CiSE: A Circular Spring Embedder Layout Algorithm

Ugur Dogrusoz, *Senior Member*, *IEEE*, Mehmet E. Belviranli, and Alptug Dilek

Abstract—We present a new algorithm for automatic layout of clustered graphs using a circular style. The algorithm tries to determine optimal location and orientation of individual clusters intrinsically within a modified spring embedder. Heuristics such as reversal of the order of nodes in a cluster and swap of neighboring node pairs in the same cluster are employed intermittently to further relax the spring embedder system, resulting in reduced inter-cluster edge crossings. Unlike other algorithms generating circular drawings, our algorithm does not require the quotient graph to be acyclic, nor does it sacrifice the edge crossing number of individual clusters to improve respective positioning of the clusters. Moreover, it reduces the total area required by a cluster by using the space inside the associated circle. Experimental results show that the execution time and quality of the produced drawings with respect to commonly accepted layout criteria are quite satisfactory, surpassing previous algorithms. The algorithm has also been successfully implemented and made publicly available as part of a compound and clustered graph editing and layout tool named CHISIO.

Index Terms—Information visualization, visualization techniques and methodologies, visualization systems and software, graph algorithms, algorithm design and analysis, graph visualization, graph drawing, force-directed layout, circular layout, clustered graphs, sequence alignment

1 INTRODUCTION

MANY complex systems in nature and society can be described in terms of networks capturing the intricate web of connections among the units they are made of [1]. Such networks typically contain parts in which the nodes (units) are more highly connected to each other than to the rest of the network. The sets of such nodes are usually called clusters, communities, cohesive groups or modules.

As graphical user interfaces have improved, and more state-of-the-art software tools have incorporated visual functions, interactive network editing and diagramming facilities have become important components in visualization systems [2]. Effective analysis of the underlying data in network or graph visualization is only possible with sound automatic layout capabilities of such systems.

Circular drawings are widely used in visualization of clustered networks. In a circular drawing of a graph, the nodes of each cluster are placed onto the circumference of a large-enough circle. Some circular drawings place hub nodes, or nodes only connected to nodes in the same cluster, at the center of the circle as well. Clustered views are required by many visualization applications for computer, telecommunication or social networks, web graphs, and biology applications. Emphasizing natural groupings or semantic qualities represented with clusters is of great help in analysis of the underlying relational data (Fig. 1).

- *U. Dogrusoz is with the Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey. E-mail: ugur@cs.bilkent.edu.tr.*
- *M.E. Belviranli is with the Department of Computer Science and Engineering, University of California, Riverside, 351 Winston Chung Hall, Riverside, CA 92521. E-mail: belviram@cs.ucr.edu.*
- *A. Dilek is with The Scientific & Technological Research Council of Turkey, TUBITAK-BILGEM, Cukurambar Mah. 1478. Cadde No:22, Ankara 06100, Turkey. E-mail: alptug.dilek@tubitak.gov.tr.*

Manuscript received 20 July 2011; revised 18 Apr. 2012; accepted 17 Aug. 2012; published online 4 Sept. 2012.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2011-07-0160. Digital Object Identifier no. 10.1109/TVCG.2012.178.

There has been a great deal of work done on layout of clustered graphs using various representations or approaches, including c-planar embeddings of hierarchical clustered graphs [5], compound digraphs [6], and modified force-directed approaches [7], as detailed in [8]. A reasonable amount specifically focuses on circular layout [3], [4], [9], [10], [11], [12], [13], [14], but only a few [3], [4] address the respective layout of clusters (i.e., the layout of a quotient graph) as well as the layout of individual clusters. The only previous algorithms to handle quotient graphs of arbitrary structure (i.e., does not assume it to be acyclic) are presented in [3] and [4].

The major drawback of the algorithm in [3] is that when the quotient graph of a clustered graph is cyclic, all clusters except for those on the acyclic parts of the quotient graph end up on a single large "backbone" circle in the middle, inevitably introducing many intercluster edge crossings (Fig. 2). Especially those intercluster edges between clusters on this backbone structure are very long compared to their intracluster counterparts.

Six and Tollis [4], on the other hand, describe a multistage circular layout algorithm. First, the layout of the quotient graph is calculated using the force-directed approach to determine and finalize the positions of each cluster. Then, a brute-force search is used to optimally orient/rotate each cluster in its fixed location. Lastly, a relaxation stage is performed to potentially reduce edge crossings by "pulling" on-circle nodes toward their neighbors in other clusters. From the rather rough descriptions and a single example drawing provided (Fig. 3), it seems the algorithm is still immature with the following drawbacks:

- Cluster positions are calculated without taking the optimal orientation of the cluster into account, and might lead to unnecessarily long (and nonuniform) intercluster edges.
- Nodes of a cluster are likely to be nonuniformly distributed around the associated circle making the

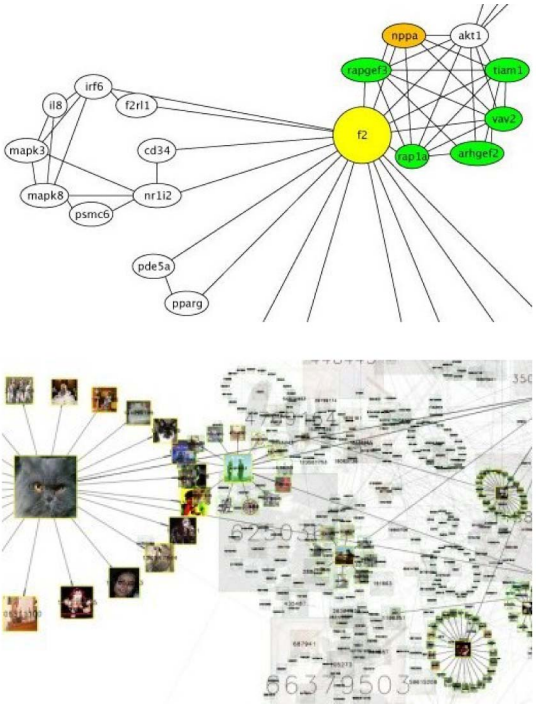


Fig. 1. Parts of sample drawings using circles to view clusters in biological (top—courtesy of Team PMAP) and social (bottom—courtesy of VisualComplexity.com) networks.

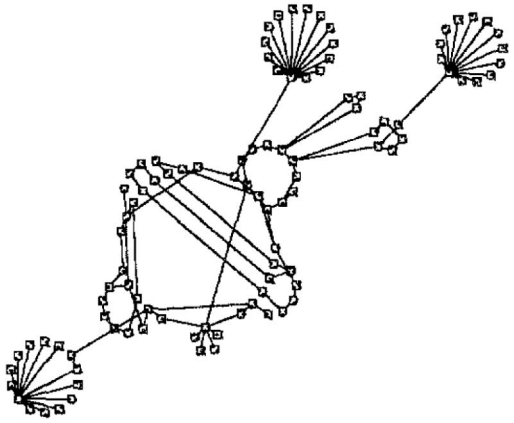


Fig. 2. Layout of a clustered graph by circular layout of the GLT described in [3].

distinction of a cluster from another less obvious as well as destroying the circular look of a cluster.

- The last step of the algorithm might actually destroy the optimal ordering within the cluster calculated.
- Separate force-directed-based relaxation methods are used for placing clusters and for determining the ordering of each node in each cluster. This also makes it difficult to customize the algorithm for domain-specific applications.

In this paper, we describe a novel algorithm for the circular layout of clustered graphs, named *C*ircular *S*pring *E*mbedder (CiSE). CiSE overcomes the drawbacks of the one in [4] and fulfills the four goals described in that paper for visualizing clustered graphs as circular drawings:

- Highly visible clusters: Nodes in a cluster are evenly separated around the associated circle, and circles representing distinct clusters repulse each other to avoid overlaps.
- Low intracluster edge crossing number: Initial locally optimal node placement around a circle is kept throughout the algorithm; heuristics to improve on the inter-cluster edge crossing number never destroy this optimality.
- Low intercluster edge crossing number: This is where other algorithms suffer most; ours, on the other hand, determines optimal location and orientation of individual clusters intrinsically within a modified spring embedder.
- Fast execution time: On-circle nodes are ignored for node-node repulsion calculations avoiding a quadratic running time on the total number of nodes. Graphs with 800-1,000 nodes can be laid out within a second or two.

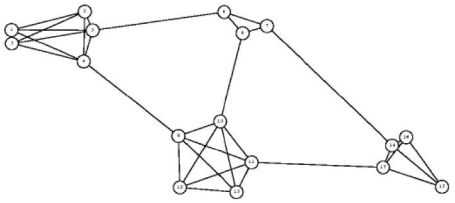


Fig. 3. A sample four-cluster graph laid out using a preliminary implementation of the circular layout algorithm of [4].

Our algorithm is a truly force-directed layout algorithm that treats nodes of a cluster as a group; individual clusters rotate and translate as needed by the physical system to reach a minimal energy. Thus, it could be easily customized for domain-specific applications. In addition, the order of nodes in a cluster can be reversed, and neighboring nodes on a circle are allowed to swap to further relax the system. Moreover, a user-specified portion of high degree nodes in a cluster can be optionally placed inside the associated circle to reduce the size of the circle. The algorithm has been implemented and made publicly available within a compound and clustered graph editing and layout tool named CHISIO.

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. The neighbors of a node v denoted by $N(v)$ are exactly the nodes in $\{w \mid \{v, w\} \in E\}$. A *clustered graph* is a graph $G = (V, E)$ with a partition $C = \{C_1, C_2, \ldots, C_k\}$ on the clustered node set, where each $C_i, i = 1, \ldots, k$, corresponds to a *cluster*, $C_i \cap C_j = \emptyset$ for all $i, j = 1, \ldots, k$, $k \geq 1$, and $V = \sum_{i=1}^{k} C_i \cup C_{k+1}$, and C_{k+1} denotes a possibly empty *unclustered* node set.

An edge is called an *intracluster* edge if both its ends belong to the same cluster; an *intercluster* 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:

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

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

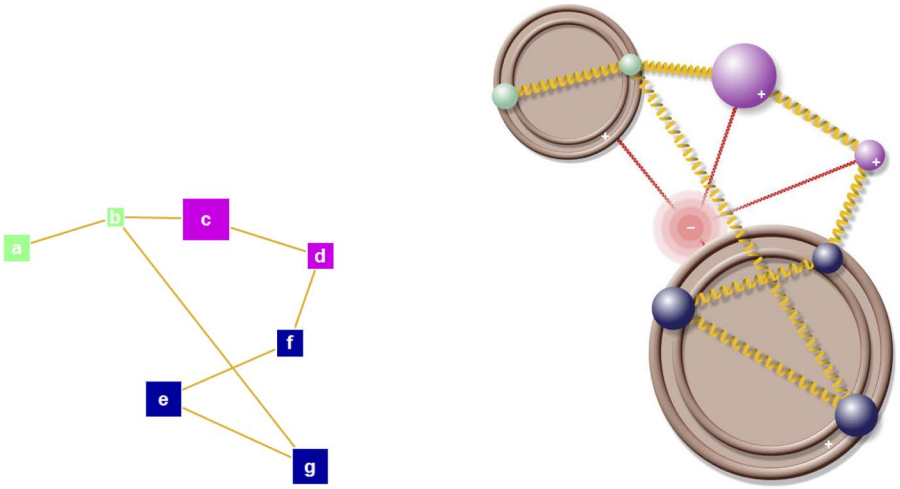


Fig. 4. A sample clustered graph with two clusters $C_1 = \{a, b\}$ and $C_2 = \{e, f, g\}$, and unclustered nodes $\{c, d\}$ (left), and the corresponding physical model used by our algorithm (right).

cluster is called an *on-circle* node, and the cluster node may be referred to as the *owner circle* of this on-circle node. Those on-circle nodes with neighbors outside the cluster are called *out-nodes*, while others with no neighbors outside the cluster are called *in-nodes*.

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 a quotient graph corresponding to clusters

$$\mathbf{V_c(G)} = \{u_i \mid u_i \in \mathcal{V}(\mathcal{G}) \ \wedge \ u_i \notin C_{k+1}\},$$

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

$$\mathbf{V_o(G)} = \left\{ v \mid v \in V(G) \ \wedge \ v \in \bigcup_{i=1}^{k} C_i \right\},$$

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

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

For the example clustered graph in Fig. 4, we have

$$\mathbf{V(G)} = \{a, b, c, d, e, f, g, C_1, C_2\}, \mathbf{V_c(G)} = \{C_1, C_2\},$$
$$\mathbf{V_o(G)} = \{a, b, e, f, g\}, \text{ and } \overline{\mathbf{V}}_\mathbf{o}\mathbf{(G)} = \{c, d, C_1, C_2\}.$$

3 Layout Algorithm

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

3.1 Underlying Physical Model

We chose a basic force-directed layout algorithm with certain extensions to satisfy the clustering conventions in circular drawings, where the basic idea is to simulate a physical system in which nodes are assumed to be physical objects with certain "electrical charges," connected via "springs" of a prespecified desired length. Objects pull or repel each other depending on the lengths of the springs. In addition, repulsion forces act on any pair of objects that are "too close" to each other to avoid node-to-node overlaps. Furthermore, we assume relatively minor "gravitational forces" to keep graph components together (i.e., when the quotient graph is disconnected). Thus, the optimal layout is regarded as the state of this system in which total energy is minimal. This basic model has proven to be successfully extended for producing specialized layouts in the past [15], [16].

The use of extra constraints for producing circular drawings is implemented by introducing the following extra properties to the physical model used by the spring embedder, trying to obey basic (Hooke's and Coulomb's) laws of physics. Each cluster/circle is represented by a "metanode" of circular shape, on whose periphery a round track sits. The physical entity for each member node of a cluster is assumed to be either *fixed* (pinned down to its owner circle) or *flexible* (via swapping with its neighbors) to move around the track on which it sits as needed by the different steps of the algorithm. For practical purposes and ease of implementation, we assume on-circle nodes can only move through swaps with neighboring on-circle nodes in a discrete manner, as opposed to freely moving around the track in a continuous manner. On-circle nodes move as their owner circle nodes do. This fulfills the requirement of member nodes staying on the periphery of the owner circle.

In addition, we assume a center of gravity in the middle of the bounding rectangle of the current drawing. All unclustered nodes and cluster nodes (i.e., all nodes except member nodes of a cluster) are attracted toward this center. This should keep disconnected parts of a graph together.

Furthermore, to handle varying node sizes (especially larger cluster nodes) and avoid overlaps with neighboring

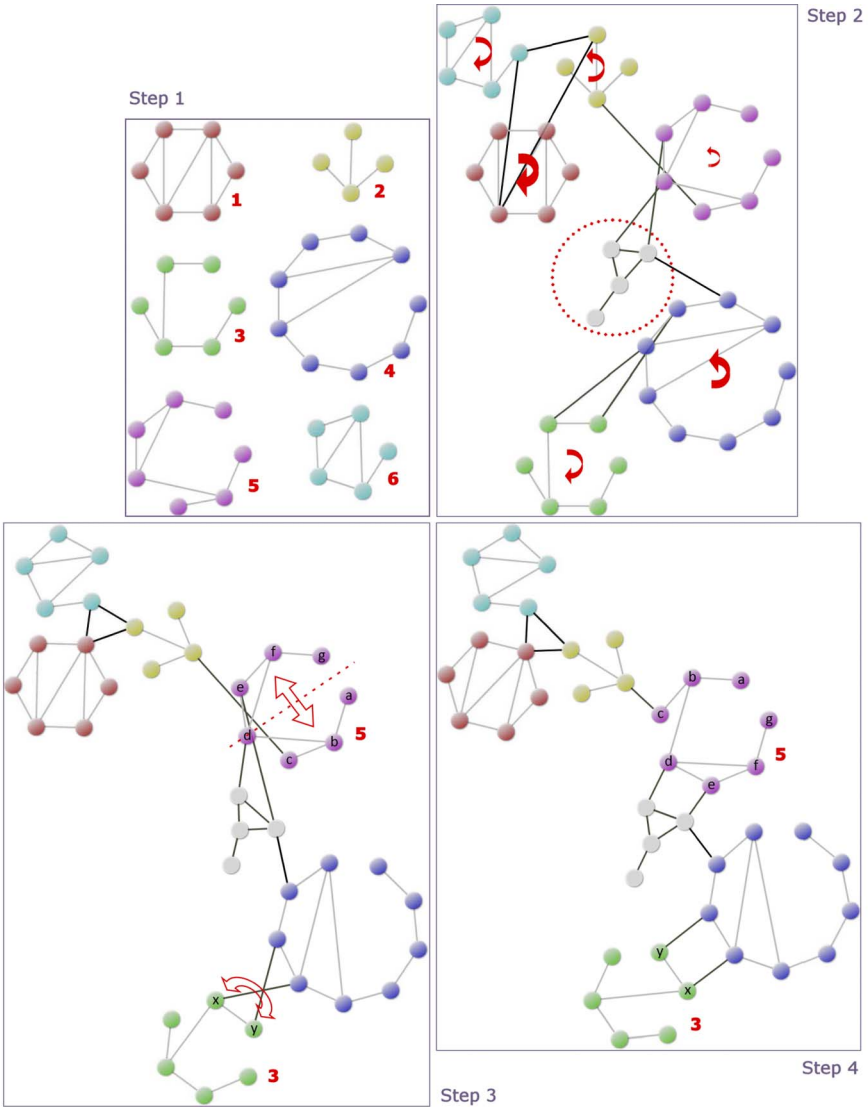


Fig. 5. Step 1: Individual clusters were laid out independently. Step 2: Skeleton graph was laid out (unclustered nodes are marked with a dashed circle). The arrow in the middle of each cluster indicates the direction and amount (the thicker the more) by which the cluster will rotate in the next step. Step 3: Clusters were allowed to rotate to relax the system. In the next step, cluster #5 is to be flipped as shown, and marked neighboring nodes of cluster #3 are to be swapped. Step 4: Clusters were allowed to be flipped, and neighboring cluster nodes were allowed to swap to further relax the system.

nodes, distance calculations are based on the borders of nodes, as opposed to their centers [17]. Fig. 4 exemplifies the basics of our physical model.

3.2 Main Idea

The CiSE algorithm is composed of five major steps preceded by an initialization phase:

- *Initialization:* This is where the necessary structures for layout, along with the quotient graph of the graph to be laid out, are constructed.
- *Step 1:* Each cluster is laid out independently using a circular layout algorithm of the user's choice (e.g., [10]).
- *Step 2:* We determine the "skeleton" of the layout by laying out the quotient graph. The specific algorithm depends on the structure of the quotient graph. If it is a tree, a radial layout is ideal. For the general case, the best choice seems to be a regular spring embedder with random initial positioning of nodes. Note, however, that the dimensions of nodes on this graph will be nonuniform, requiring extra attention for calculating edge lengths.
- *Step 3:* In this step, our aim is to reposition/rotate circles according to the location of their out-nodes and intercluster 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 with the previous step is that, we allow a cluster to be "flipped" by reversing the node order in that cluster, and on-circle nodes to move with respect to their parent circle (as well as moving with them) by swapping them with their neighbors as needed. These heuristics aim to decrease the edge crossing number. Optional post-processing, on the other hand, allows positioning of up to a user-specified portion of high degree nodes of a cluster inside the circle, reducing the drawing area used by the resulting layout. Only in-nodes for

which it is unlikely to introduce new crossings can be chosen for this purpose.

- *Step 5:* The final, polishing step is more or less identical to Step 3. Here, we finalize the positions of all nodes with a fixed layout of individual clusters, where circle nodes are allowed to move or rotate. In this phase, we set the desired intercluster edge length to be larger than intracluster ones to better separate clusters, enhancing visibility of the clustering structure.

Fig. 5 illustrates how the layout improves with each step with an example. Steps 3 through 5 make up the core of our algorithm. In the remainder of this section, we describe how we calculate and make use of different kinds of forces as part of our modified spring embedder for clustered graphs.

3.3 Force Calculations

Here, we assume that force calculations use the model described by Fruchterman and Reingold in [18] but these could be based on any other force-directed layout algorithm.

The formula for the spring force on edge $e = \{u, v\}$ is

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

where λ is the ideal edge length, η is the elasticity constant of the edge, p_u and p_v are positions of nodes u and v, respectively, and $\vec{p_u p_v}$ denotes the unit length vector pointing from p_u to p_v. Ideal edge length of intercluster edges should be chosen to be reasonably larger than that of intracluster ones to better separate the clusters during the polishing phase. In addition, nonuniform node dimensions require force calculations to be based on clipping points, where the line segment of an edge from the center of one end to the other intersects the boundaries of the end nodes, rather than node centers. Furthermore, spring forces for intracluster edges are ignored except during step 4, as we assume the nodes to be fixed; such forces would have canceled each other if they were to be transferred to their owner circles. The 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** $step = 5$ **then**
4) $idealLength :=$ *INTER_CLUSTER_COEFF* $* \lambda$
5) **if** $step = 4$ **or** e is an inter-cluster edge **then**
6) $c_u := u.boundRect \cap$ LINESEGMENT ($u.center$, $v.center$)
7) $c_v := v.boundRect \cap$ LINESEGMENT ($u.center$, $v.center$)
8) $\vec{S_{uv}} := (idealLength - \|c_u - c_v\|)^2 / \eta \cdot \vec{c_u c_v}$
9) $\vec{S_u} \mathrel{+}= \vec{S_{uv}}$
10) $\vec{S_v} \mathrel{-}= \vec{S_{uv}}$

Here, a user option *INTER_CLUSTER_COEFF* may be used to adjust how the desired edge length of intercluster edges should differ from that of intracluster ones. The overall time complexity of this method is $\Theta(|E(G)|)$ as all steps inside the for-loop can be processed in $\Theta(1)$ steps.

Node-to-node repulsion forces are calculated using

$$\vec{R_{uv}} = \frac{\alpha}{\|p_u - p_v\|^2} \; \vec{p_u p_v},$$

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

algorithm CALCREPULSIONFORCES(Graph G, int $step$)
1) **for** each pair of nodes $u, v \in \overline{\mathbf{V}}_\mathbf{o}(\mathbf{G})$ **do**
2) $c_u := u.boundRect \cap$ LINESEGMENT ($u.center$, $v.center$)
3) $c_v := v.boundRect \cap$ LINESEGMENT ($u.center$, $v.center$)
4) **if** $\|c_u - c_v\| \leq$ *REPULSION_RANGE* **then**
5) $\vec{R_{uv}} := \alpha / \|c_u - c_v\|^2 \cdot \vec{p_u p_v}$
6) $\vec{R_u} \mathrel{+}= \vec{R_{uv}}$
7) $\vec{R_v} \mathrel{-}= \vec{R_{uv}}$

Here, a user option *REPULSION_RANGE* may be used to determine node pairs that are too far from each other to take repulsions into account. Steps 2-7 are handled in $\Theta(1)$ steps, which are executed a total of maximum $|\overline{\mathbf{V}}_\mathbf{o}(\mathbf{G})|^2$ times, making the overall complexity of the method $O(|\overline{\mathbf{V}}_\mathbf{o}(\mathbf{G})|^2)$.

Gravitational forces have a fixed magnitude toward the center of the graph, where $\vec{p_u p_c}$ is the unit vector from the position of node u to the center of the graph: $\vec{G_u} = \gamma \cdot \vec{p_u p_c}$.

algorithm CALCGRAVITATIONFORCES(Graph G)
1) $center := G.boundRect$
2) **for** each $u \in \overline{\mathbf{V}}_\mathbf{o}(\mathbf{G})$ **do**
3) calculate gravitational force $\vec{G_u}$ towards $center$

The time complexity of this method is $\Theta(|\overline{\mathbf{V}}_\mathbf{o}(\mathbf{G})|)$. Notice, however, that gravitation needs to be applied to disconnected graphs only.

Fig. 6 shows with an example how forces are calculated for each node. In each iteration, once all types of forces are calculated, they are aggregated to determine the total force on each node. In addition, the total force of each on-circle node is transferred to its owner circle node for translating that node. Furthermore, the horizontal component of this force, which is tangential to the owner circle at the location of the force, contributes to the total force rotating the owner circle. For both translating and rotating nodes, the current temperature maintained as part of a global linear cooling schema is taken into account.

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

As we will discuss later on, swaps are performed periodically during step 4. During each swap cycle, we first collect rotational force information D_u for each on-circle node u (swap preparation phase), and then an iteration is dedicated to actually performing a swap of u with a neighbor if certain conditions are met (swap phase). After the total forces are calculated and transferred as needed during an iteration, we translate each node with

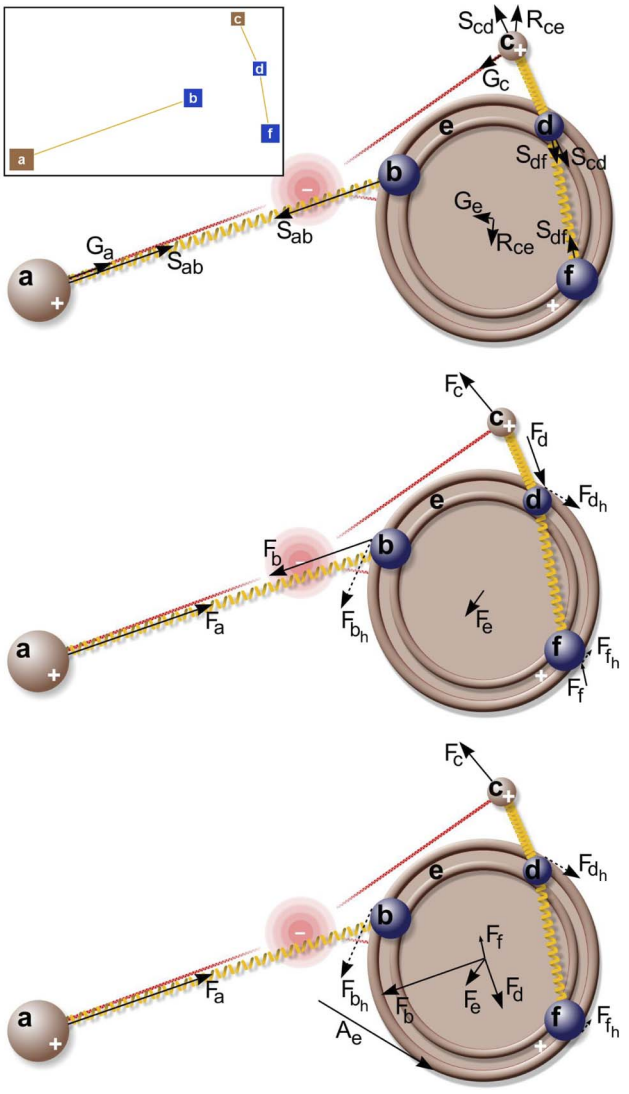


Fig. 6. Spring, repulsion, and gravitational forces are marked on the underlying physical model for a sample clustered graph with a cluster of nodes $\{b, d, f\}$ and unclustered nodes $\{a, c\}$; node e represents the single cluster of G in quotient graph of G (top). The distance between nodes a and e is assumed to be larger than the repulsion range. Gravitational forces are included for completeness for this connected graph. Total forces acting upon each node for the sample graph are shown; horizontal components of the forces for on-circle nodes are also shown (middle). Forces acting upon on-circle nodes are transferred to the owner circle node for translation: $\vec{F}_e := \vec{F}_e + \vec{F}_b + \vec{F}_d + \vec{F}_f$. In addition, their horizontal components contribute to the rotation of the owner circle node: $A_e := \|\vec{F}_{b_h}\| + \|\vec{F}_{d_h}\| + \|\vec{F}_{f_h}\|$ (bottom).

respect to the final total force acting upon it. In the case of circle nodes, we also rotate such nodes proportional to the magnitude of the total rotational force acting on the node. We limit the movement of each node in each iteration to avoid drastic movements, often resulting in oscillations.

algorithm MOVENODES(Graph G, int $step$)
1) **for** each $u \in \overline{\mathbf{V}}_\mathbf{o}(\mathbf{G})$ **do**
2) **if** $u \in \mathbf{V_c}(\mathbf{G})$ **then**
3) move it using $\max(\vec{F}_u, MAX\_DISP\ )$ / #nodes in u
4) rotate it using A_u / #nodes in u
5) **else**
6) move it using $\vec{F}_u$

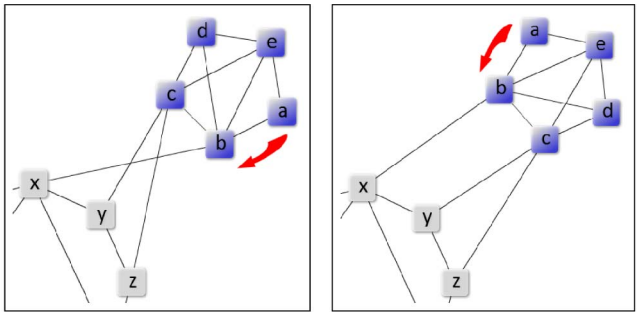


Fig. 7. Part of a clustered graph, where intercluster edge crossings of a 5-node cluster (left) is eliminated by reversal of the cluster (right).

The main steps 3, 4, and 5 make up a spring embedder by using algorithms described earlier.

algorithm PERFORMSTEP3-5(Graph G, int $step$)
1) $iter := maxIterCount[step]$
2) $totalDisp := 0$
3) **while** $(iter > 0$ **and** $totalDisp > dispThreshold[step])$ **do**
4) CALCSPRINGFORCES(G, $step$)
5) CALCREPULSIONFORCES(G, $step$)
6) CALCGRAVITATIONFORCES(G)
7) CALCTOTALFORCES(G, $step$)
8) $totalDisp :=$ MOVENODES(G, $step$)
9) $iter := iter - 1$

Here, method MOVENODES returns the total displacement of nodes during this iteration, and arrays $maxIterCount$ and $dispThreshold$ maintain values of the parameters for the maximum number of iterations to be performed and total displacement threshold used to determine convergence for each step, respectively.

Suggested default values for various parameters used in force calculations are listed in the supplemental document, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.178.

3.4 Decreasing Edge Crossing Number

We improve the edge crossing number in two ways. One is by reversing the order of nodes on a circle, and the other is by swapping neighboring on-circle node pairs where appropriate. Both heuristics are applied intermittently during step 4.

3.4.1 Reversing Node Order

Step 1 of our algorithm makes use of an existing circular layout algorithm to lay out individual clusters. Such algorithms output a locally optimal ordering for the layout of input nodes on a circle to minimize the number of edge crossings. We use this ordering to place nodes in a clockwise manner around the associated circle. However, placing such nodes in anticlockwise manner would also yield an equal number of intracluster edge crossings, and in some cases result in a smaller number of intercluster edge crossings (Fig. 7).

To determine whether clockwise or anticlockwise ordering of nodes in a cluster with at least two intercluster edges would result in a more compatible layout of the cluster with its neighbors, during step 4 we periodically apply sequence alignment.

A sequence alignment is a way of arranging the sequences of structures (e.g., DNA) to identify regions of similarity. Such similarities are generally attributed to functional, structural, or evolutionary relationships between whatever the sequences are representing. It has applications in many areas including bioinformatics.

More formally, an alignment $\mathcal{A}$ of two strings x and y of length n and m, respectively, is a sequence of ordered pairs of the form (x_i, y_j), $(x_i, -)$, and $(-, y_j)$ that preserves the order of sequence positions in both x and y. A maximal sequence of $(x_i, -)$ pairs is called a deletion, while a maximal sequence of $(-, y_j)$ is called an insertion, both introducing gaps in the alignment. The similarity score $S(\mathcal{A})$ of the alignment $\mathcal{A}$ is generally assumed to be the sum of scores for individual substitutions (match or mismatch), insertions, and deletions.

In the case of cyclic sequences, insertions and deletions may wrap around the ends. Thus, the cyclic score $S_C(\mathcal{A})$ may be larger than the score $S(\mathcal{A})$ of the linear representation of the alignment $\mathcal{A}$. The cyclic shift operator σ rotates a string or an alignment by one position: $\sigma(x) = (x_2, \ldots, x_{n-1}, x_n, x_1)$. The cyclic score of the alignment is thus

$$S_C(\mathcal{A}) = \max_k S(\sigma^k(\mathcal{A})),$$

under the above additivity assumption on the scoring model [19].

To determine whether or not node order should be reversed, we construct two strings, one corresponding to the order of the nodes in the cluster, and the other representing the angular order of the neighboring nodes of the cluster with respect to the cluster center, as detailed below.

Let $(v_1, \ldots, v_k)$ denote the on-circle nodes of a cluster C as ordered clockwise on the circle. The order of nodes is coded as a string $x = x_1 x_2 \ldots x_l$, $l \geq k$, where node v_i with intercluster edge degree d_i is represented with substring $x_j x_{j+1} \ldots x_{j+d_i-1}$ such that $j \geq i \ \wedge \ x_j = x_{j+1} \ldots = x_{j+d_i-1}$ if $d_i \geq 2$, and with just $x_j, j \geq i$, otherwise. In other words, each on-circle node v_i is represented with a unique character x_j, which is duplicated for each incident multi intercluster edge. As an example, for the 5-node cluster in Fig. 7, substrings $x =$ abccde and $\bar{x} =$ aedccb could be used for original and reversed order of the nodes in the cluster, respectively.

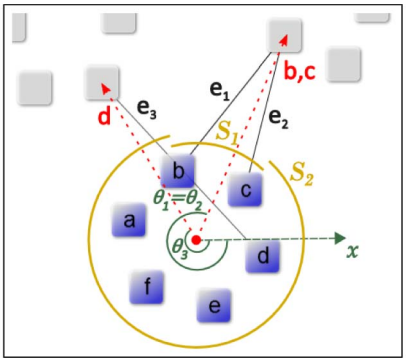


Fig. 8. Intercluster edges $\{e_1, e_2, e_3\}$ of a 6-node cluster are ordered with respect to their angles as (e_3, e_1, e_2) according to the described method. Notice here that since $\theta_1 = \theta_2$ we apply the tie-break procedure and place e_1 before e_2 as the b-c circular segment S_1 is shorter than the c-b segment S_2, where b and c are on-circle end nodes of e_1 and e_2, respectively.

Let $\{e_1, \ldots, e_m\}, m \geq 2$, be the intercluster edges of C and $w_i, 1 \leq i \leq m$, denote the end node of e_i not in C. Now imagine a vector $\vec{AB_i}$ for each intercluster edge e_i, where A is the center of the circle associated with cluster C and B_i is the center of node w_i. Let θ_i be the angle of the vector $\vec{AB_i}$ with the positive x-axis in radians. Assume, without loss of generality, that edges $(e_1, \ldots, e_m)$ are ordered in nondecreasing order with respect to their angles θ_i. When end nodes of two intercluster edges are the same, ties are broken in favor of the end node that comes earlier as the shorter of the two circular segments defined by the centers of the two end nodes are traversed clockwise (Fig. 8). The ordering of neighboring nodes of a cluster is based on this sorted edge list $(e_1, \ldots, e_m)$ by defining a second string $y = y_1 \ldots y_m$, where y_i is the character code of the on-circle end node of e_i, not on C.

Using the constructed strings x and y, two circular alignments are performed; one for x and y, and another for the inverse of x, $\bar{x} = x_l \ldots x_2 x_1$, and y. Should $\bar{x}$ better align with y than x itself, we reverse the order of the nodes in the cluster. Notice here that the scoring scheme used for alignment should highly reward matches, whereas mismatches (substitutions) and gaps (insertions or deletions)

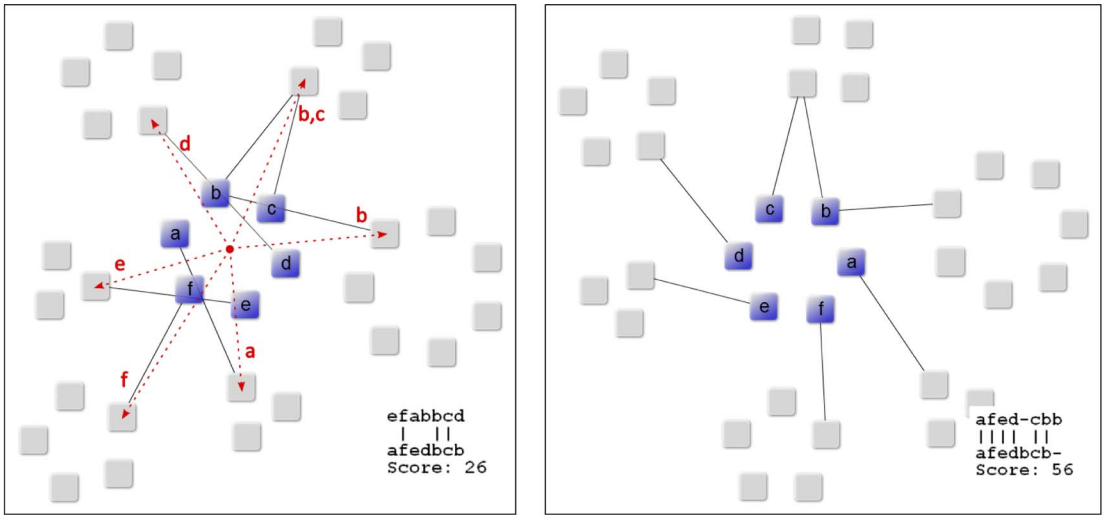


Fig. 9. The order of the nodes of a 6-node cluster is coded with the string $x =$ abbcdef, whereas the order of their neighbors is represented with the string $y =$ afedbcb, yielding a circular alignment score of 26 (left). The strings $\bar{x} =$ fedcbba (inverse of x) and y yield a score of 56 (right).

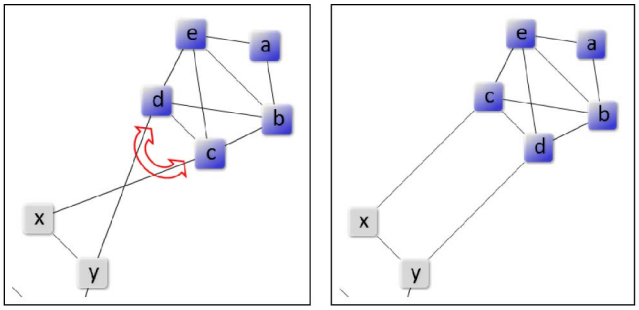


Fig. 10. Neighboring on-circle nodes c and d of the 5-node cluster are pulled in opposite directions (clockwise and anticlockwise, respectively) due to incident intercluster edges (left); during swap phase of step 4, nodes c and d are swapped to relax the system, resulting in an improvement of the intercluster edge crossing number (right).

should be penalized lightly. Since no single scoring scheme will guarantee this heuristic to perform optimally, we relied on experimental results to fine-tune the scheme.

For example, for the 6-node cluster in Fig. 9, these strings are calculated as $x =$ abbcdef and $y =$ afedbcb. Supposing we use a basic scoring scheme, where matches are rewarded 10 points, mismatches are penalized with 1 point, and deletions/insertions (i.e., gaps) cost 2 points for our circular alignment, $\bar{x}$ and y circular alignment undoubtedly outscores x and y circular alignment. This results in reversal of the 6-node cluster, resolving a number of intercluster edge crossings and node-edge overlaps.

3.4.2 Swapping Neighboring Nodes

Two neighboring on-circle nodes sometimes want to move in reverse directions due to intercluster edges incident upon them (Fig. 10). Such node pairs are allowed to swap during step 4, only if the operation does not increase the edge crossing count.

In fact, the swapping substep is composed of two phases: one, named the swap preparation phase, is dedicated to gathering information to decide whether or not neighboring on-circle nodes should be swapped, and the other, named the swap phase, actually performs any swaps that would not augment the edge crossing number of the graph.

When in swap phase, extra operations take place, forming sets of any potential swaps and performing these swaps if they do not augment the edge crossing count. Two neighboring on-circle nodes are considered for a swap if the rotational component of the associated forces are toward each other (e.g., one is in clockwise direction and the other is in counterclockwise direction). We also consider node pairs, where one wants to move toward the other node, and the other node is an in-node (no rotational force) with the hope to decrease total energy of the system. Here, we first eliminate node pairs whose swap would increase the intracluster edge crossing count. Then, we classify a node pair as "safe" when at least one of these nodes is not an out-node (i.e., would surely not augment the intercluster edge crossing), and "nonsafe" otherwise. In each swap phase, we perform all safe swaps but no more than one nonsafe swap to stay away from drastic changes in the layout. Also note that to avoid oscillations we do not swap node pairs already swapped in previous phases. The following pseudocode can be appended to the algorithm MOVENODES described earlier to apply this heuristic.

```
7) if step = 4 and in swap phase then
8)     S := N := ∅
9)     for each neighboring node pair {u, v}, u, v ∈ V_o(G)
       do
10)        if D_u and D_v are not towards each other or
           {u, v} swap augments intra-cluster edge crossings
           or {u, v} swapped in previous iteration then
11)          continue
12)        if both u and v are out-nodes then
13)          N := N ∪ {{u, v}}
14)        else
15)          S := S ∪ {{u, v}}
16)    H := BUILDMAXHEAP(N) // with |D_u − D_v| as key
17)    repeat
18)        {u, v} := EXTRACTMAX(H)
19)        if {u, v} swap does not augment inter-cluster
           edge crossings then
20)          SWAP({u, v})
21)          break
22)    until H is empty
23)    for each safe pair {u, v} ∈ S do
24)      if u and v not already involved in a swap then
25)        SWAP({u, v})
26)    for each u ∈ V_o(G) do
27)      D_u := 0
```

The worst case running time of modified MOVENODES is $O(|\overline{\mathbf{V}}_\mathbf{o}(\mathbf{G})| + |\mathbf{V}_\mathbf{o}(\mathbf{G})| \cdot \lg |\mathbf{V}_\mathbf{o}(\mathbf{G})|)$ as the maximum size of the heap can be at most $|\mathbf{V}_\mathbf{o}(\mathbf{G})|$.

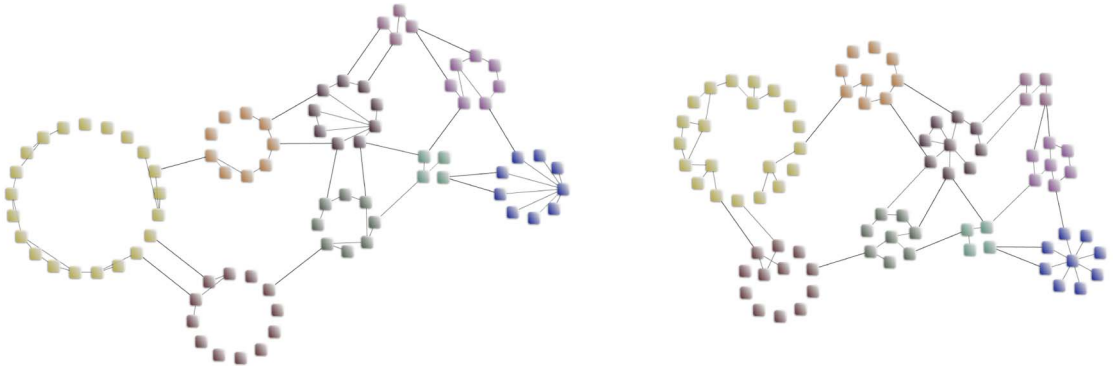


Fig. 11. The same graph laid out with CiSE where nodes inside cluster circles are disallowed (left), and allowed (right), respectively. The area of the drawing on the right is approximately 40 percent smaller.

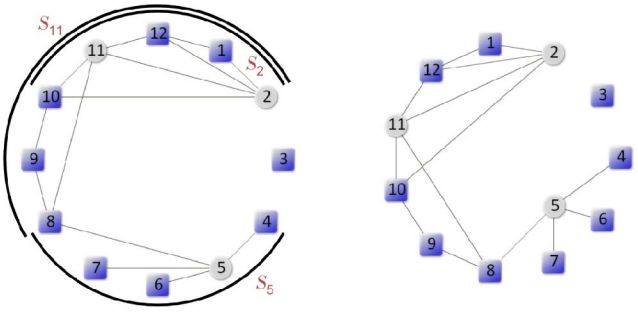


Fig. 12. An example cluster with high degree in-nodes 2, 5, and 11 as candidates to be placed inside its associated circle, along with respective circular segments S_2, S_5, and S_{11} of minimum length (left) Only node 5 satisfies the criteria to be pulled inside, resulting in reduction of the area used by the cluster by 16 percent (right).

3.5 Reducing Drawing Area

An optional postprocessing for step 4 of the algorithm tries to find nodes to move inside the circle to reduce the size of the circle for each cluster using a heuristic as follows (Fig. 11). Starting with a highest degree in-node u on cluster C, we first calculate a minimum circular segment S of the associated circle, spanning all neighbors of u on C and the node u itself. u is chosen to be moved inside if no node on this circular segment S is connected to a node outside its immediate neighbors (geometric neighbors, not necessarily joined by an edge) on C. This is to ensure that moving u inside the associated circle is not going to introduce any node-edge overlaps. For instance, node 5 of the cluster in Fig. 12 satisfies this criteria and may be pulled inside, whereas node 11 of the same cluster does not satisfy the criteria since its neighbor 2 has a neighbor (node 10) outside its immediate geometric neighbors (nodes 1 and 3) on the circle. Thus, moving node 11 inside the circle would potentially result in node 11 overlapping with edge $\{2, 10\}$. The heuristic tries high degree in-nodes satisfying these criteria as long as the user-specified maximum number of such inner nodes is not reached. This option is expected to be defined as a percentage of the total number of nodes in a cluster.

Here, is the algorithm to calculate such inner nodes.

algorithm FINDINNERNODE(Graph G, Circle C)
1) $L := \{u \in C \mid d_{G[C]}(u) \geq 2 \wedge u$ is an in-node in $G\}$
2) sort nodes in L in non-ascending order using their degrees in $G[C]$
3) **for** each $u \in L$ **do**
4) $S :=$ nodes of minimum length circular segment spanning $N(u) \cup \{u\}$
5) $isCandidate := true$
6) **for** each $v \in N(u)$ **do**
7) **for** each $w \in N(v)$ **do**
8) **if** $(w \in S) \wedge (w \neq u) \wedge$ (v and w are not geometric neighbors on S) **then**
9) $isCandidate := false$
10) **if** $isCandidate$ **then**
11) **return** u
12) **return null**

If this option is enabled and some node is moved inside the associated circle, it should be treated the same way unclustered nodes are, for the remainder of layout. In other words, spring and repulsion forces determine the final positions of such nodes.

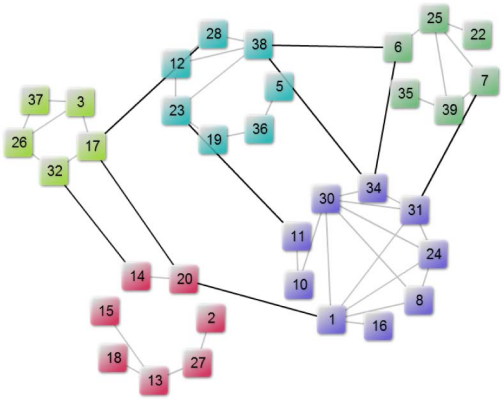


Fig. 13. A randomly generated graph laid out by our algorithm. ($n = 40$, $m/n = 1.2$, $m_{ic}/m = 0.20$, $d_{max} = 10$, and $d_{min} = 2$).

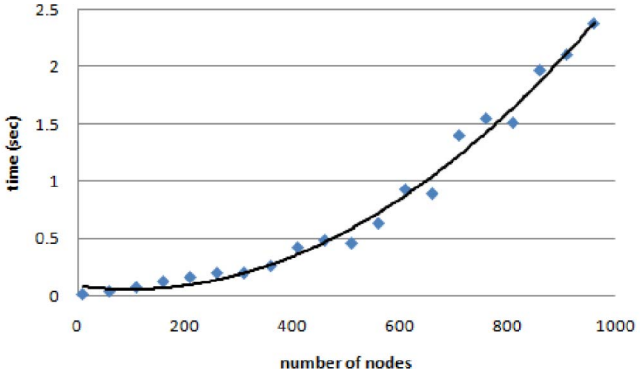


Fig. 14. Number of nodes (n) versus execution time of our algorithm, fitting a quadratic polynomial trendline ($m/n = 1.5$, $m_{ic}/m = 0.10$, $d_{max} = 15$, and $d_{min} = 2$).

3.6 Execution Time

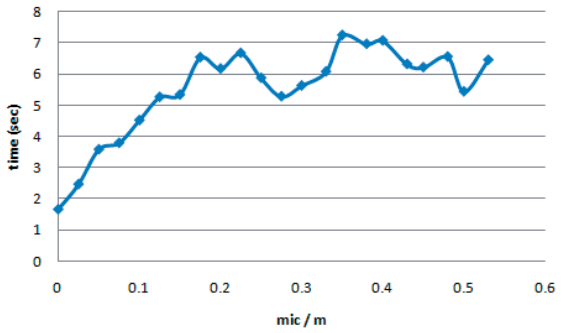
The main body of the algorithm simply calls the five steps described earlier after proper initialization. A quick analysis reveals that the overall running time of the layout of a clustered graph is $O(k \cdot [|E(G)| + |\overline{\mathbf{V}}_{\mathbf{o}}(\mathbf{G})|^2 + |\mathbf{V}_{\mathbf{o}}(\mathbf{G})| \cdot \lg |\mathbf{V}_{\mathbf{o}}(\mathbf{G})|])$, where k is the number of iterations required to reach a minimal energy state. Notice that in the worst case this expression is quadratic in the total number of nodes of the graph.

4 Implementation

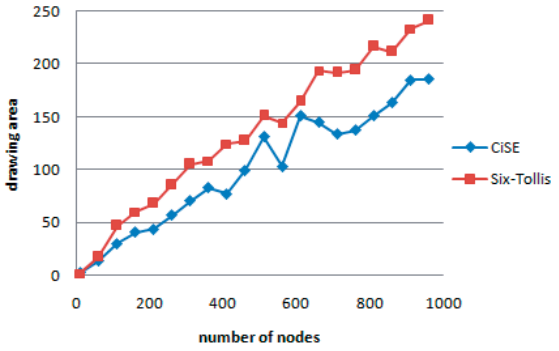
We developed and tested the proposed layout algorithm within version 2.0 of CHISIO, an open-source generic graph visualization tool. The algorithm in [10] is used for the layout of the individual clusters. The development environment was Sun's Java SDK 1.5 and Microsoft's Windows XP operating system on an ordinary 32-bit personal computer (Pentium D 2.8 GHz CPU and 3 GB memory).

Whenever the provided data were not clustered, the algorithm described in [20] to find community structures in networks was used to obtain one. In addition, for practical purposes (i.e., for ease of implementation), we made use of linear global alignment (the one described in [21]) to *emulate* circular alignment by duplicating the contents of the string x, and ignoring gaps at the beginning and at the end of the alignment. In addition, the option for placement of in-nodes inside associated circles was disabled.

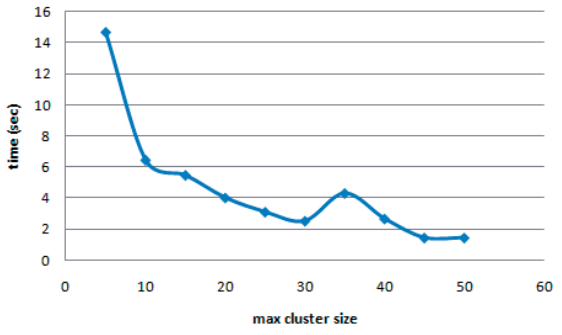
We performed experiments on randomly generated synthetic graphs with one of several parameters changing



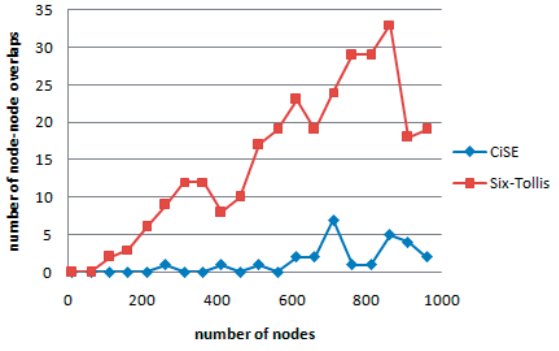
(a) Inter-cluster edge ratio (m_{ic}/m) vs. execution time of our algorithm ($n = 500$, $m/n = 1.5$, $d_{max} = 15$ and $d_{min} = 2$)



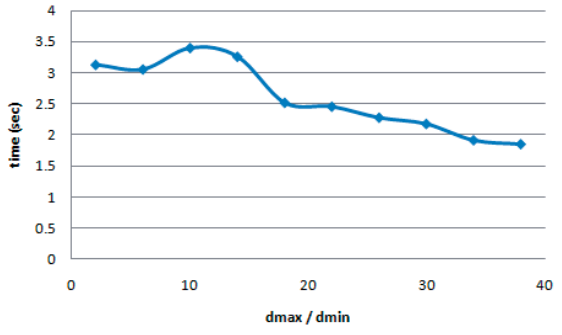
(b) Number of nodes (n) vs. drawing area of CiSE and circular layout of [4] ($m/n = 1.5$, $m_{ic}/m = 0.10$, $d_{max} = 15$ and $d_{min} = 2$)



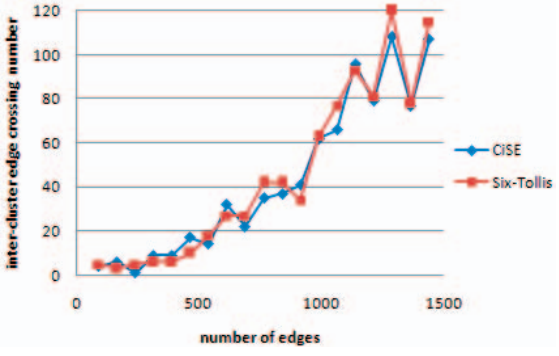
(c) Maximum cluster size (d_{max}) vs. execution time of our algorithm ($n = 500$, $m/n = 1.5$, $d_{min} = 2$ and $m_{ic}/m = 0.10$)



(d) Number of nodes (n) vs. number of node-node overlaps produced by CiSE and circular layout of [4] ($m/n = 1.5$, $m_{ic}/m = 0.10$, $d_{max} = 15$ and $d_{min} = 2$)



(e) 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 $m_{ic}/m = 0.10$)



(f) Number of edges (m) vs. number of inter-cluster edge crossings of CiSE and circular layout of [4] ($m/n = 1.5$, $m_{ic}/m = 0.10$, $d_{max} = 15$ and $d_{min} = 2$)

Fig. 15. Experimental results on running time performance (left) and quality (right).

for each set. For each test, a random graph was generated with the provided parameters:

- n: desired total number of nodes,
- m/n: desired proportion of edges to number of nodes,
- m_{ic}/m: desired proportion of intercluster edges to number of all edges,
- $[d_{min}, d_{max}]$: cluster size range.

Uniformly drawing a random graph from the set of all clustered graph is not easy if not impossible; we generate our random clustered graphs as follows: First, nodes are created and distributed to clusters, respecting minimum and maximum cluster sizes. One distinguished cluster holds the set of unclustered nodes. Then, we create intercluster edges, respecting the ratio m_{ic}/m, leaving the remaining count for intracluster edges. Finally, we remove any isolated nodes. Notice here that some of the input parameters may not be fully satisfied. Each test is executed 10 times and the average is taken. Fig. 13 shows an example of a randomly generated clustered graph.

The results were found to be quite satisfactory, as far as the general graph drawing criteria, such as the number of crossings and the total area are concerned. Furthermore, the experimental executions were found to be not only reasonably fast for interactive use but also in line with the earlier theoretical analysis, as detailed below. A supplemental document, available online, contains sample drawings of, mostly real life, relational data laid out by CiSE.

4.1 Running Time Performance

From the theoretical analysis given earlier, a quadratic behavior of execution time is expected. The experiments validate this argument (Fig. 14). Also note that our algorithm has the same asymptotic running time complexity as previous algorithms in [3] and [4]. Even though

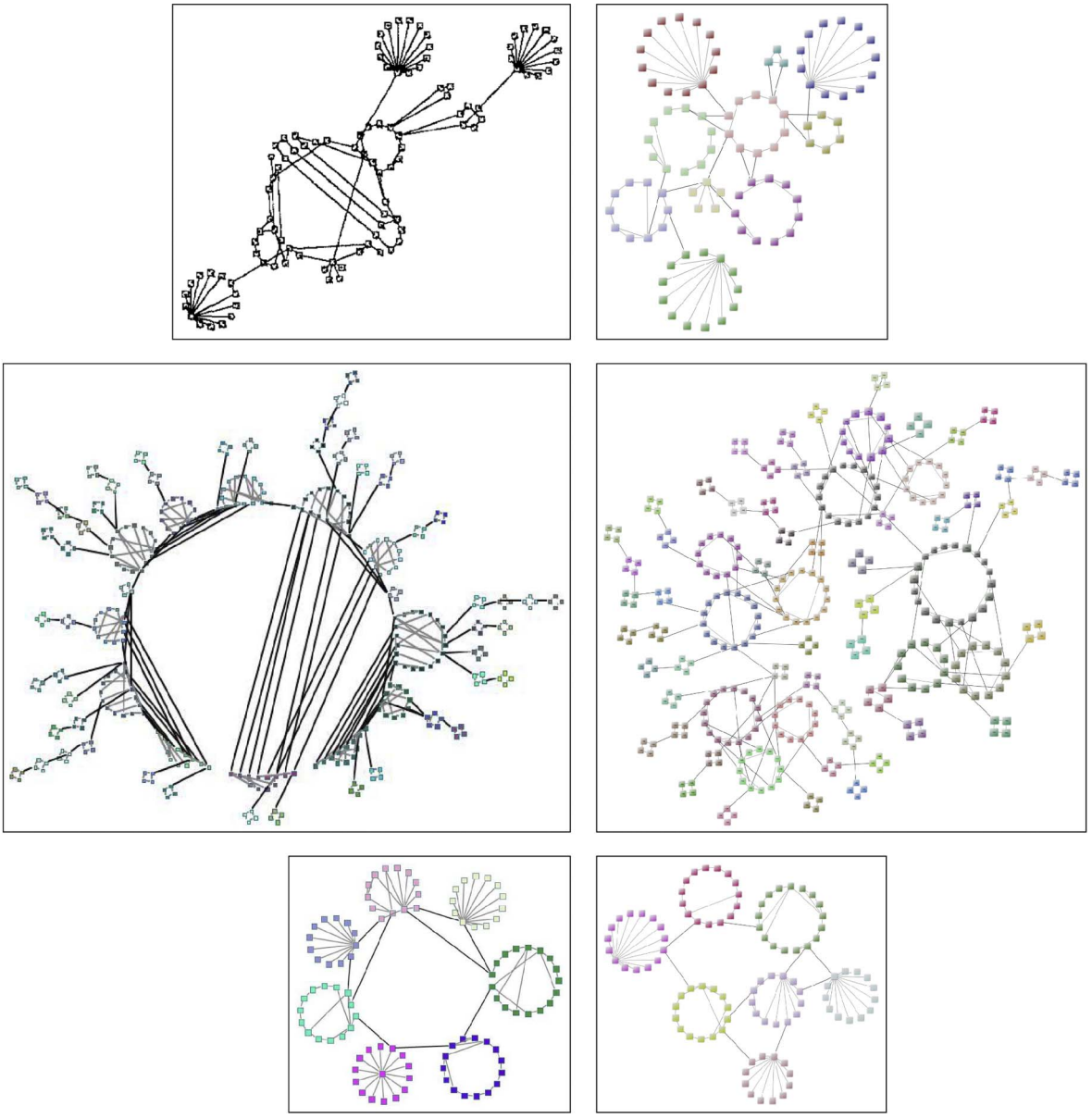


Fig. 16. Example layouts of the same graph by circular layout of GLT (left) and CiSE (right).

heuristics employed by CiSE, in addition to the basic spring embedder, results in CiSE executing slower in practice (see the supplemental document for details, available online), we think being able to lay out graphs of up to a thousand elements within a couple of seconds qualifies CiSE for use in an interactive tool. For larger graphs, layout is rarely the bottleneck and effective analysis requires good complexity management techniques [22].

We also performed a test set to see how the proportion of intercluster edges to all edges affects the execution time (Fig. 15a). The running time seems to depend on this ratio for low values, due to the fact that the layout of the quotient graph converges very quickly when there are few inter-cluster edges. However, for larger values of the ratio, this behavior is not observed, as the execution times do not exhibit a correlation with the ratio.

We also conducted an experiment on the effect of cluster sizes on execution time (Fig. 15c). As clusters of a graph get bigger, the corresponding quotient graph gets smaller, resulting in faster layout of the quotient graph. The layout of individual clusters [10] takes longer due to the increasing size of the clusters; however, this slowdown is only linear in the number of the nodes in the cluster. As a result, an increase in the cluster size results in a decrease in the overall running time.

We also looked into the effect of the uniformity of cluster sizes on execution time. As can be seen from the resulting plot (Fig. 15e), nonuniformity has a positive effect on execution time. Differences between cluster sizes help the cluster nodes move more freely during the quotient-graph layout. In other words, smaller clusters can move more easily around bigger ones, yielding faster convergence, as more edges can relax in each iteration.

4.2 Quality

In our experiments, we also inspected the quality of the layouts produced by CiSE. We used the criteria of clear cluster separation on top of commonly accepted aesthetic criteria such as edge crossings and area [23]. In general, the results

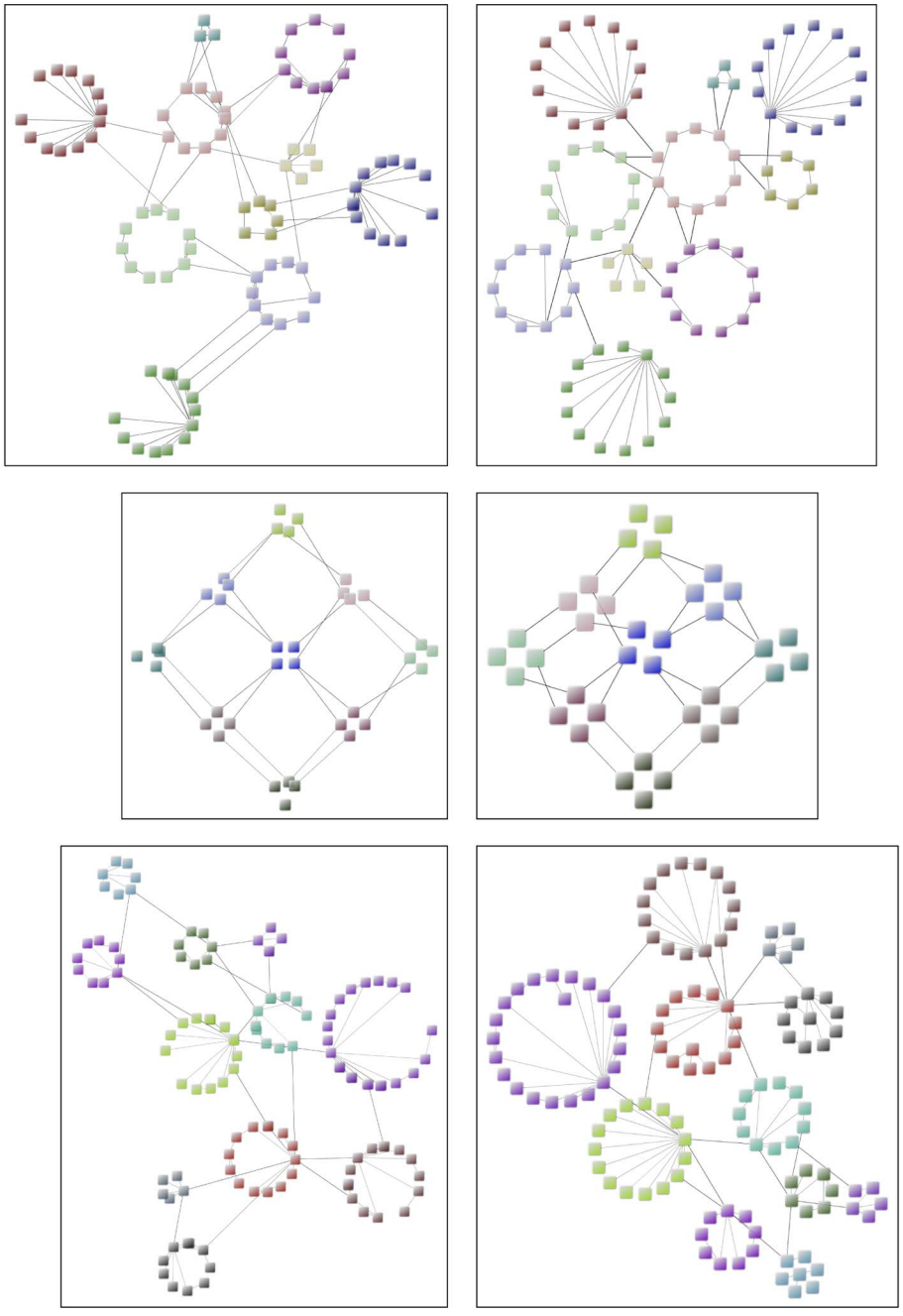


Fig. 17. Example layouts of the same graph by circular layout of [4] (left) and CiSE (right).

produced by the algorithm are satisfactory. With the help of repulsion forces, nodes almost never overlap. On the other hand, they stay sufficiently close to each order, resulting in compact drawings. The drawing area is uniformly occupied by clusters, preserving the symmetry of the visualization.

It is difficult, however, to state that edge lengths in drawings produced by our algorithm are uniform. This is due to two main reasons:

- Intracluster edges usually have varying lengths because of the circular positioning of nodes. Since minimizing edge crossings is of highest priority in individual cluster layout and nodes are placed at a fixed distance from each other, the edge lengths will inevitably vary according to the order of the nodes around the circle. Optional movement of on-circle nodes to inside associated clusters helps with this problem.
- Intercluster edges might sometimes be arbitrarily long because of unachievable swaps. Swaps are requested as a result of opposite spring forces caused by long incident edges acting on two on-circle nodes. Since we try to avoid edge crossings at any cost, we never swap two such nodes if the swap would augment the edge crossing count.

We also verified the use of our novel heuristics in regards to quality improvement. For instance, rotation of clusters help reduction of edge-crossing number around 35 percent on the average, whereas local swaps reduce edge-crossing number by nearly 10 percent on the average. We also tested how improvements of rotations stand as the graphs get denser. They seem to contribute significantly even for graphs with

densities as high as $m/n = 1.5$. Details may be found in the supplemental document, available online.

To better evaluate the layout quality, we compared our algorithm with the circular layout algorithm in graph layout toolkit (GLT) [3]. Since the details of this algorithm implemented as part of a commercial tool are not available, we ended up comparing our algorithm with theirs for only some of the very few graphs provided in their paper and publicly available documentation. As apparent from Fig. 16, cyclic parts of clustered graphs end up on a single large backbone circle, inevitably introducing very long intercluster edges with many crossings. Details of the comparison is available in the supplemental document, available online.

The algorithm in [4], on the other hand, is able to better handle the quotient graph layout using a force-directed method but exhibits many poor layout characteristics, such as unnecessarily long and nonuniform edge lengths, nonuniform distribution of nodes of a cluster around associated circle, and most importantly node-node overlaps, as discussed earlier. Examples in Fig. 17 contrast the algorithm in [4] with ours. Further examples and details of the comparison are available in the supplemental document, available online. Random experiments performed to contrast the two, in regards to layout quality, support our theoretical findings as well (Figs. 15b, 15d, and 15f).

4.3 Availability

A web demo of our algorithm may be accessed at

http://www.cs.bilkent.edu.tr/~ivis/cise.html.

In addition, an implementation of CiSE can be found within CHISIO 2.0:

http://www.cs.bilkent.edu.tr/~ivis/chisio.html.

The Java sources of CHISIO, including the sources for CiSE, are also available through a SourceForge project.

5 Conclusion

We presented a novel algorithm for the circular layout of clustered graphs. To our knowledge, this is the first natural extension to the basic spring embedder to nicely handle clustering structure of arbitrary graphs. The main novelties of our work include the use of a modified spring embedder system that treats clusters as part of the physical system and optional use of the space inside each circle to reduce total drawing area. Needless to say that our algorithm inherits the disadvantages of the force-directed approach such as heavy use of computational resources in addition to its nice properties such as uniform vertex distribution, uniform edge lengths, and symmetry. Experimental results were found satisfactory both in terms of quality and computational efficiency, surpassing previous algorithms in almost all aspects. In the future, we plan to work on improving the layout of individual clusters by placing high degree nodes inside the circle as part of the individual cluster layout as opposed to a postprocessing step.

Acknowledgments

This work was supported in part by TUBITAK (The Scientific and Technological Research Council of Turkey) under Grant No. 111E036. The authors would also like to thank the anonymous referees for their extremely helpful comments and suggestions.

References

[1] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, "Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society," *Nature,* vol. 435, pp. 814-818, 2005.

[2] U. Dogrusoz, Q. Feng, B. Madden, M. Doorley, and A. Frick, "Graph Visualization Toolkits," *IEEE Computer Graphics and Applications,* vol. 22, no. 1, pp. 30-37, Jan./Feb. 2002.

[3] U. Dogrusoz, B. Madden, and P. Madden, "Circular Layout in the Graph Layout Toolkit," *Proc. Symp. Graph Drawing (GD '96),* S. North ed., pp. 92-100, 1997.

[4] J.M. Six and I.G. Tollis, "A Framework for User-Grouped Circular Drawings," *Proc. 11th Symp. Graph Drawing (GD '03),* G. Liotta, ed., pp. 135-146, 2004.

[5] Q. Feng, R. Cohen, and P. Eades, "Planarity for Clustered Graphs," *Proc. Third Ann. European Symp. Algorithms (ESA '95),* P. Spirakis, ed., pp. 213-226, 1995.

[6] K. Sugiyama and K. Misue, "Visualization of Structural Information: Automatic Drawing of Compound Digraphs," *IEEE Trans. Systems, Man and Cybernetics,* vol. 21, no. 4, pp. 876-892, July/Aug. 1991.

[7] X. Wang and I. Miyamoto, "Generating Customized Layouts," *Proc. Second Int'l Symp. Graph Drawing (Proc. GD '95),* F. Brandenburg, ed., pp. 504-515, 1995.

[8] *Drawing Graphs: Methods and Models,* M. Kaufmann, and D. Wagner eds. Springer, 2001.

[9] J.M. Six and I.G. Tollis, "A Framework and Algorithms for Circular Drawings of Graphs," *J. Discrete Algorithms,* vol. 4, no. 1, pp. 25-50, 2006.

[10] H. He and O. Skora, "New Circular Drawing Algorithms," *Proc. Workshop Information Technologies - Applications and Theory (ITAT '04),* 2004.

[11] M. Baur and U. Brandes, "Crossing Reduction in Circular Layouts," *Proc. 30th Int'l Workshop Graph-Theoretic Concepts in Computer-Science (WG '04),* pp. 332-343, 2004.

[12] E.R. Gansner and Y. Koren, "Improved Circular Layouts," *Proc. 14th Int'l Conf. Graph Drawing,* pp. 386-398, 2006.

[13] M. Baur and U. Brandes, "Multi-Circular Layout of Micro/Macro Graphs," *Proc. 15th Int'l Conf. Graph Drawing,* pp. 255-267, 2007.

[14] M. Kaufmann and R. Wiese, "Maintaining the Mental Map for Circular Drawings," *Proc. 10th Int'l Symp. Graph Drawing,* pp. 12-22, 2002.

[15] U. Brandes, "Drawing on Physical Analogies," *Drawing Graphs: Methods and Models,* M. Kaufmann and D. Wagner, eds., pp. 71-86, Springer,, 2001.

[16] U. Dogrusoz, E. Giral, A. Cetintas, A. Civril, and E. Demir, "A Layout Algorithm for Undirected Compound Graphs," *Information Sciences,* vol. 179, pp. 980-994, 2009.

[17] D. Harel and Y. Koren, "Drawing Graphs with Non-Uniform Vertices," *Proc. Working Conf. Advanced Visual Interfaces (Proc. AVI '02),* pp. 157-166, 2002.

[18] T.M.J. Fruchterman and E.M. Reingold, "Graph Drawing by Force-Directed Placement," *Software Practice and Experience,* vol. 21, no. 11, pp. 1129-1164, 1991.

[19] A. Mosig, I.L. Hofacker, and P.F. Stadler, "Comparative Analysis of Cyclic Sequences: Viroids and Other Small Circular Rnas," *Proc. German Conf. Bioinformatics,* pp. 93-102, 2006.

[20] M.E.J. Newman and M. Girvan, "Finding and Evaluating Community Structure in Networks," *Physical Rev.,* vol. E 69, no. 026113, 2004.

[21] S.B. Needleman and C.D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins," *J. Molecular Biology,* vol. 48, no. 3, pp. 443-453, 1970.

[22] U. Dogrusoz and B. Genc, "A Multi-Graph Approach to Complexity Management in Interactive Graph Visualization," *Computers & Graphics,* vol. 30, no. 1, pp. 86-97, 2006.

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

Ugur Dogrusoz received the PhD degree from the Computer Science Department of Rensselaer Polytechnic Institute, Troy, New York. He is an associate professor of computer engineering at Bilkent University, Ankara, Turkey. His research interests are at the intersection of information visualization, bioinformatics, and graph algorithms. He co-founded the Bilkent Center for Bioinformatics in 2002 and directed the center until 2010. He is also the recipient of the National Young Scientist Career Development Award from TUBITAK. He was the vice president of Engineering as well as a researcher and developer at Tom Sawyer Software (Berkeley, CA) for a number of years before joining Bilkent. He is a senior member of the IEEE.

Mehmet E. Belviranli received the BS and MSc degrees in computer engineering from Bilkent University, and is currently working toward the PhD degree in the Computer Science and Engineering Department of the University of California, Riverside.

Alptug Dilek received the BS and MSc degrees in computer engineering from Bilkent University. He is a software engineer working in the Software and Data Engineering Division of TUBITAK.