ALGORITHMS FOR VLSI SPECTRAL GRAPH DRAWING

Ali A.Yarmohammadi

Master students in Innovation and Research in Informatics
Polytechnic University of Catalonia (UPC)
Faculty of Computer Science - FIB

Project Presentation, January 2022

Table of Contents

1. Introduction
2. Terms and concepts
3. Methods
4. Results

Table of Contents

1 Introduction

2 Terms and concepts

3 Methods

4 Results

Graph Drawing

- A graph drawing is a visual representation of the vertices and edges of a graph.
- 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.

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.
- 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 so that all of the edges are of roughly equal length and there are a few crossing edges as possible.
- It is achieved by assigning forces between the set of edges and the set of vertices based on their relative locations.
- Using these forces to either simulate the motion of the edges and nodes or to minimize their energy.

Introduction

Physical synthesis of circuits

- 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.
- The spectrum approaches use graph attributes to locate each node, while a wire length estimation is minimized.

Table of Contents

1 Introduction

2 Terms and concepts

3 Methods

4 Results

Terms and concepts

Eigenvectors and Eigenvalues

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

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 $p(\lambda) = \det(\boldsymbol{A} - \lambda \boldsymbol{I})$.

Eigenvectors and Eigenvalues

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: $\boldsymbol{Ax} = \lambda \boldsymbol{x}$

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

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.

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

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.

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

Spectral layout

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.

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

Table of Contents

1 Introduction

2 Terms and concepts

3 Methods

4 Results

Methods

Three approaches were investigated in this work.

- **Derivation of the Eigen-Projection (Laplacian)**
- **Degree-Normalized Eigenvectors (Laplacian Normalized)**
- **Optimized degree-normalized eigenvectors using power-iteration**

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.

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.

$$\min_{x} E(x) = \sum_{(i,j)\in E} (x(i) - x(j))^2$$

$$\text{given } \ \text{Var}(x) = 1$$

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.

Derivation of the Eigen-Projection Methods (Laplacian)

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.

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 following generalized eigenproblem can be used to find the answer to this new model:

$$L u_i = \lambda_i D u_i$$

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, $y = u_2$.

Optimized degree-normalized eigenvectors using power-iteration

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,

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

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 $\frac{1}{2}\left(I + D^{-1}A\right)$.

Methods

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

$\hat{u}_i \leftarrow \frac{\hat{u}_i}{\|\hat{u}_i\|}$

do

$u_i \leftarrow \hat{u}_i$

$\%D -$ Orthogonalize against previous eigenvectors:

for $j = 1$ to $i - 1$ do

$u_i \leftarrow u_i - \frac{u_i^T D u_j}{u_j^T D u_j} u_j$

end for

% multiply with $\frac{1}{2}\left(I + D^{-1}A\right)$:

for $j = 1$ to n do

$\hat{u}_i(j) \leftarrow \frac{1}{2} \cdot \left(u_i(j) + \frac{\sum_{k \in N(j)} w_{jk} u_i(k)}{\deg\ (\mathrm{j})}\right)$

end for

$\hat{u}_i \leftarrow \frac{\hat{u}_i}{\|\hat{u}_i\|}$ % normalization

while $u_i < 1 - \epsilon$ % halt when direction change is negligible

$u_i \leftarrow \hat{u}_i$

end for

return $u_2, \ldots, u_k$

Table of Contents

1 Introduction

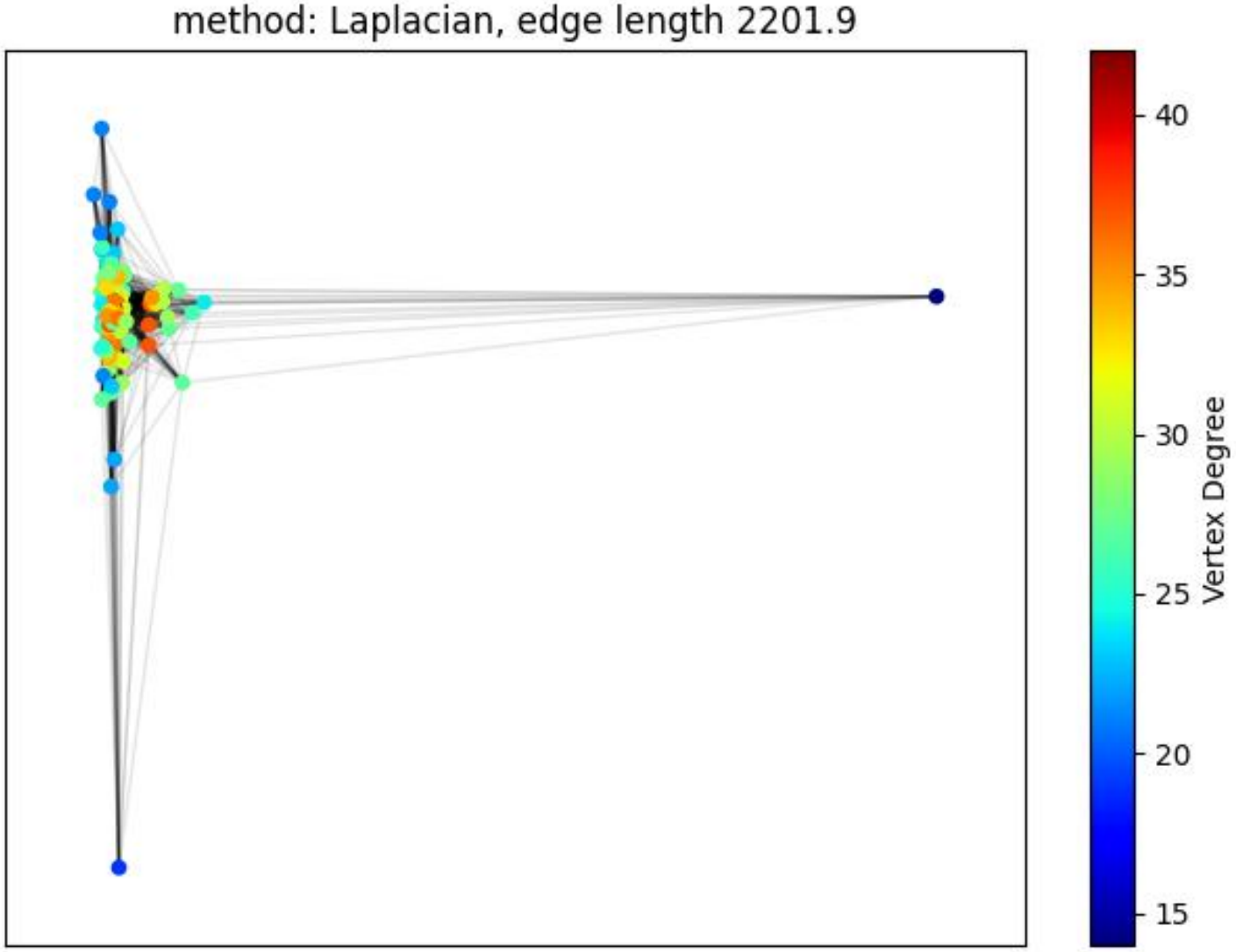
2 Terms and concepts

3 Methods

4 Results

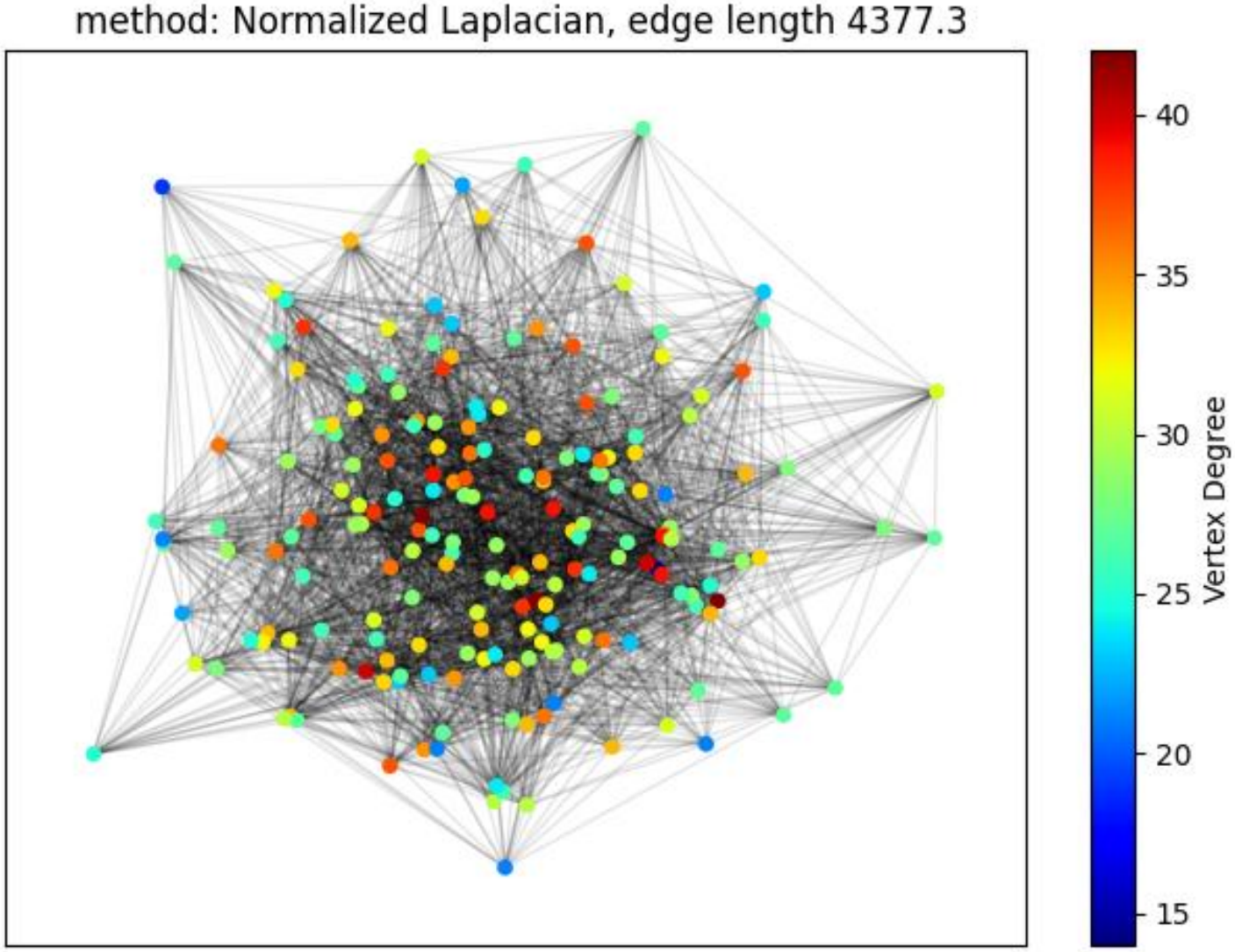
Results

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



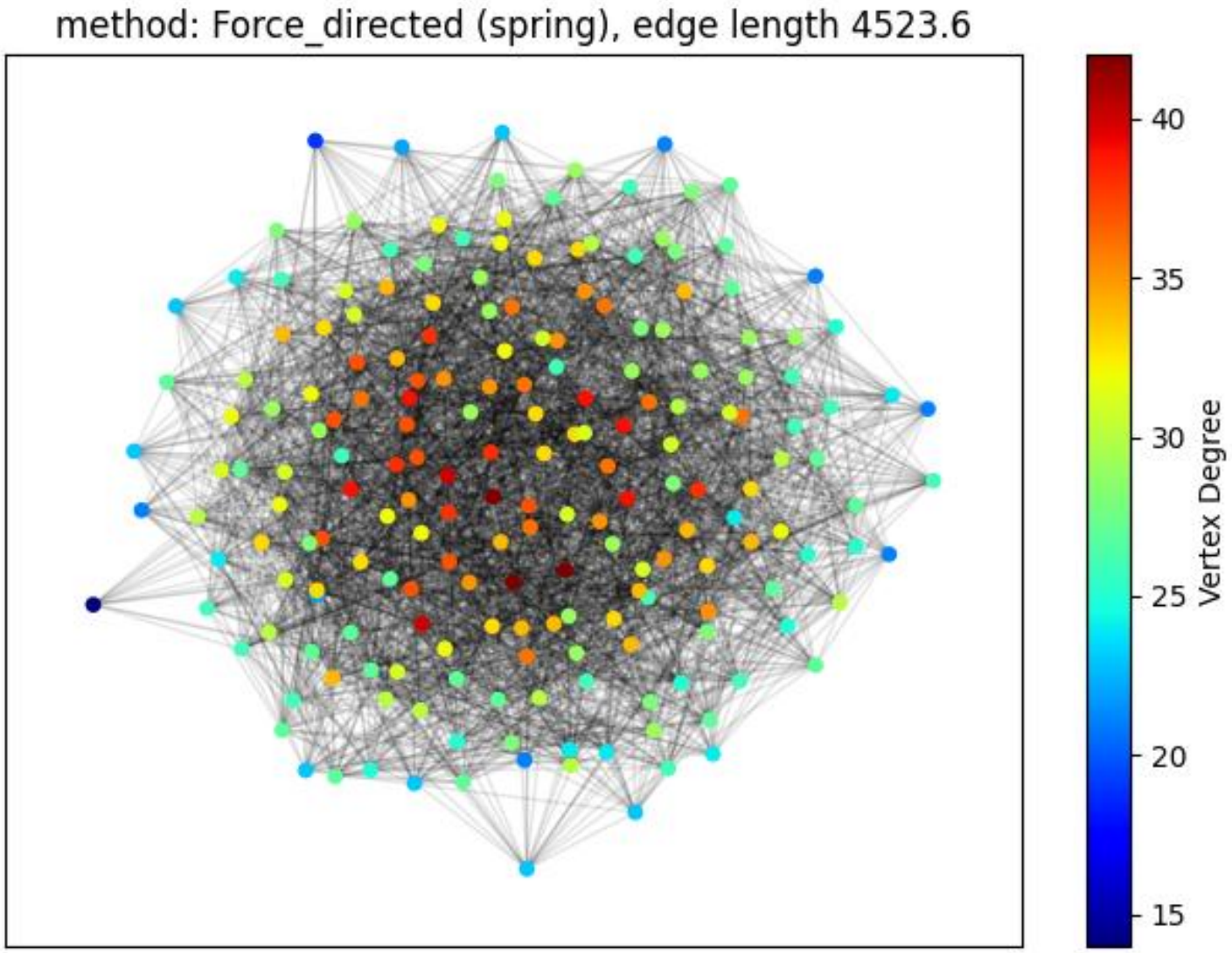
Results

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



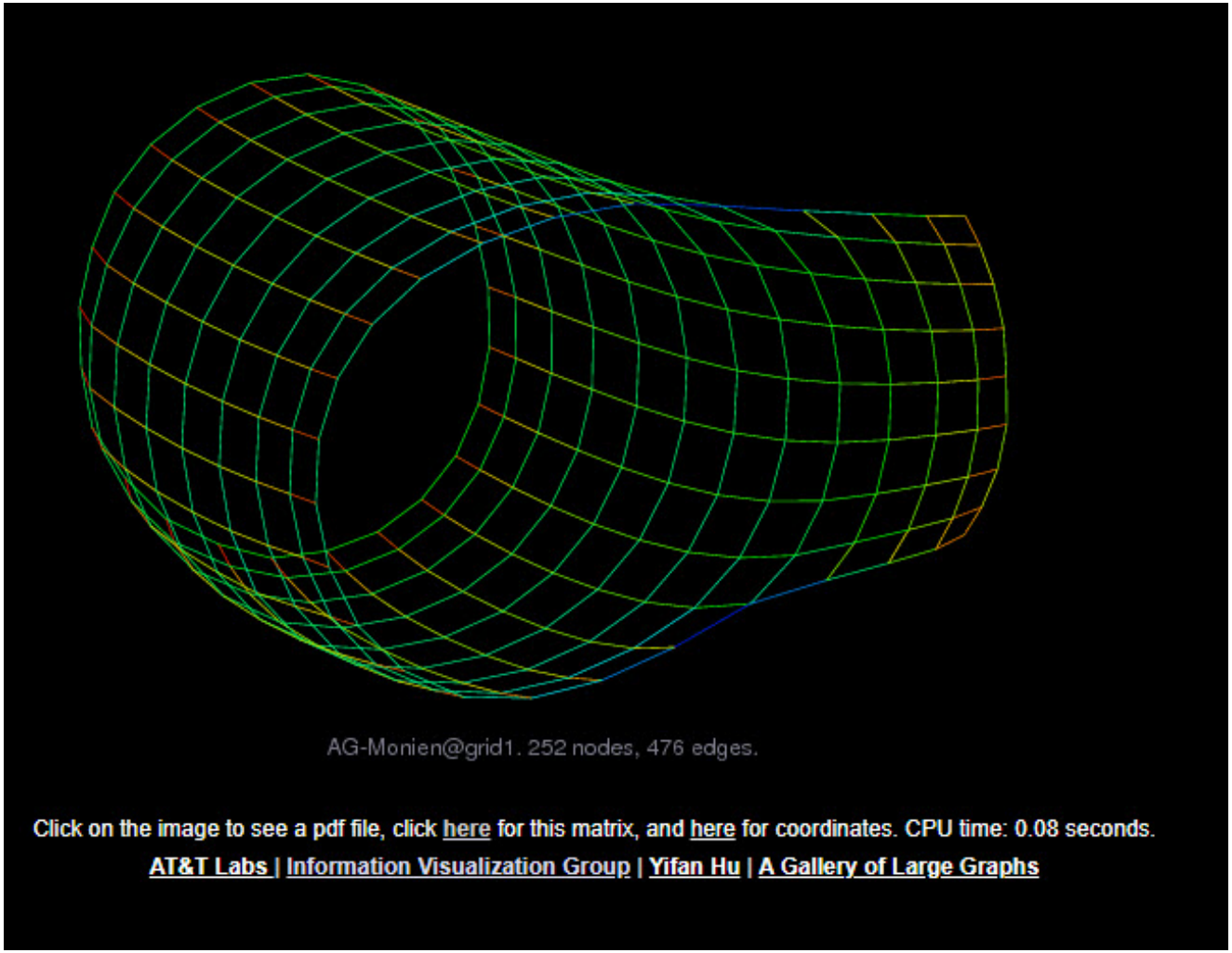
Results

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



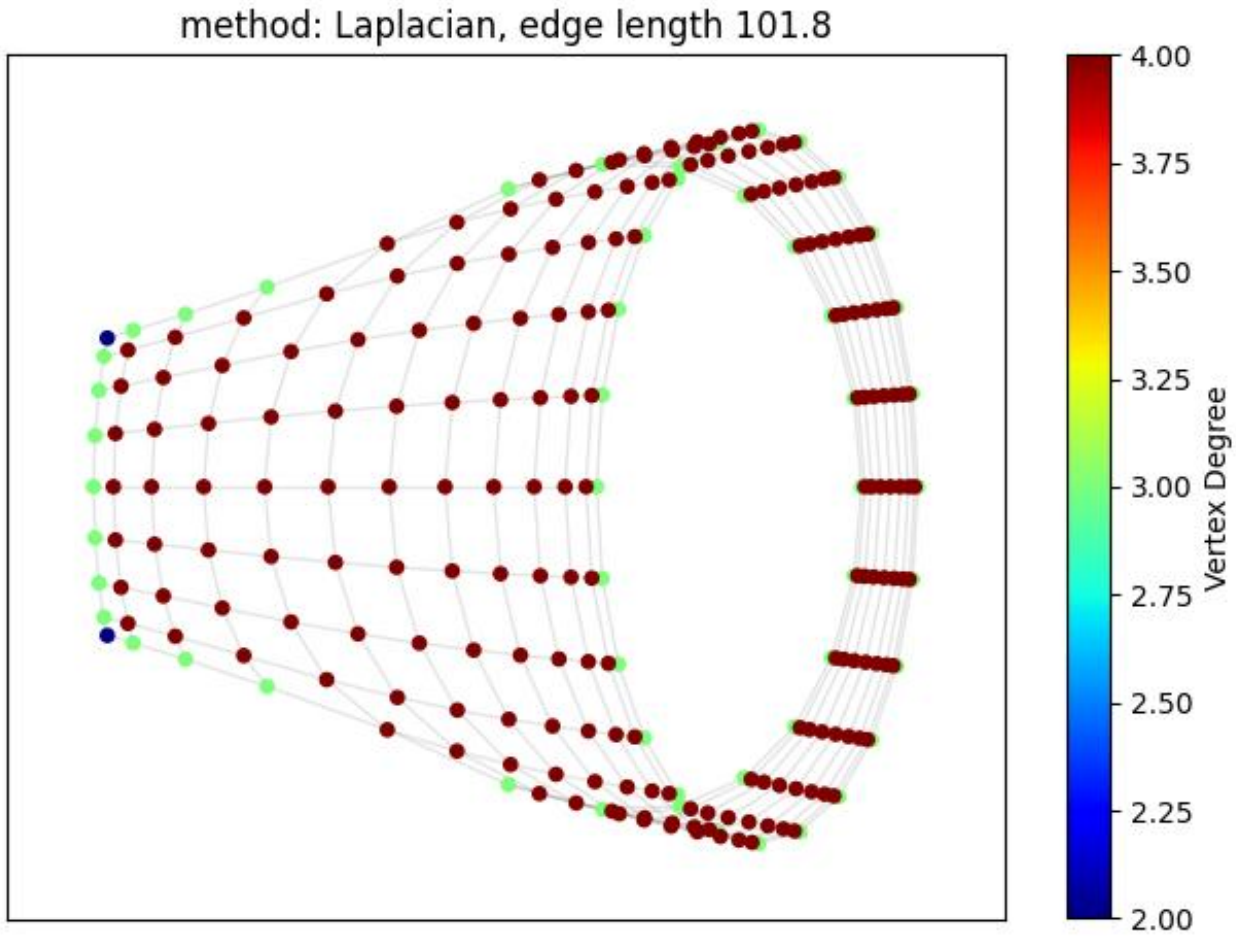
Results

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



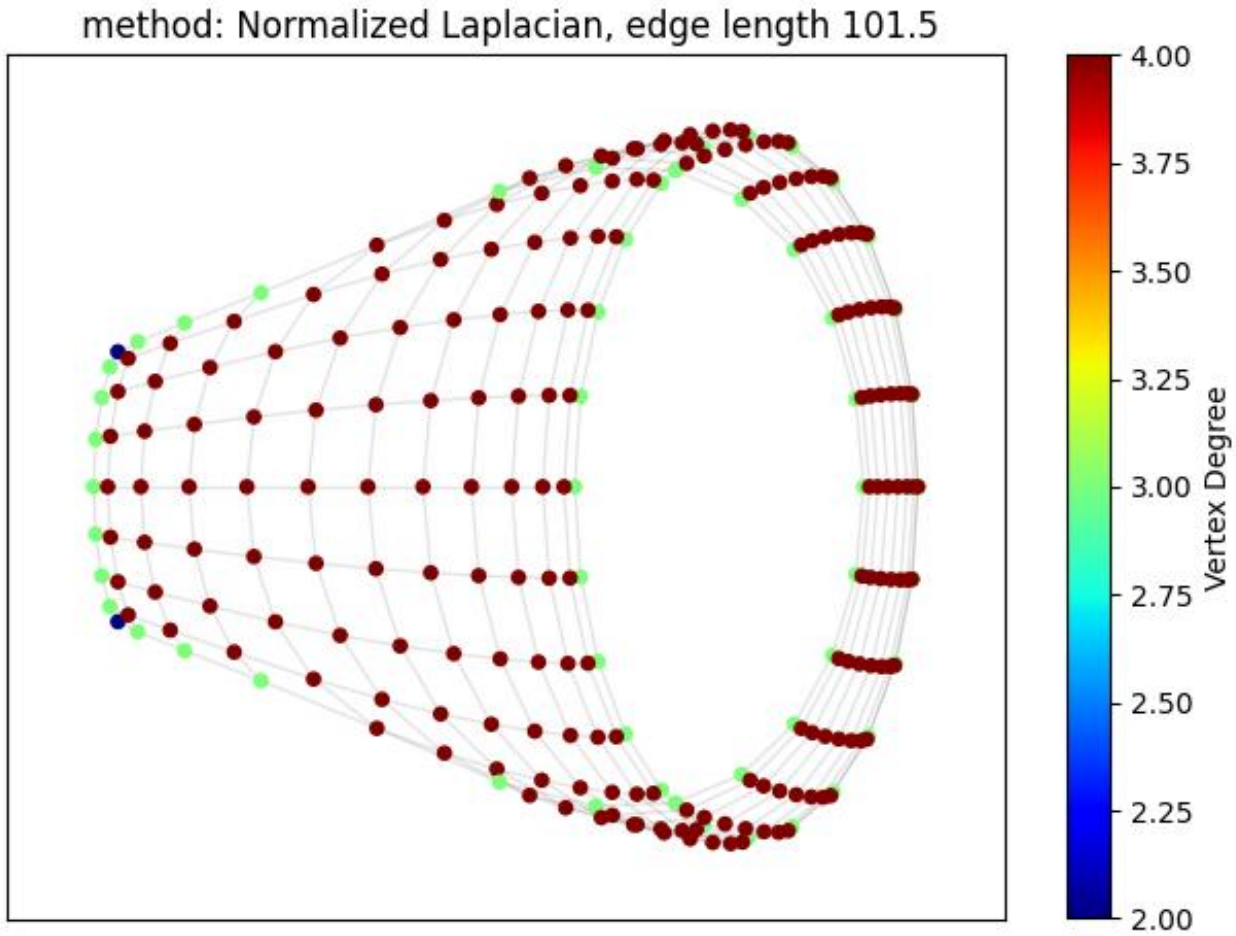
Results

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



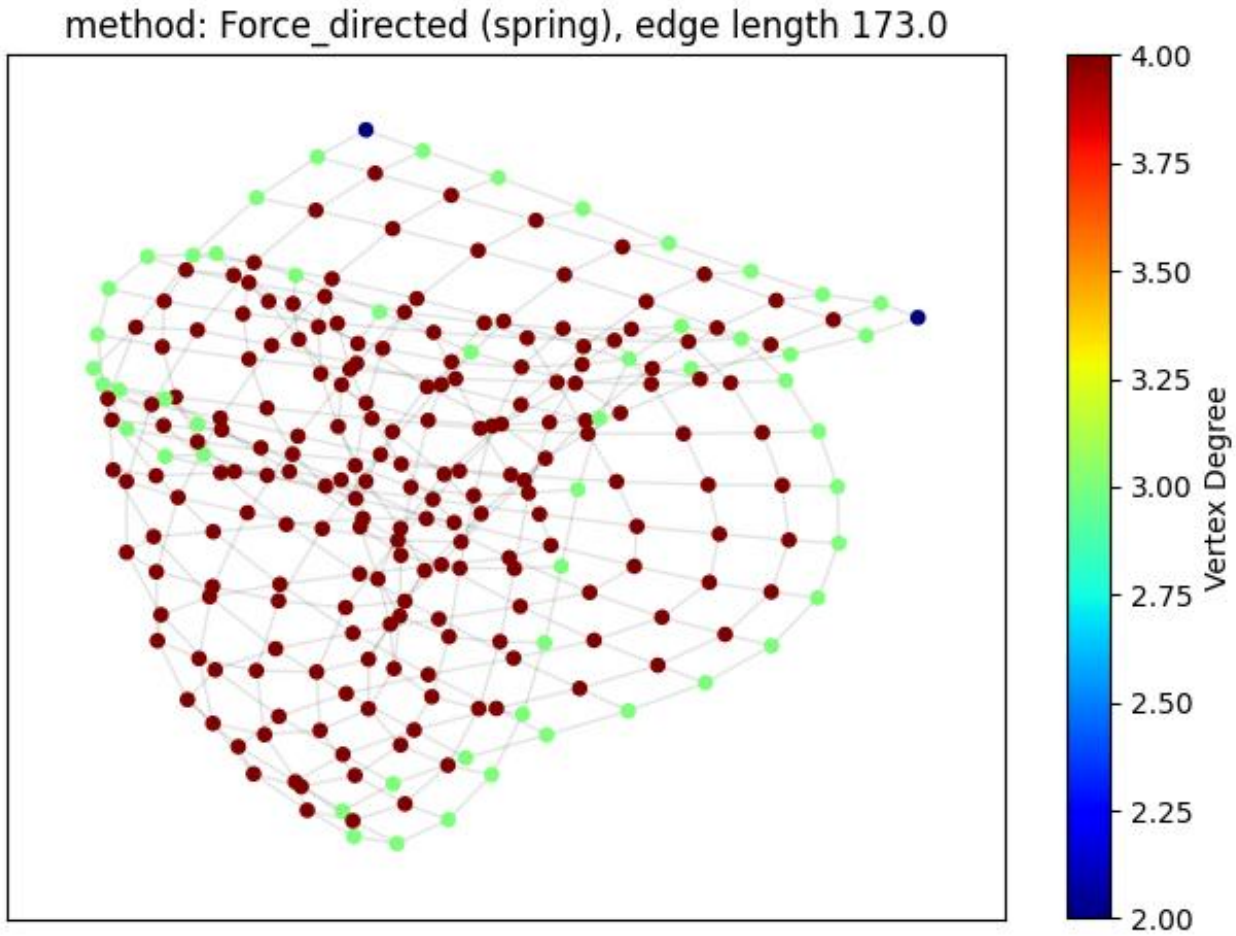
Results

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



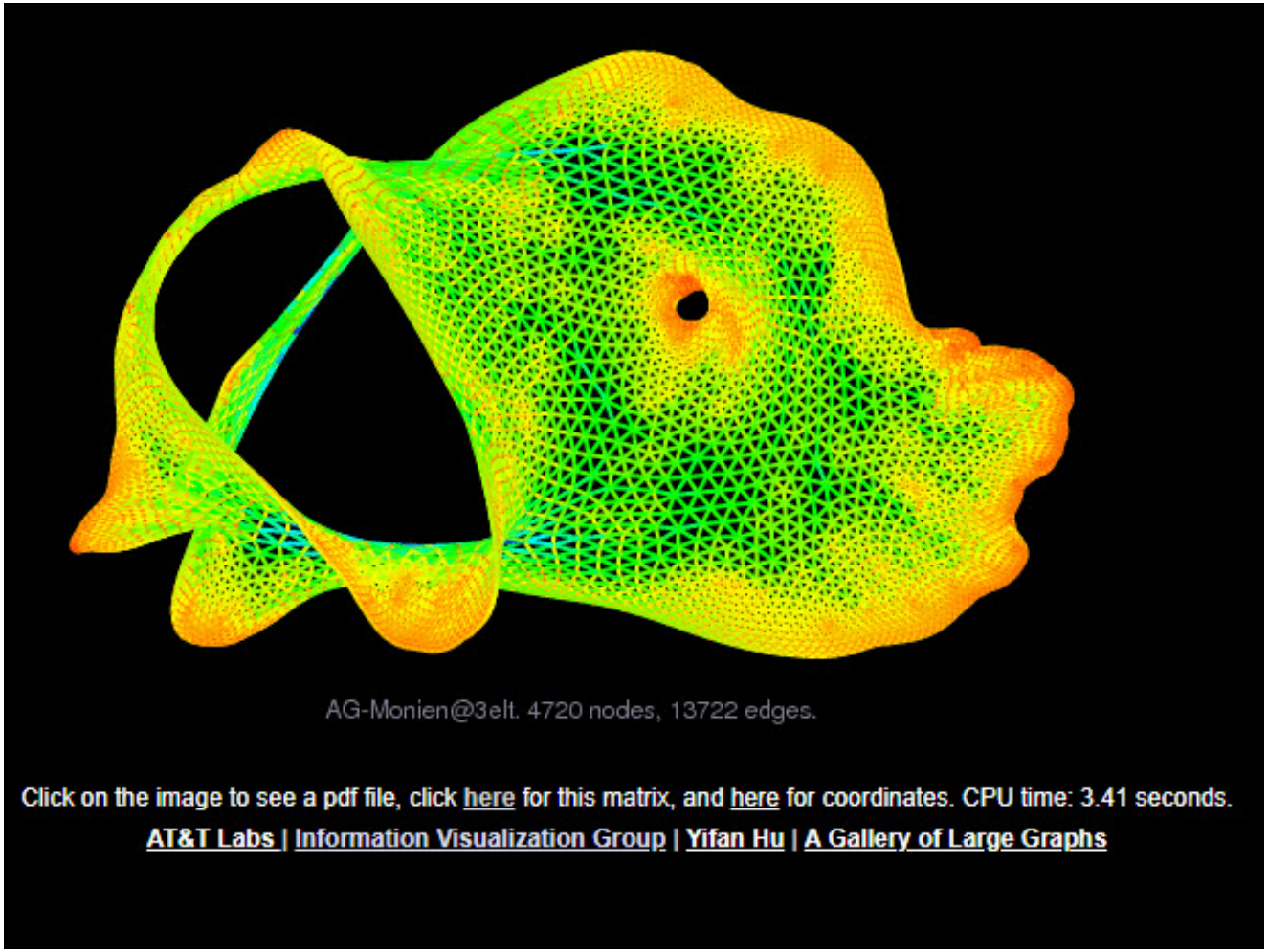
Results

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



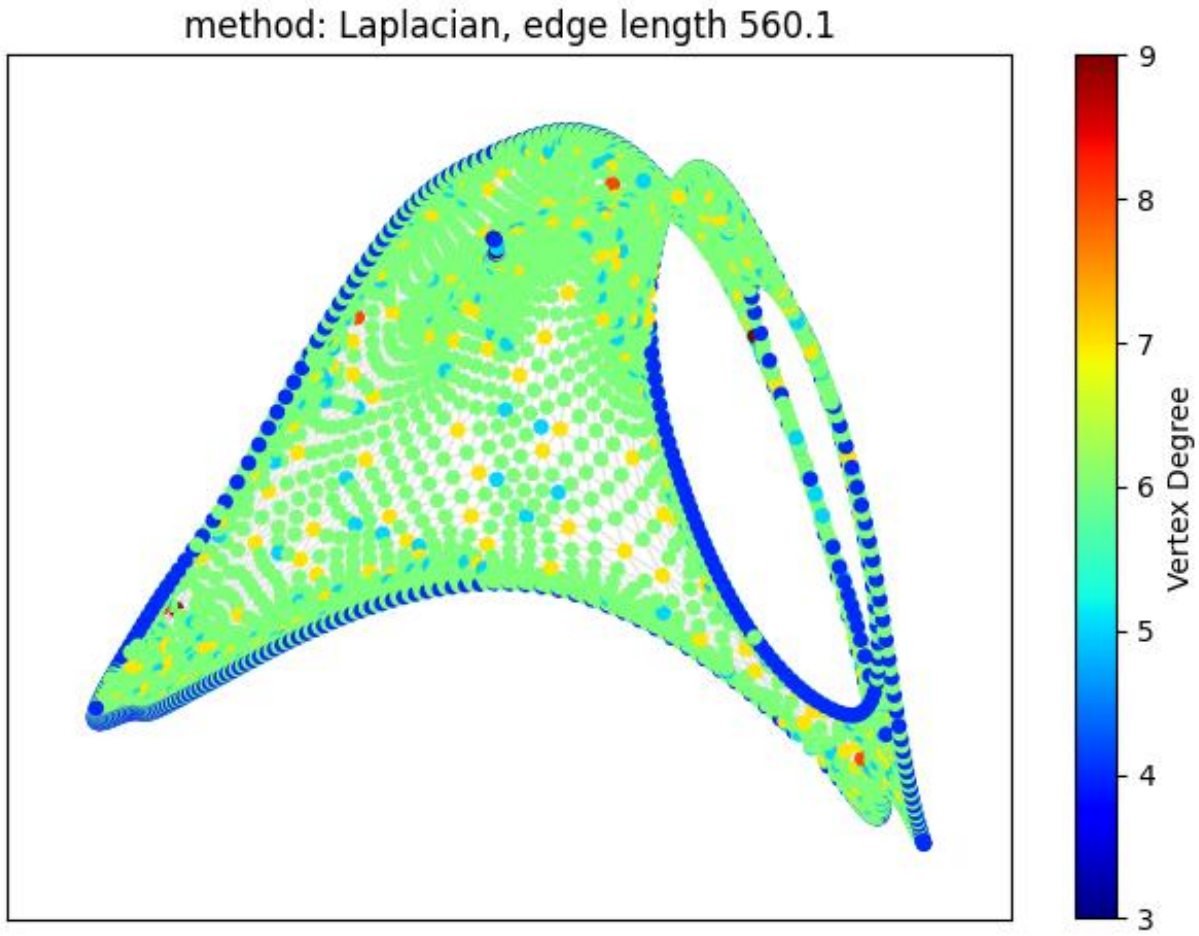
Results

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



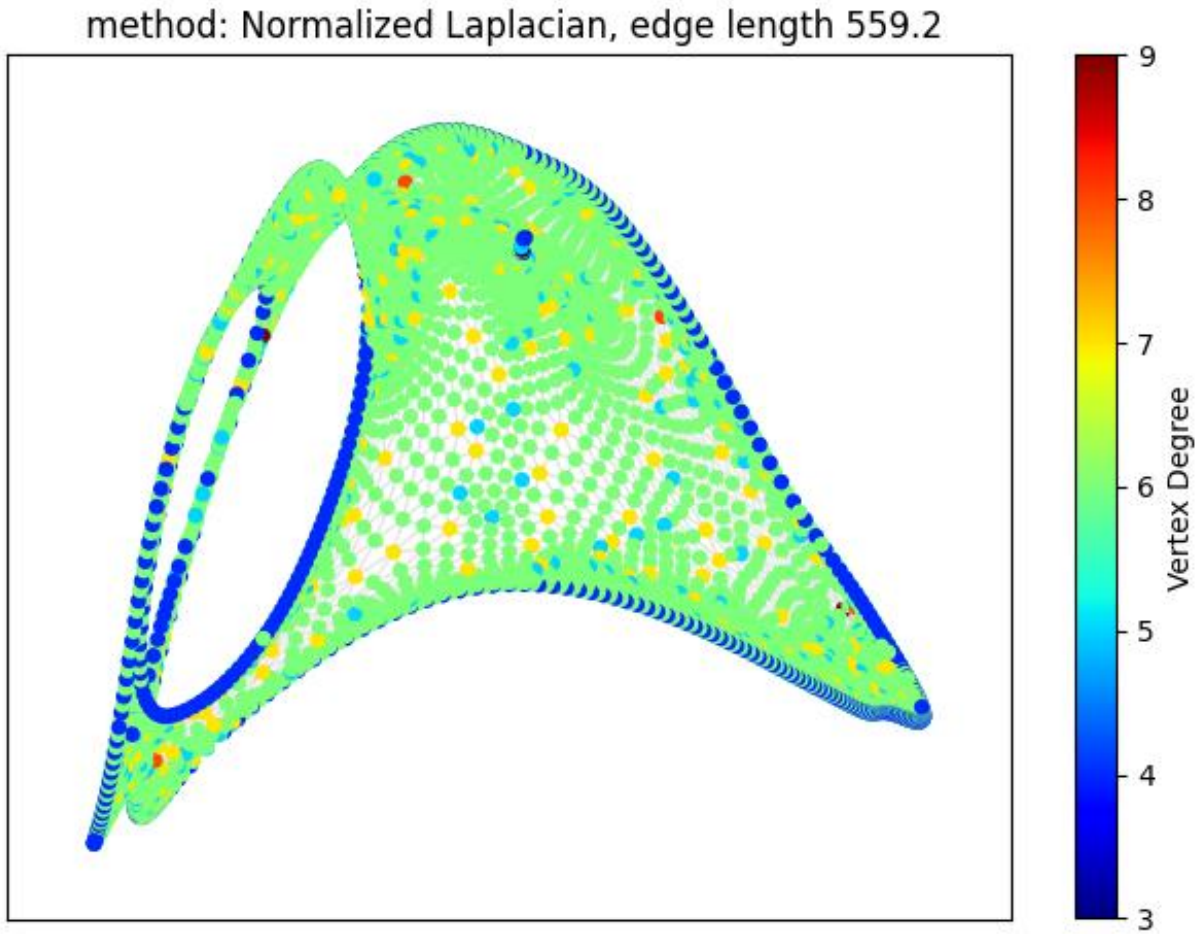
Results

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



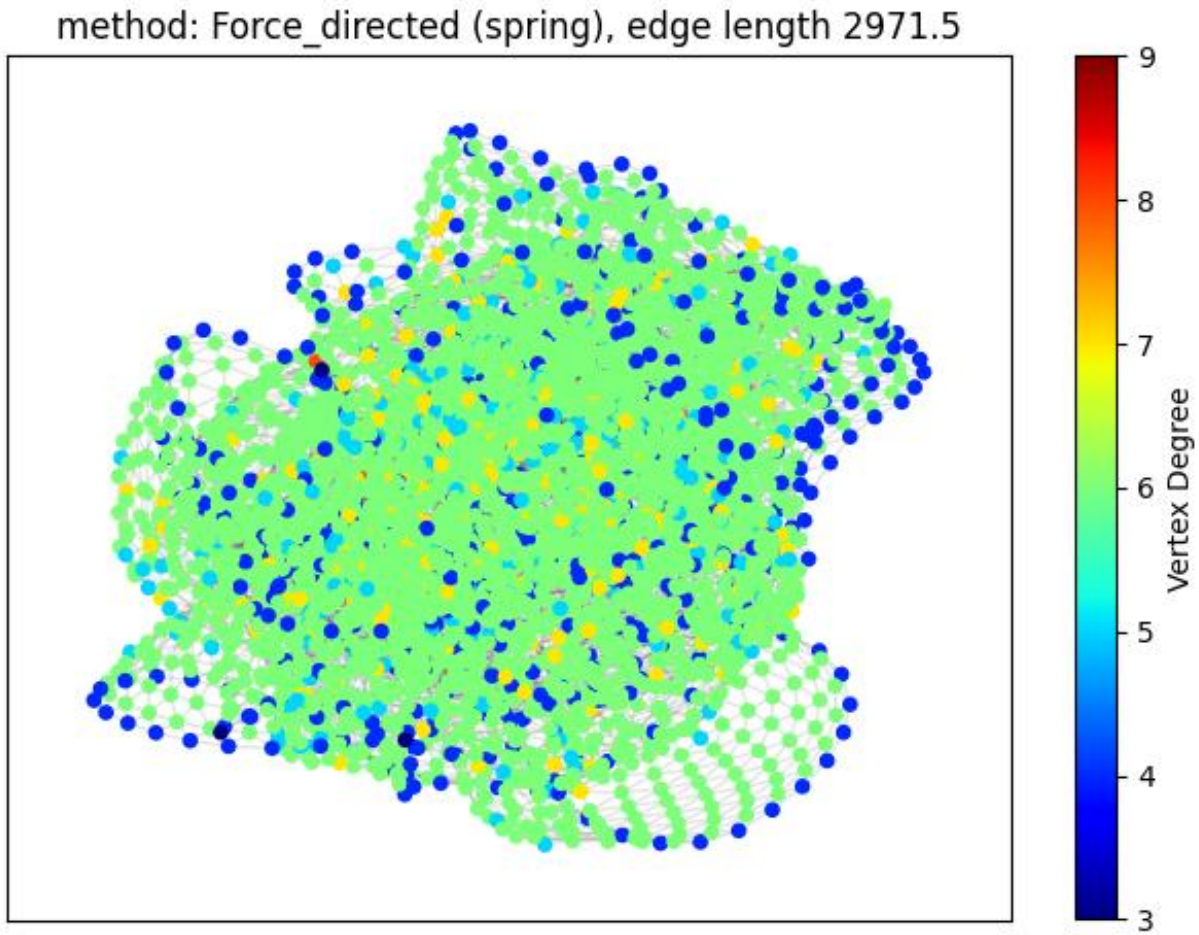
Results

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



Results

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



Thank you