ALGORITHMS FOR VLSI
SPECTRAL GRAPH DRAWING

Ali Arabyarmohammadi January 2022

1 Introduction

1.1 Graph Drawing

Graph theory is used in many fields of studies and because of the wide range of its uses, each discovery in this field can have a significant societal influence. A graph drawing, also known as a network diagram, is a visual representation of the vertices and edges of a graph. We know that the same graph can have several alternative layouts. All that concerns is which vertices are connected by edges. The way these vertices and edges are arranged in a drawing has an impact on its readability, utility, fabrication cost, and aesthetic.
In this project, the method called spectral method is used which is a class of algorithm for drawing graphs. Two classes of algorithms for drawing graphs will be discussed in this report. Spectral and Force-directed graph drawing algorithm.

Spectral Method
The spectral layout is a type of graph-drawing algorithm. The eigenvectors of a matrix, such as a graph's Laplace matrix, are used as Cartesian coordinates for the graph's vertices in this configuration. Spectral drawing is a projection of a three-dimensional drawing. This is because the two-dimensional drawing is just the first two coordinates of the three-dimensional drawing. The node placements are determined by the matrix's second and third-biggest eigenvectors, with the second Eigenvector determining the x coordinate and the third determining the y coordinate.

Force-directed Method
The goal of this class of algorithms is to place the nodes of a graph in two-dimensional or three-dimensional space so that all of the edges are of roughly equal length and there are a few crossing edges as possible, by assigning forces between the set of edges and the set of vertices based on their relative locations, and then using these forces to either simulate the motion of the edges and nodes or to minimize their energy.

1.2 Physical synthesis of circuits

Several steps are involved in the design and synthesis of an electrical circuit. In the placement step, each cell or logic unit in the circuit is determined in global and detailed placement phases. The global placement provides an approximate position, whereas the detailed placement fine-tunes the final placements. Some quality parameters, such as wire length, wire congestion, and signal delay, are estimated by the placer.
Analytics approaches can be used to model problems as a mathematical study of the minimization of an objective function. Undirected graphs can be used to model cells and nets, with nodes representing cells and edges representing wires. The spectrum approaches use graph attributes to locate each node, while a wire length estimation is minimized.
When compared to force-directed approaches, spectral methods have two key advantages. The global optimum can be calculated quickly, and the energy function has $O(|E|)$ entries rather than $O\left(|V|^2\right)$ entries.

2 Terms and concepts

2.1 Eigenvectors and Eigenvalues

The eigenvalues of a graph's adjacency matrix are its eigenvalues. The eigenvectors of a linear operator in linear algebra are non-zero vectors that, when transformed by the operator, result in a scalar multiple of themselves, so they do not change direction. The eigenvalue refers to the scalar lambda.

$$Ax = \lambda x$$
$$(A - \lambda I)x = 0$$

So the matrix $(\boldsymbol{A} - \lambda \boldsymbol{I})$ is singular $\Rightarrow \det(\boldsymbol{A} - \lambda \boldsymbol{I}) = 0$
The characteristic polynomial of degree n is $p(\lambda) = \det(\boldsymbol{A} - \lambda \boldsymbol{I})$.

Example:

$$\boldsymbol{A} = \begin{pmatrix} 2 & 1 \\ 4 & 2 \end{pmatrix} \quad \det \begin{pmatrix} 2-\lambda & 1 \\ 4 & 2-\lambda \end{pmatrix} = 0$$

Solution of characteristic polynomial gives: $\lambda_1 = 4, \lambda_2 = 0$ To get the eigenvectors, we solve: $\boldsymbol{Ax} = \lambda \boldsymbol{x}$

$$\begin{pmatrix} 2-(4) & 1 \\ 4 & 2-(4) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$
$$\begin{pmatrix} 2-(0) & 1 \\ 4 & 2-(0) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad x = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

2.2 Adjacency matrix

An adjacency matrix is a square matrix used to represent a finite graph in graph theory. The matrix's elements show whether two vertices in the graph are adjacent or not. The adjacency matrix is a (0,1)-matrix with zeros on its diagonal in the special situation of a finite simple graph.

$$A_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are connected} \\ 0 & \text{otherwise} \end{cases}$$

2.3 Degree matrix

The degree matrix of an undirected graph is a diagonal matrix that contains information about the degree of each vertex—that is, the number of edges attached to each vertex. It is used in conjunction with the adjacency matrix to construct the Laplacian matrix of a graph: the difference between the degree matrix and the adjacency matrix is the Laplacian matrix.

$$D_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

2.4 Spectral layout

The spectral layout is a graph drawing approach[1] that performs remarkably well on normal graphs. The second and third largest eigenvectors of the graph's Laplacian matrix are used to calculate the coordinates of each node. Subtracting the adjacency matrix from the graph's degree matrix yields the Laplacian matrix. The Laplacian matrix LG of graph G is L = D - A, where D is the degree matrix and A is the graph's adjacency matrix.

3 Methods

The spectral technique has two key benefits that make it particularly appealing. First, it gives us a mathematically sound formulation that leads to an exact layout solution. The good speed of computing is the second benefit. Three approaches were investigated in this work. **Laplacian**, **Laplacian Normalized** and **Optimized Degree-Normalized method**. Algorithms were coded using "Python" which has many useful packages like "Networkx" for generation random graphs and exporting the layouts to the files. Packages like "NumPy" are also helpful to work with matrices.

3.1 Derivation of the Eigen-Projection Methods (Laplacian)

The problem is then explained as the squared wire-length minimization. Let $x(i)$ be the position of node I in the horizontal axis.

$$\min_x E(x) = \sum_{(i,j)\in E} (x(i) - x(j))^2$$
$$\text{given } \operatorname{Var}(x) = 1$$

The amount of energy to be saved must be kept to a minimum. The distance between nodes is reduced by *E(x)*, while the variance *Var(x)* prevents the nodes from compressing into a single location. The problem can alternatively be phrased in terms of the Laplacian matrix L, which is defined as $L = D - A$.

$$L(i,j) = \begin{cases} d_i & \text{if } i = j \\ -1 & \text{if } i \text{ and } j \text{ are connected} \\ 0 & \text{otherwise} \end{cases}$$

So we finally get

$$\min_x x^T L x$$
$$\text{given } x^T x = 1$$
$$\text{in the subspace } x^T \cdot 1_n = 0$$

Where the eigenproblem $Lu_i = \lambda_i u_i$ is solved to find the solution for x. $x = u_2$ determines the nodes' horizontal locations. The next eigenvector $y = u_3$, which is orthogonal to x, can be used to obtain the vertical coordinates.

3.2 Degree-Normalized Eigenvectors (Laplacian Normalized)

A minor change to the model can produce more intriguing findings. To lower the costs of their multiple edges, nodes with a high degree should be located near the center. In the model below,

$$\min_x \frac{x^T L x}{x^T D x}$$
$$\text{in the subspace } x^T D 1_n = 0$$

The numerator clusters high-degree nodes around the center, whereas the denominator scatters them out. As a result, the graph is more proportional, avoiding low-degree vertices placed far away. The following generalized eigenproblem can be used to find the answer to this new model:

$$Lu_i = \lambda_i D u_i$$

And using the associated eigenvector u2 of the second lowest eigenvalue λ_2 we get the horizontal positions of the nodes $x = u_2$. For the vertical positions, we use the third eigenvector, $y = u_2$.

3.3 Optimized degree-normalized eigenvectors using power-iteration method

The degree-normalized eigenvectors are appealing because they may be generated using an intuitive technique that is directly tied to their aesthetic features. This algorithm works by inserting each node at the weighted centroid of its neighbors iteratively.
It can be demonstrated by noting that putting each node at the weighted centroid of its neighbors is the same as multiplying by the transition matrix$-D^{-1}A$. The procedure described can be stated concisely as,

$$\begin{cases} x_0 = \text{ random vector, s.t. } x_0^T D 1_n = 0 \\ x_{i+1} = D^{-1}Ax_i \end{cases}$$

This is referred to as the power-iteration process. Scaling the eigenvalues to the range [0, 1] is more intuitive, therefore the algorithm works with the matrix $\frac{1}{2}\left(I + D^{-1}A\right)$.

Algorithm 1: Degree-normalized eigenvectors of Laplacian Matrix

Function SpectralDrawing ($G -$ the input graph , k - dimension)
% This function computes $u_2, \ldots, u_k$, the top (non-degenerate) eigenvectors of $D^{-1}A$.
const $\epsilon \leftarrow 10^{-7}$% tolerance
for $i = 2$ to k do
$\hat{u}_i \leftarrow$ random % random initialization
$\hat{u}_i \leftarrow \dfrac{\hat{u}_i}{\|\hat{u}_i\|}$
do
$u_i \leftarrow \hat{u}_i$
$\%D -$ Orthogonalize against previous eigenvectors:
for $j = 1$ to $i - 1$ do
$u_i \leftarrow u_i - \dfrac{u_i^T D u_j}{u_j^T D u_j} u_j$
end for
% multiply with $\dfrac{1}{2}\left(I + D^{-1}A\right)$:
for $j = 1$ to n do
$\hat{u}_i(j) \leftarrow \dfrac{1}{2} \cdot \left(u_i(j) + \dfrac{\sum_{k \in N(j)} w_{jk} u_i(k)}{\deg(\mathrm{j})} \right)$
end for
$\hat{u}_i \leftarrow \dfrac{\hat{u}_i}{\|\hat{u}_i\|}$ % normalization
while $u_i < 1 - \epsilon$ % halt when direction change is negligible
$u_i \leftarrow \hat{u}_i$
end for
return $u_2, \ldots, u_k$

Koren[2] employs the power-iteration concept and the D-Orthogonality of the eigenvectors to compute the Eigenvector uk. It chooses some random x, such that x is D-orthogonal to $u^1, \ldots, u^{k-1}$, implying $x^T D u^p = 0$ for $p = 0$ to $k - 1$. Then if $x^T D u^k \neq 0$, it can be shown that the series,

$$\frac{1}{2}\left(I + D^{-1}A\right)x, \quad \left(\frac{1}{2}\left(I + D^{-1}A\right)\right)^2 x, \quad \left(\frac{1}{2}\left(I + D^{-1}A\right)\right)^3 x, \ldots,$$

will be converged in the direction of u^k.
Algorithm 1 shows the whole procedure for computing the Degree-normalized eigenvectors.

4 Results

Case 1: A random Erdös-Renyí graph with n=200 p=0.15

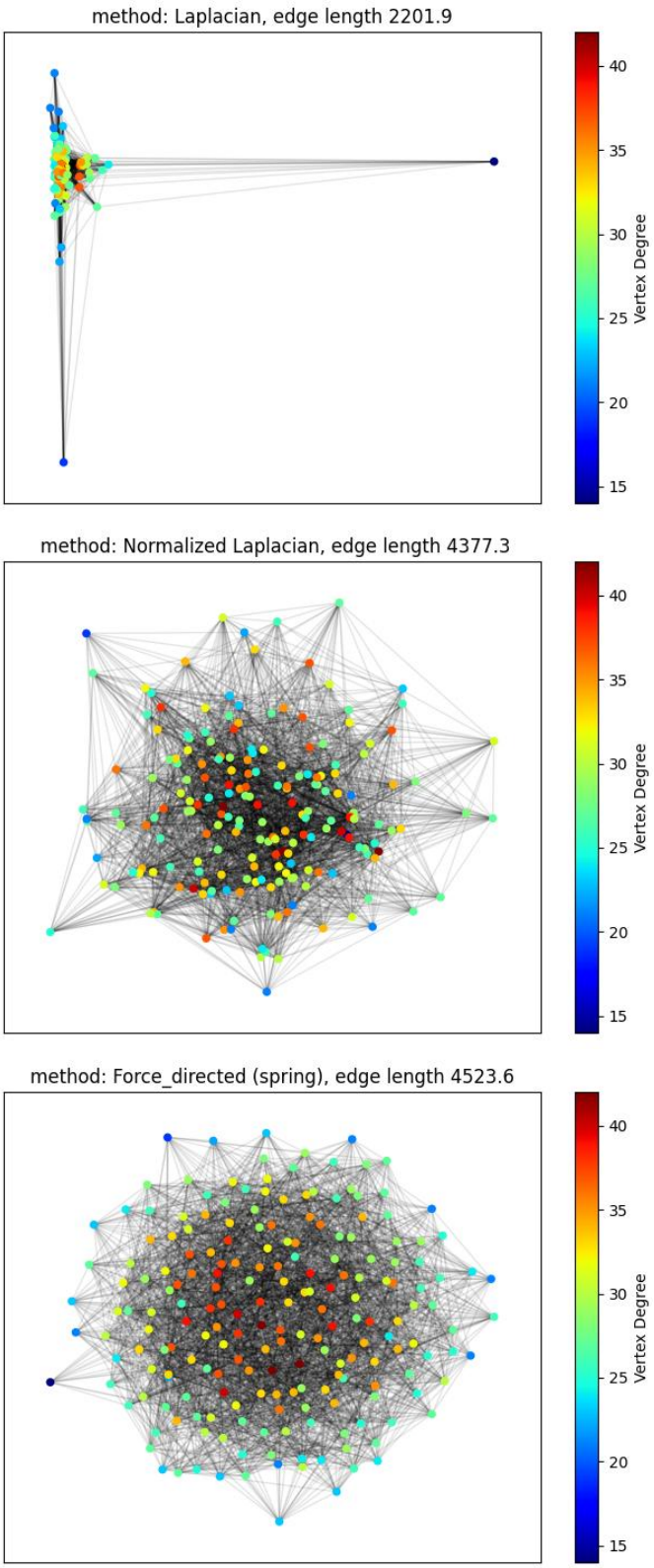

Figure 1: Erdos_Renyi random graph with 200 nodes and the probability for edge creation=0.15

.

Case 2: AG-Monien@grid1, 252 nodes, 476 edges.

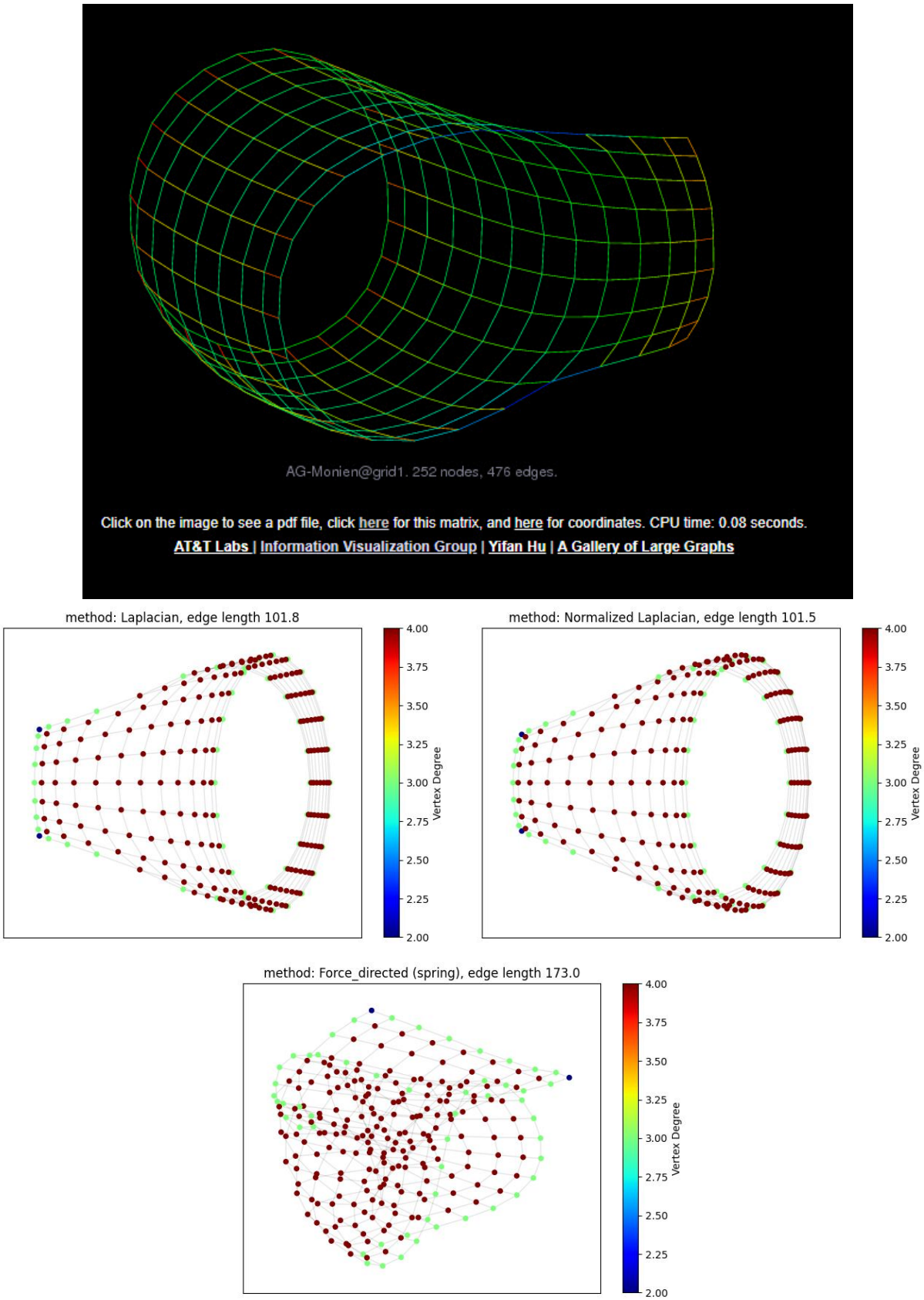

Figure 2: Source: `http://yifanhu.net/GALLERY/GRAPHS/GIF_SMALL/AG-Monien@grid1.html`
.

Case 3: AG-Monien@3elt, 4720 nodes, 13722 edges.

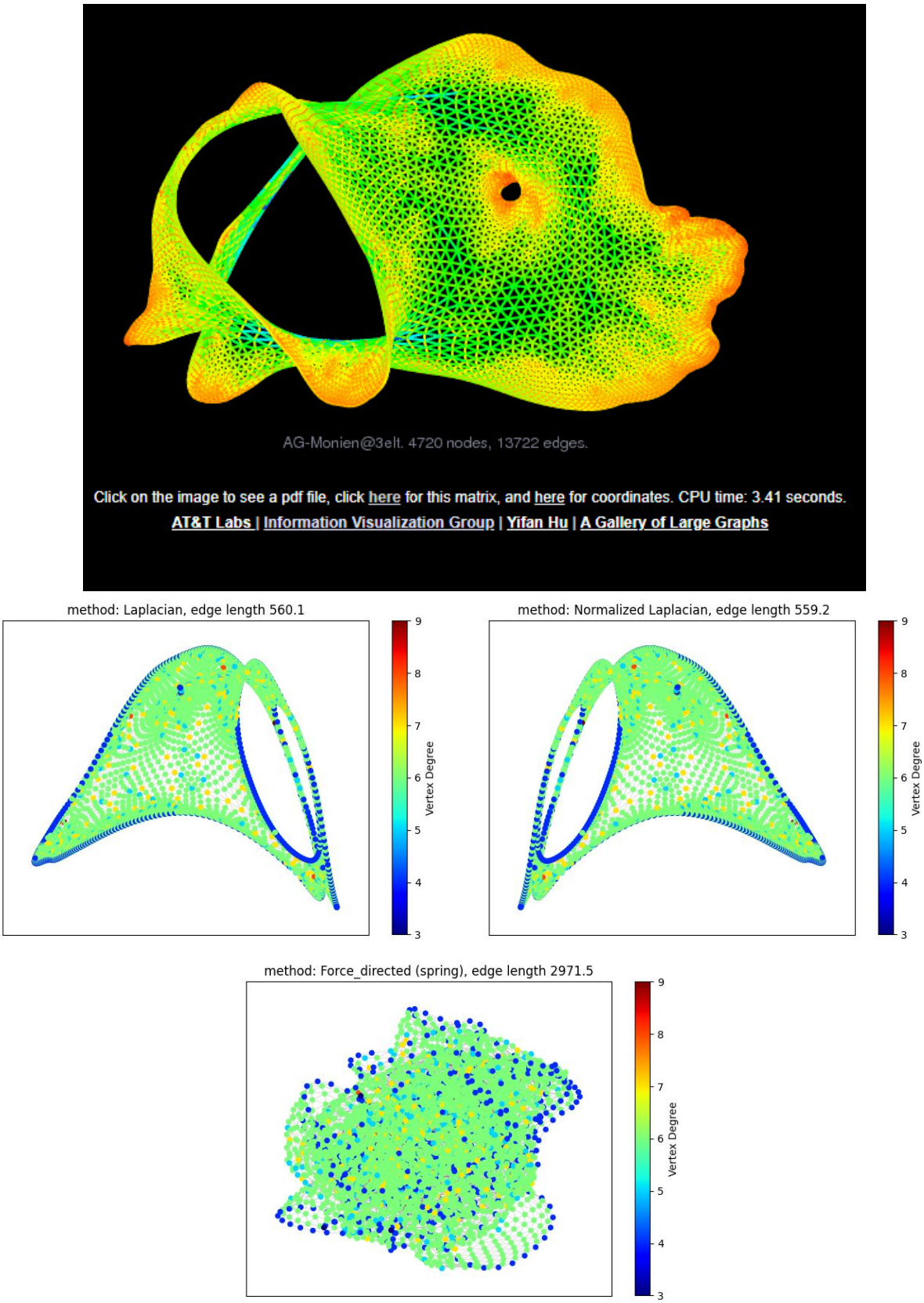

Figure 3: Source: `http://yifanhu.net/GALLERY/GRAPHS/GIF_SMALL/AG-Monien@3elt.html`
.

The method of force-directed placement using attractive and repulsive forces was also added for comparison in order to test and compare the results of the two algorithms. Two graphs of progressively larger sizes were utilized.

The first graph is a random Erdös-Renyí graph with 200 nodes and probability for edge creation $p = 0.15$. The second graph was chosen from the Collection of Graph visualization of matrices from the University of Florida.

The edge-length (wire-length in circuits) was calculated as the sum of the Euclidean lengths of the edges, and the positions were scaled to have a standard deviation of $sigma = 1$ so that wire-length could be compared across dimensions.

Figures 1,2 and 3 show the results of the two graphs using the three approaches previously stated.

5 Discussion

Figure 1, which is related to a random graph (with probability for edge creation=0.15), shows that all high degree nodes are positioned in the center for all three techniques. On the other hand, the Laplacian approach produces a layout with a lot of overlapping nodes and a lot of variation in the node edges. A few low-degree nodes were positioned very far away. The normalized technique solves this problem by arranging the nodes in a more uniform manner while preserving the nodes with the highest degree in the center.

The Laplacian method still finds the best result in terms of wire length, but normalized and force-directed approaches appear more acceptable in an actual case.

In figure 1, the total minimized edge-length using the Laplacian method has a significant difference in comparison to the normalized version. The total edge-length of the random graph in the Laplacian plot is 2201.9, while this amount in the Normalized Laplacian is 4477.3. It seems the Laplacian method performed more efficiently to minimize the lengths. The force-directed method has an edge length close to the Normalized-Laplacian.

We know the spectrum of a graph's normalized Laplacian is scaled on the interval $[0, 2] \in \mathbb{R}$ and its eigenvalues will be suitably scaled.So we can see in Figure 1 that the normalized version in the case of random graphs has a more meaningful look than the Laplacian method.

In Figures 2 and 3, we can see an acceptable drawing in the Laplacian and normalized versions. But in the spring layout (force-directed method), as this algorithm tries to keep the edge distance close to each other, the result does not look like the predefined picture of the graph provided on the website. Regarding the minimization of the total edge length, in Figures 2 and 3, it is evident that Laplacian and Normalized-Laplacian performed much more efficiently than the spring method. In figure 2, the total length for Laplacian and normalized version is 101.8 and 101.5, respectively, while this number in the force-directed method is 173. This difference in figure 3 is much greater.
(Laplacian: 560.1, Normalized-Laplacian: 559.2 and force-directed: 2971.5)
So we can see for the big and dense graphs that spectral methods have advantages like reaching to lower total edge length. In terms of aesthetics, it shows superiority compared to force-directed.

We can see that the Laplacian approach uses a lot of overlapping nodes to obtain a very short wire-length solution.

The Koren[1] paper's methodology, which is based on power iteration, should have been used to extract only the required eigenvectors, resulting in a fast strategy of achieving the global minimum. But despite the long debugging procedure and time spent, it was not finalized to compare the timing of the drawing.

Comments

- In the presentation of this project, the implementation of the optimized degree-normalized method, which is an optimization process in the Spectral method and based on the power-iteration method, was incomplete, unfortunately. The code scrip was implemented in python at the beginning of the codes by the function called **"power_Iteration_degree_normalized_Koren_layout."**.
- After a long debugging process, the correct results (second and third eigenvectors) were not obtained. Implementation of the two methods Laplacian and Normalized-Laplacian were successful, and in order to be able to compare the methods and complete the project, "Spring layout," which is a Force-directed graph drawing, is used by calling the function called "nx.layout.spring_layout(G)" in the Networks package. This function positions nodes using Fruchterman-Reingold force-directed algorithm.
- Sample graphs were selected from the first page of to verify the results: `http://yifanhu.net/GALLERY/GRAPHS/` (Graph visualization of matrices from the University of Florida Collection)
- As I couldn't finish implementing the optimized version of the algorithm, drawing graphs from the provided links will take a couple of minutes, depending on how big the graph is and the computer's processing power.

References

[1] Yehuda Koren, *Drawing Graphs by Eigenvectors: Theory and Practice* Computers and Mathematics with Apphcatmns 49 (2005) 1867-1888.

[2] Yehuda Koren, *On Spectral Graph Drawing* Proceedings of the 9th Annual International Conference on Computing and Combinatorics, p496–508, 2003.

[3] Yehuda Koren, Liran Carmel, and David Harel, *Drawing huge graphs by algebraic multigrid optimization* Multiscale Modelling and Simulation, Society for Industrial and Applied Mathematics, Vol. 1, No. 4, pp. 645–673, 2003.