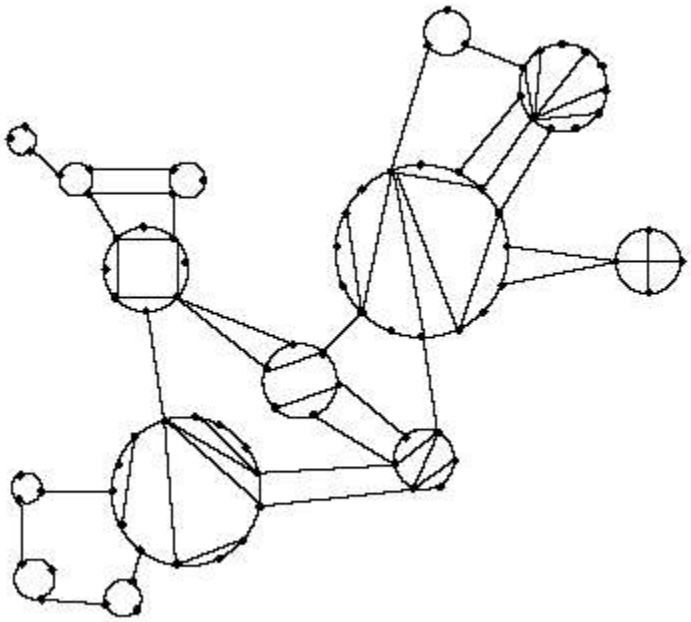
HY483

Αλγόριθμοι γράφων

Βιβλιογραφική εργασία

Ανδρέου Δημήτρης

14/12/2005



1. A Fast Adaptive Layout Algorithm for Undirected Graphs (1994)

1.1 Introduction

There are many techniques that are used in the context of producing aesthetically pleasing graph drawings. Under the assumption that edges are drawn as straight lines, the problem is transformed to the problem of positioning the nodes of a graph. A well-known approach for creating such a positioning, in a general graph, is the spring-embedder model. By using forces between the elements of a graph, we allow graph layout to evolve naturally.

Some of the most important aesthetic criteria include displaying symmetries existing in the graph and avoiding as much as possible of edge crossings. Furthermore, edges should have as few bends as possible (while straight-line edges completely by-pass this problem), and their length should not deviate greatly. The drawing should occupy sensibly small space, while nodes should be evenly distributed in the area.

It is known that the optimal solution to this criteria is involves NP-complete problems (for instance, edge crossing minimization is shown to be NP-complete). So, only good approximations to optimal solutions are generally sought.

1.2 Previous work

The basic spring-embedder model is due to Eades. It treats nodes as mutually repulsive charges and edges as springs connecting the nodes. The nodes are positioned randomly on the plane, and at each iteration the system calcucates for each node the forces that are applied to each, and moves the nodes accordingly. A problem is raised, as to when to stop the iteration. Eades' approach was simply to run a fixed number of iterations, and then stop. This is problematic because the graph may not have converged, or it may have converged long ago, wasting time.

Kamada and Kawai refined the model, firstly by introducing the notion of optimal edge length, and updating only a single node at each step. This algorithm converges deterministically to a local minimum.

To overcome the problem of ending up in a local minimum, randomness is introduced. *Simulated annealing* is used, allowing for positive changes in energy state. Any move that decreases energy is of course accepted, but additionally, moves to the opposite direction may sometimes be accepted, by a probability that is dependend on the cu *temperature*. The temperature is high when the system starts, so at that phase the system has great flexibility in its movement. Then, the system is "cooled-down", and gradually the probability of a move that increases energy approaches zero. Unfortunately, this technique is quite slow. Fruchterman and Reingold improved Eades' algorithm by using a simple cooling schedule. In their algorithm, the distance that a node can travel at each single step is depended on the current temperature.

Finally, there have been proposed methods of graph preprocessing, in order to get a good initial placement, and hopefully, a better final local minimum.

1.3 The GEM (Graph EMbedder) algorithm outline

The algorithm proposed is a variation of the spring-embedder model. Innovations (at that time) included: attraction of nodes towards their barycenter, the concept of a local, per node, temperature, and the detection of oscillations and rotations.

A major goal is to achieve high convergence speed, so that drawings can be produced interactively by a user. To this end, integer arithmetic is utilized.

The authors observe that cooling schedules give better results than methods based only on a gradient descent, but they are slower than the latter. So, their intention is to improve the running time of their based-on-temperature-model method. In GEM, temperature indicates the maximum distance a node is allowed to travel, and it directly affects the proper values for other constant parameters, i.e. attractive and repulsive forces.

In contrast to other proposals, GEM does not follow a cooling schedule in the strict sense, but adapts locally to the data, and instead of a global temperature, a *local* temperature is introduced. This temperature depends on its old temperature and the possibility that the node takes part in an oscillation or a subgraph rotation. It can be raised if the algorithm determines that the node is probably not close to its final destination. Thus, the global temperature is now defined as the average of all local temperatures of the nodes.

The algorithm comprises of an initialization phase and an iteration phase. At initialization, an initial position is assigned, along with impulse and temperature for each node. Iteration updates node positions and temperatures, until global temperature is lower than a desired minimum.

At each iteration, every node is updated once, in random order. A node's position is updated according to attractive forces of its edges, and repulsive forces to the other nodes. Additionally, a gravitational force is exerted, pushing the node towards the barycenter of the graph. This helps keeping disconnected subgraphs together. The resultant force is scaled with the node's current temperature. Due to the fact that local temperature affects global energy distribution, GEM has the ability to escape from local mimima, and this presents a difficulty in the effort to provide proofs of convergence.

1.4 Details of GEM algorithm

Each node remembers its current position j, its last impulse p, its (local) temperature T and a skew gauge d. At initialization, $p = 0$, $d = 0$ and $T = T_{init}$. The initial positioning of nodes does not affect much the resulting drawings, as proposed by Kamada/Kawai. So, a random positioning is utilized in the general case, although for special graphs such as trees and meshes, inserting the nodes one by one can accelerate convergence.

At each iteration, each node is updated by the following algorithm:

-- *Input:*
-- v *vertex to be updated*
-- $\mathbf{c}$ $\sum_u u.\boldsymbol{\xi}$*; the barycenter of* G *is computed as* $\mathbf{c}/|V|$
-- Φ *function growing with* $\deg(v)[1 + \deg(v)/2]$
-- *Output:*
-- $\mathbf{p}$ *current impulse of* v
-- *Constants and Functions:*
-- E_{des} *desired edge length* $[128]$
-- γ *gravitational constant* $[1/16]$

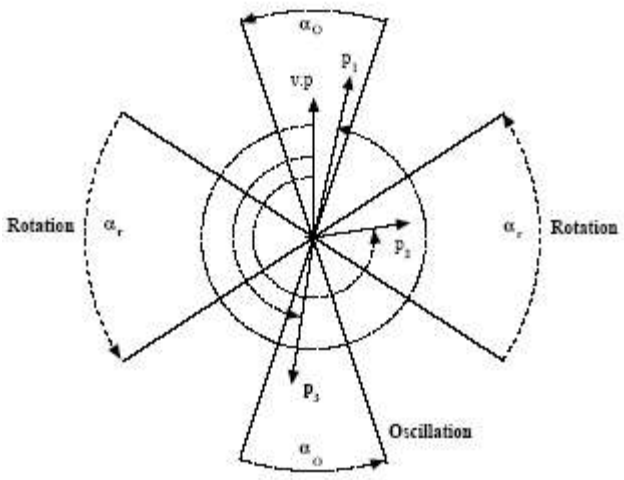
-- *attraction to center of gravity*
$\mathbf{p}$:=$(\mathbf{c}/|V| - \mathbf{v.pos}) \cdot \gamma \cdot \Phi(v)$;
-- *random disturbance*
$\boldsymbol{\delta}$:=*small random vector;* -- *default range:* $[-32, 32] \times [-32, 32]$
$\mathbf{p}$:=$\mathbf{p} + \boldsymbol{\delta}$;
forall $u \in V \setminus \{v\}$ **do**
    -- *repulsive forces*
    $\boldsymbol{\Delta}$:=$v.\boldsymbol{\xi} - u.\boldsymbol{\xi}$;
    if $\boldsymbol{\Delta} \neq \mathbf{0}$ **then** $\mathbf{p}$:=$\mathbf{p} + \boldsymbol{\Delta} \cdot E_{\text{des}}^2/|\boldsymbol{\Delta}|^2$;
forall $(u, v) \in E$ **do**
    -- *attractive forces*
    $\boldsymbol{\Delta}$:=$v.\boldsymbol{\xi} - u.\boldsymbol{\xi}$;
    $\mathbf{p}$:=$\mathbf{p} - \boldsymbol{\Delta} \cdot |\boldsymbol{\Delta}|^2/(E_{\text{des}}^2 \cdot \Phi(v))$;

Φ is a function that scales impulse conversely to the node's degree. This provides higher *inertia* for nodes with many edges, which accelerate convergence of the drawing, since positional changes to nodes with high degree propagate to more nodes, causing ripple-like updates.

At each step, a modification of the current temperature is needed. This modification is dependent on the last temperature, the last and current movement of the node, and the skew gauge, which indicates the likeliness that the node is oscillating or is part of a rotation. To detect these phenomena

At every stage of the algorithm execution, each node remembers its last impulse. Using this information, one can diagnose that a node is part of a subgraph rotation, or is oscillates, by means of trigonometry. Given the vector of the last repulse, it is easy to see if the vector of the new impulse is on the same direction, is about vertical, or, finally, opposite. These conditions are approximated using an opening angle, which must be chosen carefully.

If the new impulse is on the same direction as the old one, the local temperature is raised, as it is assumed that the node moves at the "right" direction, and higher temperature accelerates node's next movement. If the node participates in a rotation, skew gauge increases, and as it approaches 1, the more unbalanced the node is. This scales down local temperature (and, thus, future movement). Finally, if the node oscillates, GEM assumes that the node just passed its optimal position and will continue to oscillate in the next rounds, so it scales down the temperature. This has the effect of freezing the node.



1.4 Measurements and conclusions

GEM has been benchmarked in comparison to Kamada/Kawai (KK) and Fruchterman/Reingold (FR) algorithm. GEM was expected to outperform the other two implementations, on the account that only integer arithmetic is used, but what remained to be seen was whether the quality of the drawings deteriorates. The results are quite convincing – the algorithm produces nice drawings for many graphs of various type and size, and many times it succeeds in finding a planar embedding, if such one exists.

The GEM algorithm's local adaptation is quite interesting. With slightly more memory (one vector per node), it can detect the type of moves, and scale them according to that type. This is reasonable to accelerate running time, as nodes approach faster to the position that they are heading to, or, if they rotate or oscillate, they slow down till they freeze in place. This technique has the additional benefit that the algorithm does not depend heavily on spring constants; if edge attraction is low, it gets accelerated, if it is too high, the nodes oscillate and the moves scale down as well.

2. A Framework for User-Grouped Circular Drawings (2003)

2.1 Introduction

Circular drawings are known to be well-adept in demonstrating group/child relations in a graph. When there is the need to show the strong relevance of a subset of nodes, it is a viable alternative to draw this subset on the circumference of a circle. This way, the viewer understands that these nodes have further structural meaning, and have to be understood together, as a group.

Such drawings have wide applications in the fields of telecommunication, computer networks, social network analysis, project management, and more.

There have been several approaches that partition the graph into groups, and then place the groups onto embedding circles. What lacks from most of those techniques was the ability for the user to provide the initial partitioning, and thus, take control of the final drawing and the message that it attempts to convey. Graph Layout Toolkit, by Tom Sawyer software is an exception, but it still places user selections themselves on a single circle.

2.2 Review of previous algorithms

Authors' own previous work has yielded an linear-time algorithm that layout biconnected and non-biconnected graphs on multiple circles, with few edge crossings. It is noted that the problem of minimizing edge crossings is proven to be NP-Complete, even when the nodes are restricted to appear on a circle.

2.2.1 Circular Drawings of Biconnected Graph (CIRCULAR)

The biconnected-case algorithm produces draws with fewer crossings than previous techniques, by its tendency of placing edges toward the outside of the embedding circle, and of placing nodes near their neighbors.

A wave-like node visit is employed, seeking for pair edges (edges incident to neighbor nodes) which are removed. A depth-first search (DFS) is performed on the reduced graph. This procedure yields a longest path, which is then placed continuously on the embedding circle, while the rest of the nodes are merged into the ordering.

The time-complexity of this algorithm is O(m), where m is the number of edges. A very important property of this algorithm is that if it is applied to an outer-planar graph, it will find produce a node placement with zero edge crossings.

2.2.2 Circular Drawings of Non-Biconnected Graph on a Single Circle (CIRCULAR-Nonbiconnected)

The non-biconnected-case algorithm is composed of the following steps: at first, the graph is decomposed to its block-cutpoint tree. Then, the resulting component array is layout on a circle. This can be achieved by a simple DFS, without any edge crossings. Finally the components themselves are layout using a variant of CIRCULAR. It is noted

that explicit treatment is needed for the issue of placement of articulation points. Another issue worth mentioning is the strategy by which to place the biconnected components on an arc of the circle.

The final drawing has the property that nodes that belong in a single graph, appear consecutively on the circle. The time-complexity of this algorithm is O(m).

2.2.3 Circular Drawings of Non-Biconnected Graph on Multiple Circles (CIRCULAR-withRadial)

This algorithm firstly decomposes the graph into its biconnected components, as CIRCULAR-Nonbiconnected did. One circle is assigned for each component. Then the circles are laid out using a radial layout technique. The time complexity of this algorithm is again O(m).

2.3 A Framework for User-Grouped Circular Drawing

The dominating difference of a circular drawing grouped by biconnectivity and a user-grouped circular drawing is that in the latter, no structural information can be inferred about the subgraphs. So, an algorithm that handles user selections must be general enough to be able to apply to arbitrary graph groupings, either biconnected or not. The goals of the proposed technique are: 1) Highly-visible user groupings, 2) Low edge crossings in groups, 3) Low inter-group edge crossings and 4) Fast layout.

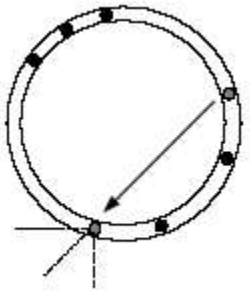
An important assumption being made is that inter-group relations are not very complex. The algorithm begins as follows: Define supergraph $G_s = (V, E, P)$, where P is the user-defined node partitions. For each inter-partition edge, define an edge connecting the respective partitions. This structure is then laid out with a force-directed technique. Based on the assumption that this structure will not be very complex, the force-directed part will finish quickly.

The partitions can be either biconnected or not. The respective algorithms that were briefly mentioned are used to draw the partition in either case. Till here, goals 1) and 2) have been reached, as these algorithms are shown to perform well at practice.

Next, inter-group edge crossings must somehow be reduced. A force-directed approach is applied, that is different to the traditional algorithms in that the nodes must appear on the circumference of their circles. Yet, the ordering of the nodes on the circles is allowed to change, thus nodes are highly encourages to move close to their incident groups. This, unfortunately, is working against the good drawings (with few edge crossings) produces at the previous step. Hopefully, the intra-group drawings will not be affected much, due to the simplicity of the inter-group relations. In either case, the reduction of inter-group edge crosses is more important than intra-group edge crossings, since the former produce more visual clutter, and the algorithm is willing to pay the relatively smaller price of deteriorating the inner layout of circles, to achieve better overall outcome.

2.4 Circular-Track Force-Directed

As mentioned previously, a new force-directed algorithm is needed to layout the nodes of the circles. The requirement is that nodes remain on their circles at all time, while they are able to jump over each other, as depicted in the following diagram:



As nodes' location is restricted by the circle they reside in, they can be located by simply using a theta coordinate, which denotes the angle on the circumference of the circle where the node is.

$$x_i = x_\alpha + r_\alpha * cos(\theta_i)$$
$$y_i = y_\alpha + r_\alpha * sin(\theta_i)$$

(x_a, y_a) is the center of the circle.
Hooke's law provides for the potential energy V in the system:

$$V = \sum_{ij} k_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2]$$

These equations can be combined to produce:

$$V = \sum_{(i,j)\in E} k_{ij}\ [((x_\alpha + r_\alpha * cos(\theta_i)) - (x_\beta + r_\beta * cos(\theta_j)))^2 + ((y_\alpha + r_\alpha * sin(\theta_i)) - (y_\beta + r_\beta * sin(\theta_j)))^2]$$

Furthermore, a repulsive force between the nodes is added, to avoid nodes occlusion:

$$\rho_{ij} = [(x_\alpha + r_\alpha * cos(\theta_i) - x_\beta - r_\beta * cos(\theta_j))^2 + (y_\alpha + r_\alpha * sin(\theta_i) - y_\beta - r_\beta * sin(\theta_j))^2]$$

$$V = \sum_{(i,j)\in E} k_{ij}\rho_{ij} + \sum_{(i,j)\in V\times V} g_{ij}\frac{1}{\rho_{ij}}$$

Finally, the force applied to each node is defined as:

$$F_i = \frac{V(\theta_i + \epsilon, \theta_j) - V(\theta_i - \epsilon, \theta_j)}{2\epsilon}$$

2.5 Algorithm CIRCULAR-withFORCES

The full algorithm is summarized here:

Algorithm 1 *CIRCULARwithFORCES*

Input: A graph $G = (V, E, P)$.
Output: User-grouped circular drawing of G Γ.

1. Determine superstructure G_s of G.
2. Layout G_s with a basic force-directed technique.
3. For each group p_i in P
   (a) If the subgraph induced by p_i, G_i, is biconnected layout G_i with *CIRCULAR*.
   (b) Else layout G_i with *CIRCULAR-Nonbiconnected*.
4. Place layout of each group p_i at the respective location found in Step 2.
5. For each group p_i
   (a) rotate layout circle and keep position which has lowest local potential energy.
   (b) reverse order of nodes around embedding circle and repeat Step 5a.
   (c) if result of Step 5a had a lower local potential energy than that of Step 5b revert to result of Step 5a.
6. Apply force-directed technique using equations of Section 4.

Steps 5 and 6 need some clarification. At step 5, we have an almost final drawing: the user-defined components have been laid out by the CIRCULAR algorithms, and the super-structure has been laid out as well using a basic force-directed scheme. At step 5, prepares the graph for the final step, by reducing locally the potential energy. It is meant as a simple heuristic that may help the final step perform better. The algorithm doesn't try to find the best combination of rotations, but treats the groups sequentially, in one iteration, and each circle independently. The final step applies the force-directed algorithm outlined in the previous section.

Worse-case time complexity is unknown, as per the use of force-directed algorithms, but authors claim that in practice it is expected to be $O(n^2)$. Evidence are provided that almost final drawings do not take much time to converge using force-directed techniques, so the final step (No. 6) can be regarded to spend $O(n^2)$ as well.

2.6 Conclusions

This work outlines a framework of algorithms useful to draw user-selected sub-graphs. The framework achieves a nice balance between elements that work against each other. That is, few inter-group edge crossings, few intra-group edge crossings and high visibility of groups. It does it by combining a set of algorithms, including basic and circular-track

force-directed methods, and circular drawing algorithms of biconnected and non-biconnected components.

The authors' work is based on the assumption that inter-group relations will be relatively small. Yet, that may not hold on every problem instance, as user selections are arbitrary. Also, the method described implies that the user partitions the whole graph, while it seems sensible that the user only cares about a set of nodes, and chooses them. In that case, the rest of the graph will end up in a circle of its own, and it is quite probable for these groups to have many incident edges.

Another point that merits attention is the use of angle parameters in the circular-track force-directed algorithm. The outcome of this algorithm is not reformed into a simple ordering of nodes, thus not balancing the nodes uniformly on the circumference of the circle. This could cause visual problems if the nodes of a tree are highly biased toward a specific direction; that group could be misunderstood for another shape. Yet, spreading uniformly the node on the circle could in turn cause more edge crossings, so the right path to this matter is unclear.