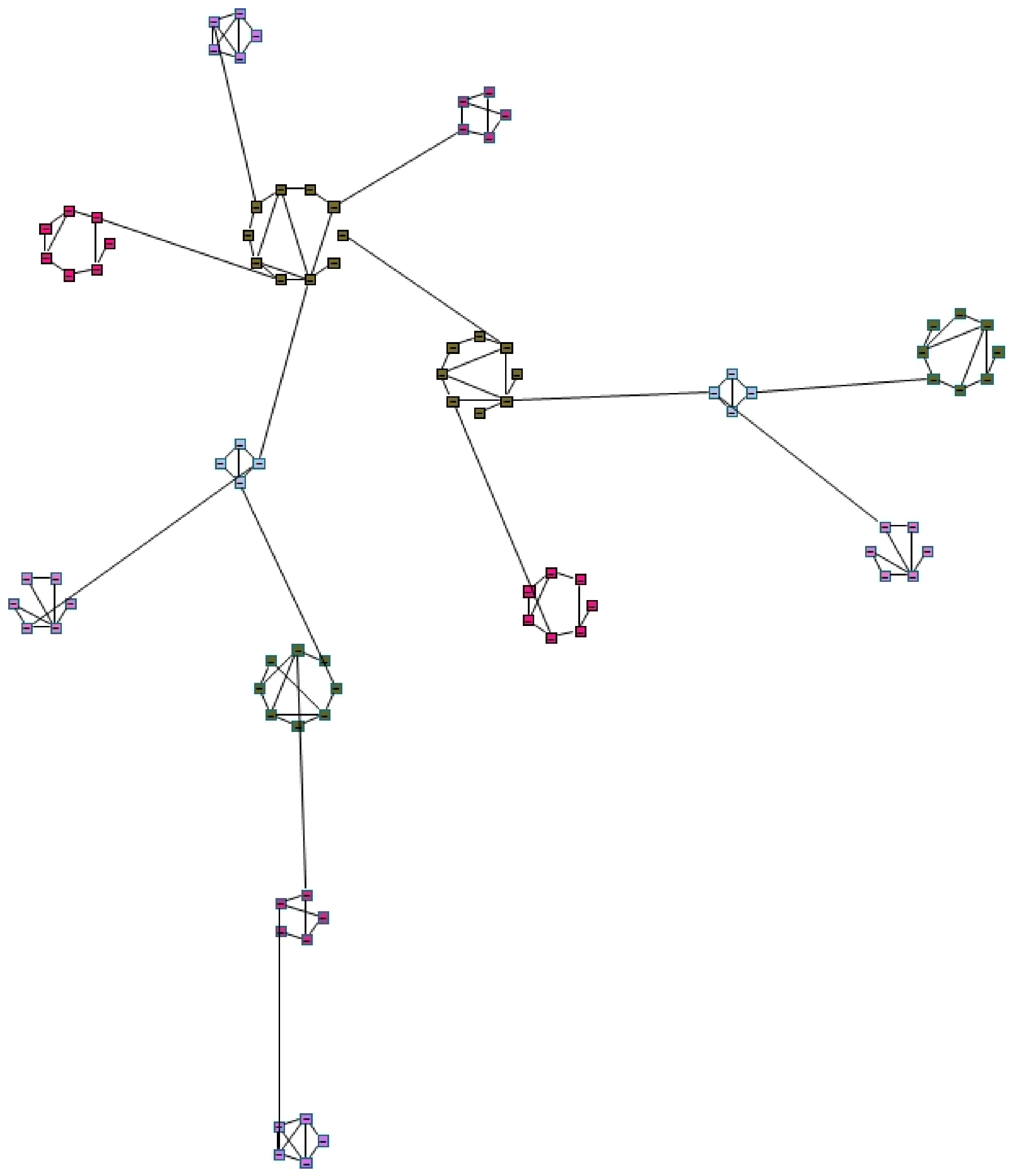

Figure A.4: A drawing created by CiSE

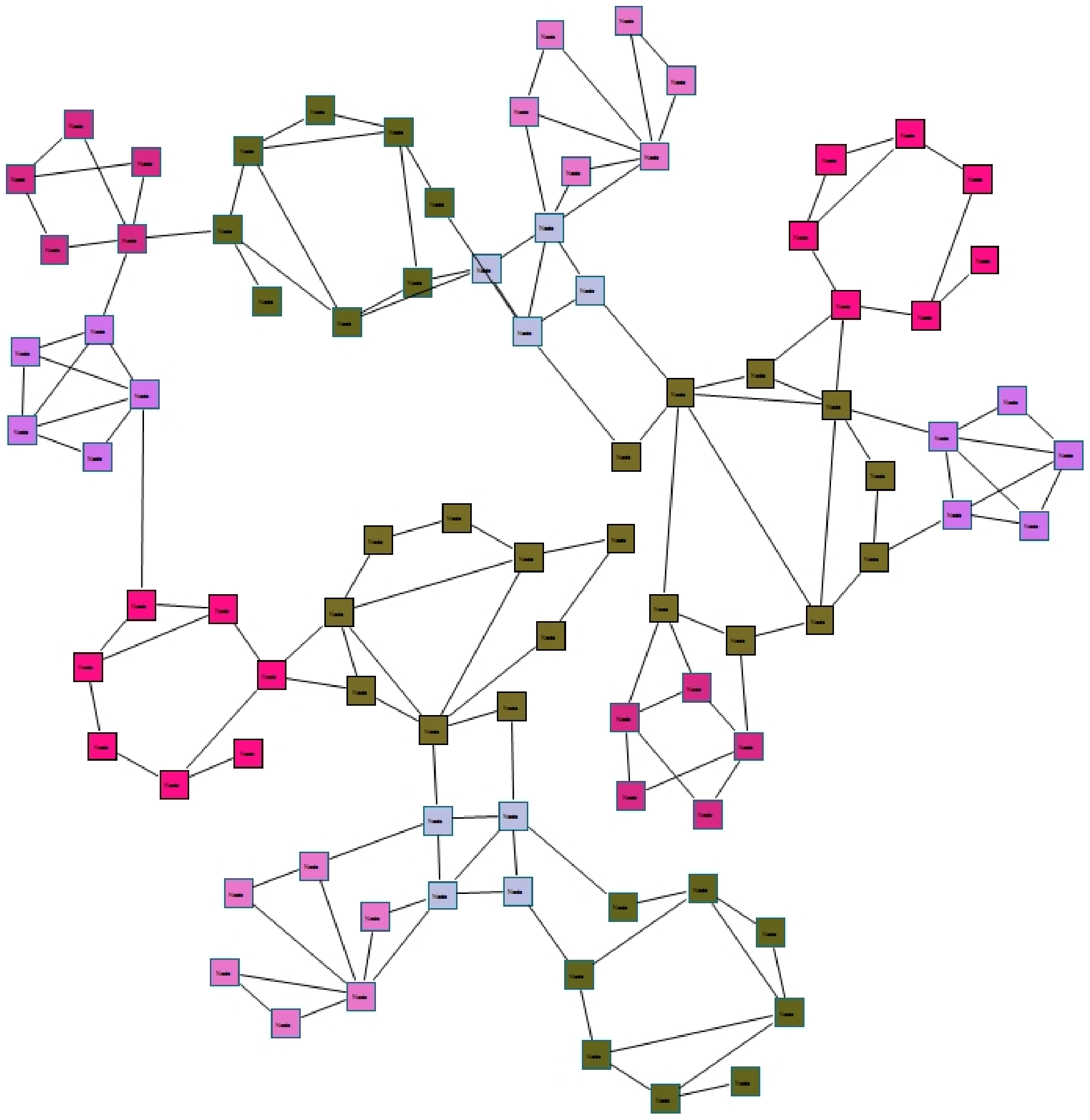

Figure A.5: A drawing created by CiSE

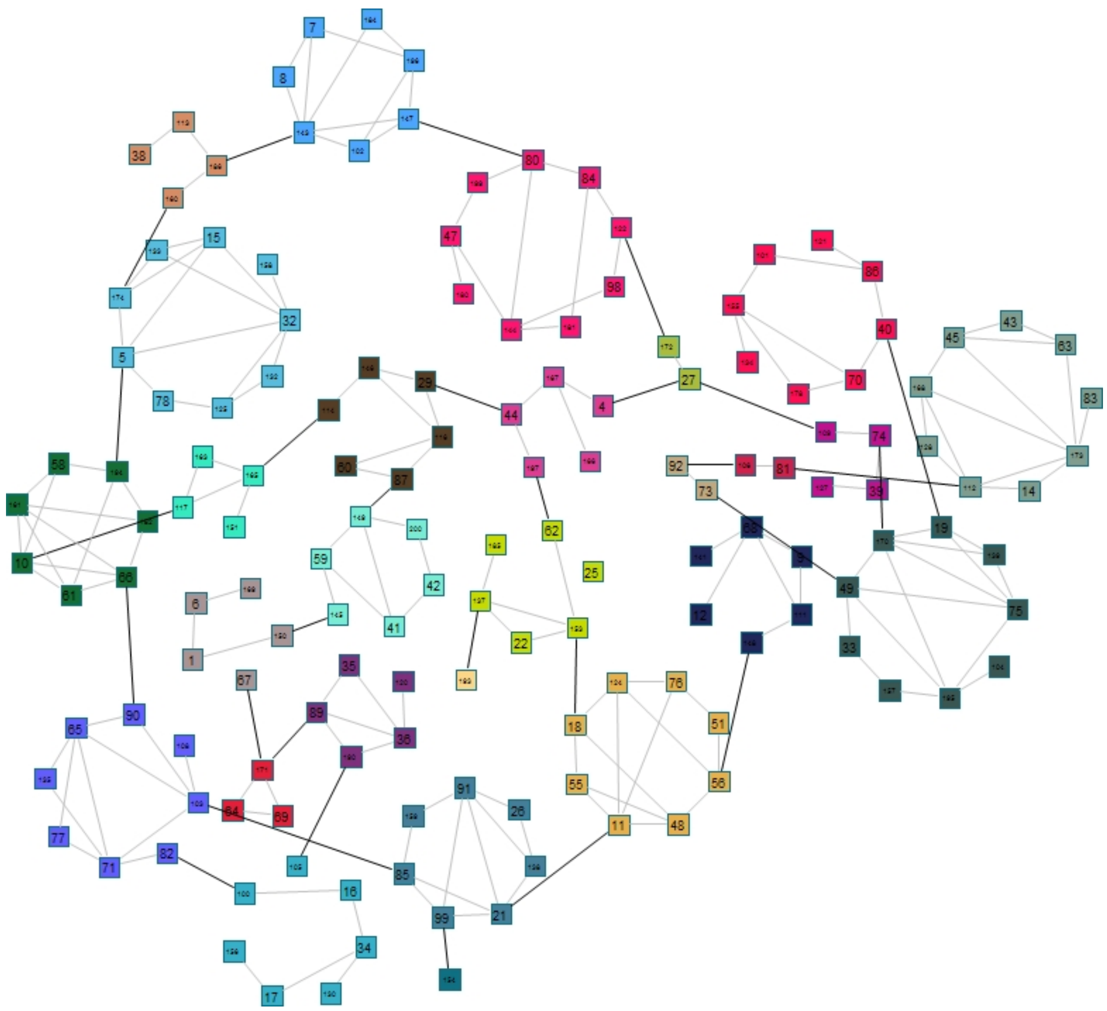

Figure A.6: A drawing created by CiSE

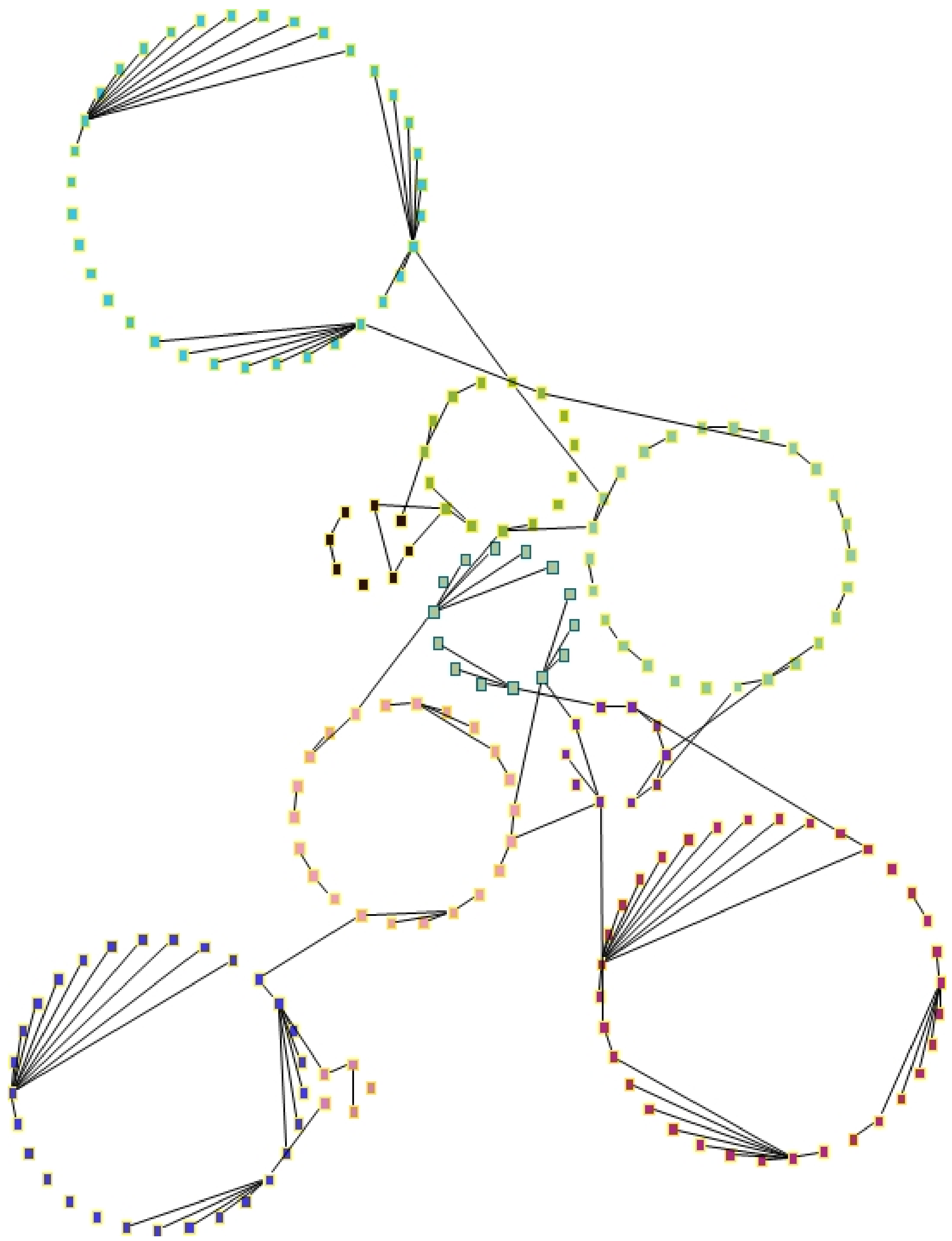

Figure A.7: A drawing created by CiSE

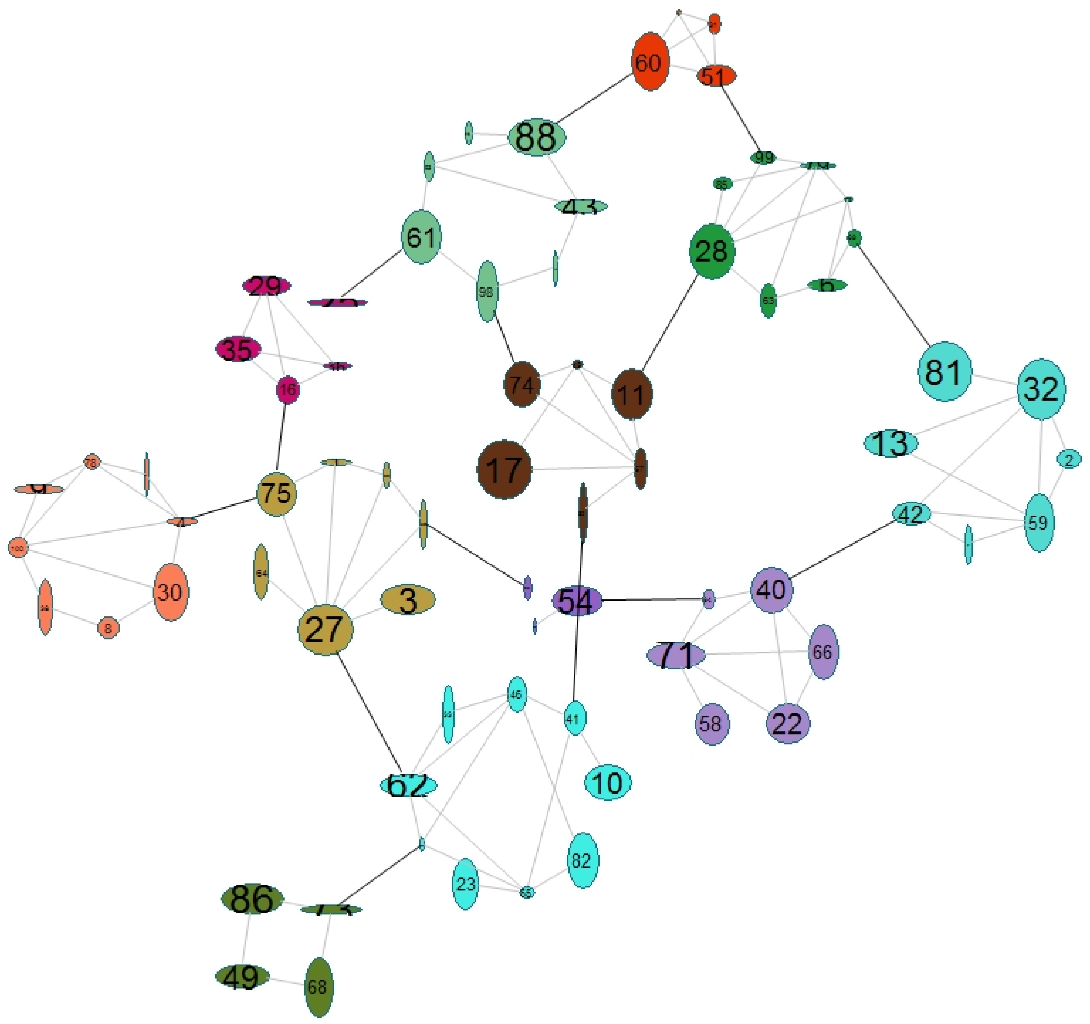

Figure A.8: A drawing of a graph containing non-uniform node dimensions created by CiSE.

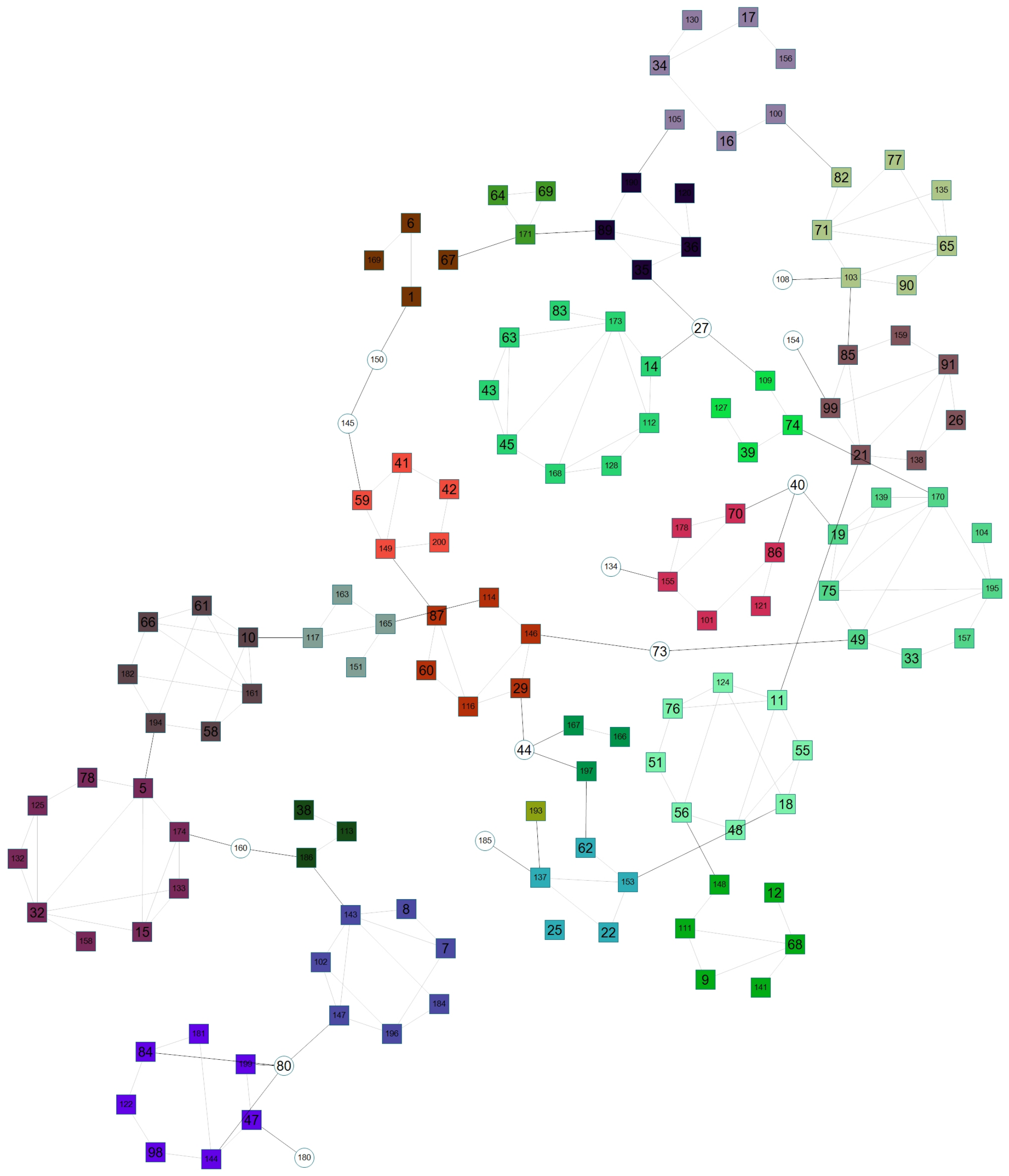

Figure A.9: A drawing of a graph containing also un-clustured nodes as well as clustered ones created by CiSE. Un-clustered nodes are represented by circles.

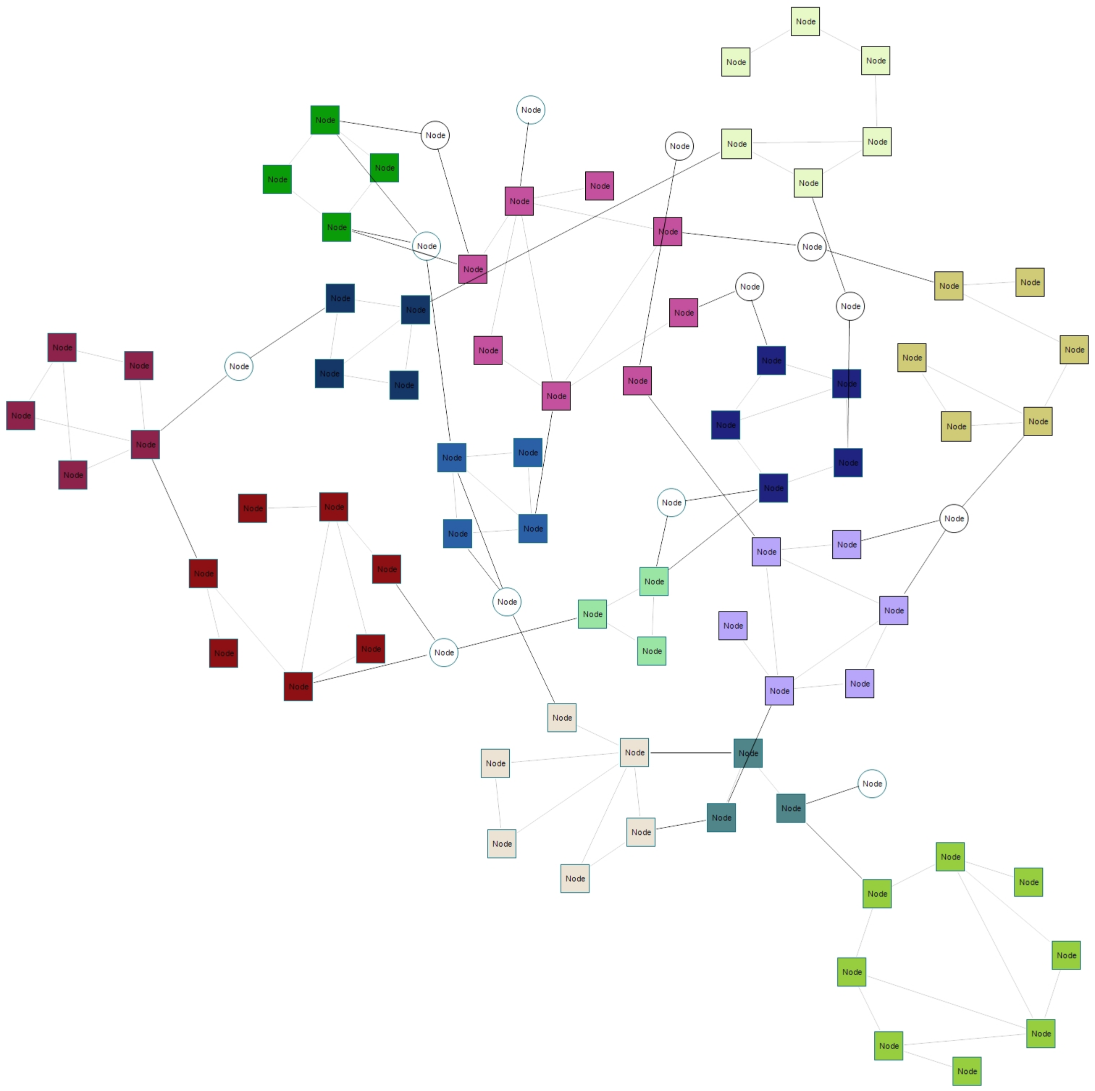

Figure A.10: A drawing of a graph containing also un-clustured nodes as well as clustered ones created by CiSE. Un-clustered nodes are represented by circles.

Bibliography

[1] Govisual Software Libraries. Oreas GmbH., Köln, Germany. http://www.oreas.com/.

[2] Aisee User Manual. AbsInt Angewandte Informatik GmbH., Saarbruecken, Germany, 2000-2005. http://www.aisee.com/.

[3] Chisio: Compound or hierarchical graph visualization tool. http://www.cs.bilkent.edu.tr/ ivis/chisio.html, 2007.

[4] M. Baur and U. Brandes. Crossing reduction in circular layouts. *Technical Report 2004-14, Universitat Karlsruhe (TH), Fakultat fur Informatik*, 2004.

[5] D. Beyer. Ccvisu: automatic visual software decomposition. In *ICSE Companion '08: Companion of the 30th international conference on Software engineering*, pages 967–968, New York, NY, USA, 2008. ACM.

[6] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing, Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

[7] U. Dogrusöz, B. Madden, and P. Madden. Circular layout in the Graph Layout Toolkit. In *GD '96: Proceedings of the Symposium on Graph Drawing*, pages 92–100, London, UK, 1997. Springer-Verlag.

[8] P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combinatoria*, page 21(A):8998, 1986.

[9] G. Erhard. Graphs as structural models: The application of graphs and multigraphs in cluster analysis. *Advances in system analysis vol. 4*, 1988.

[10] Q. Feng. *Algorithms for Drawing Clustered Graphs.* PhD thesis, 1997.

[11] Q.-W. Feng, R. F. Cohen, and P. Eades. How to draw a planar clustered graph. In *COCOON '95: Proceedings of the First Annual International Conference on Computing and Combinatorics*, pages 21–30, London, UK, 1995. Springer-Verlag.

[12] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In R. Tamassia and I. Tollis, editors, *GD '94*, volume 894 of *Lecture Notes in Computer Science*, pages 388–403. Springer-Verlag, 1995.

[13] E. Giral. *A layout algorithm for undirected compound graphs.* PhD thesis, Computer Engineering Department, Bilkent University, 2005.

[14] V. Guernier, C. Sola, K. Brudey, J. F. Guegan, and N. Rastogi. Use of cluster-graphs from spoligotyping data to study genotype similarities and a comparison of three indices to quantify recent tuberculosis transmission among culture positive cases in french guiana during a eight year period. *BMC Infectious Diseases*, 8(1):46+, April 2008.

[15] D. Harel and Y. Koren. Drawing graphs with non-uniform vertices. In *Working Conference on Advanced Visual Interfaces (Proc. AVI'02)*, pages 157–166. ACM Press, 2002.

[16] H. He and O. Skora. New circular drawing algorithms. In *Workshop on Information Technologies - Applications and Theory (ITAT04)*, 2004.

[17] JViews User's Guide. 94253 Gentilly Cedex, France, 2002. http://www.ilog.com.

[18] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, April 1989.

[19] G. Karypis, V. Kumar, and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48:71–95, 1998.

[20] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(1):291–307, 1970.

[21] M. Lorr. *Cluster analysis for social scientists / Maurice Lorr.* Jossey-Bass, San Francisco :, 1st ed. edition, 1983.

[22] E. Makinen. On circular layouts. *International Journal of Computer*, 1988.

[23] A. Quigley and P. Eades. Fade: Graph drawing, clustering, and visual abstraction. pages 77–80. 2001.

[24] S. C. R. Brockenauer. *Drawing Graphs Methods and Models, Chapter 8. Drawing Clusters and Hierarchies.* Springer Berlin / Heidelberg, 2001.

[25] H. D. Simon and S.-H. Teng. How good is recursive bisection? *SIAM J. Sci. Comput*, 18:1436–1445, 1995.

[26] J. M. Six and I. G. Tollis. A framework and algorithms for circular drawings of graphs. *Journal of Discrete Algorithms*, 4(1):25 – 50, 2006.

[27] H. Spath. Cluster analysis algorithms for data reduction and classifcation of objects. Horwood, 1980.

[28] K. Sugiyama and K. Misue. A simle and unified method for drawing graphs: Magnetic-spring algorithm. In R. Tamassia and I. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 364–375. Springer-Verlag, 1995.

[29] S. Teng. *Points, spheres, and separators: A unifed geometric approach to graph partitioning.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[30] Tom Sawyer Visualization Documentation. Tom Sawyer Software, Oakland, CA, USA, 1992-2005. http://www.tomsawyer.com.

[31] Wikipedia. Cluster analysis. http://en.wikipedia.org/wiki/Cluster_analysis, 2009.

[32] Wikipedia. Planar graph. http://en.wikipedia.org/wiki/Planar_graph, 2009.

[33] yFiles User's Guide. yWorks GmbH, D-72076 Tübingen, Germany, 2002. http://www.yworks.de.