

# Graph Visualization

Prof. O-Joun Lee

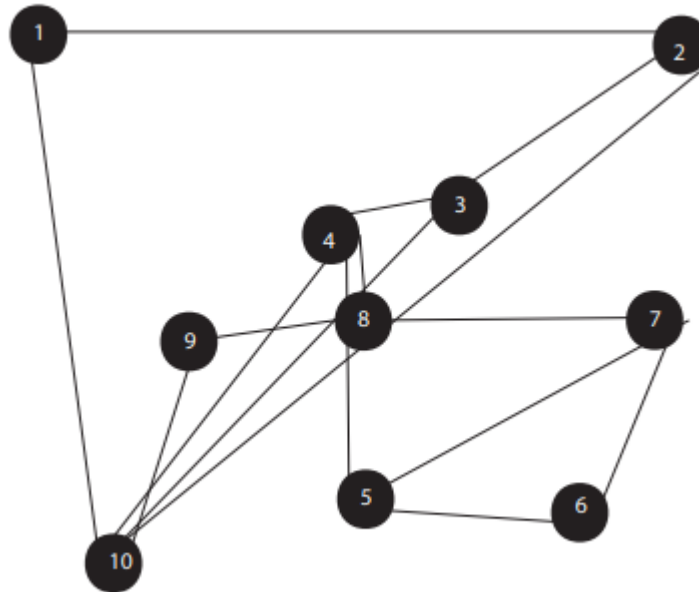
Dept. of Artificial Intelligence,  
The Catholic University of Korea  
*ojlee@catholic.ac.kr*

# Contents

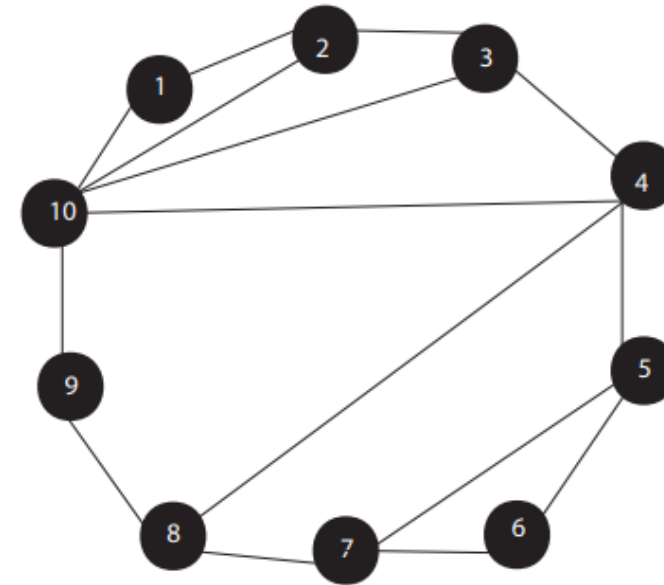
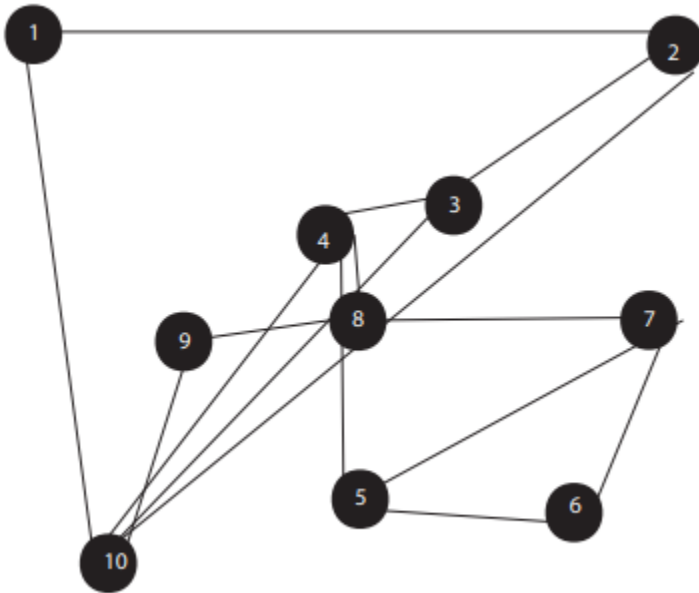


- Visualization techniques:
  - Spring-embedded
  - Circular, etc.
- Tools for graph exploration and visualization:
  - Gephi, Cytoscape, etc.

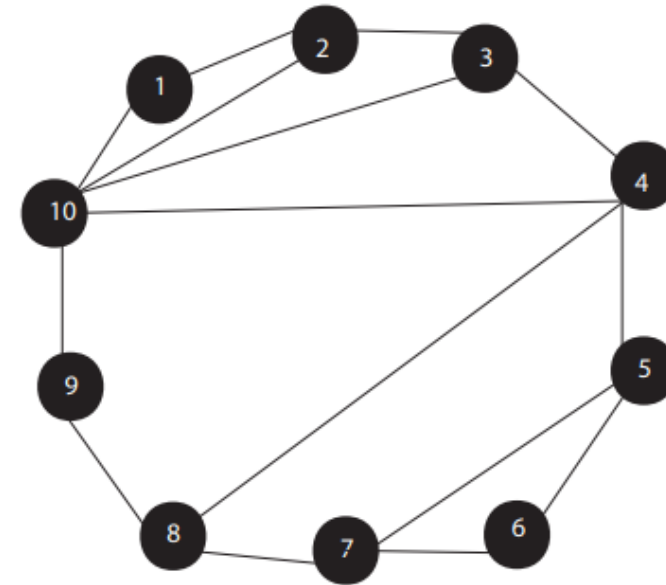
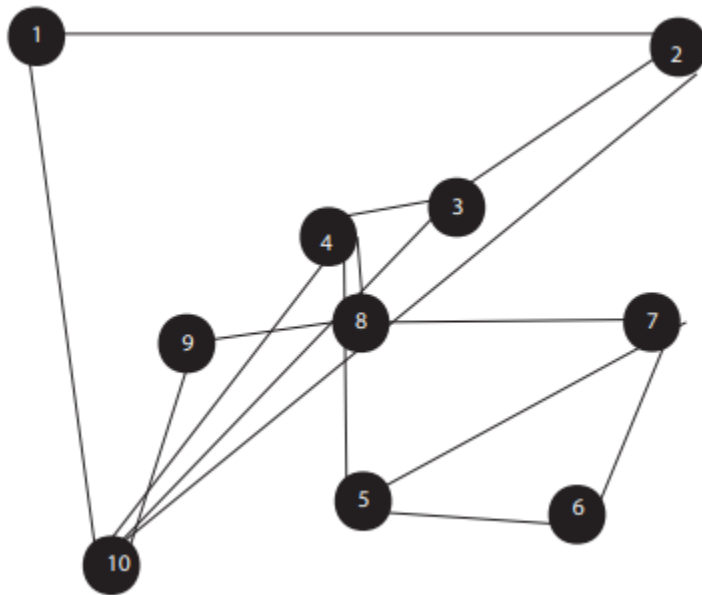
➤ **Input:** Graph  $G = (V, E)$



- **Input:** Graph  $G = (V, E)$
- **Output:** Clear and readable drawing of  $G$



- **Input:** Graph  $G = (V, E)$
- **Output:** Clear and readable drawing of  $G$



- Which criteria would you optimize?

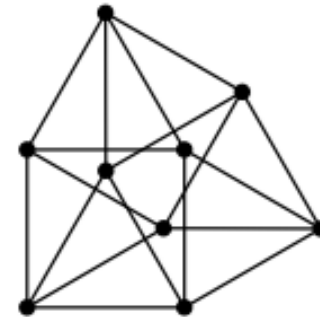
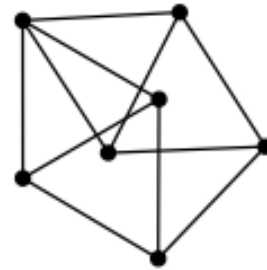
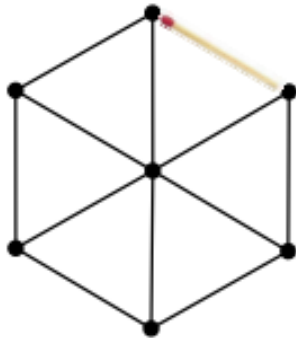
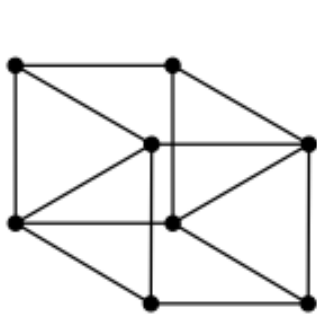
- **Input:** Graph  $G = (V, E)$
- **Output:** Creating clear and readable drawings of graph  $G$ .
- **Criteria:**
  - Adjacent nodes are close.
  - Non-adjacent nodes are far.
  - The preservation of edge length.
  - Densely connected nodes tend to close.
  - Draw  $G$  with as few crossings as possible.

- **Input:** Graph  $G = (V, E)$
- **Output:** Creating clear and readable drawings of graph  $G$ .
- **Criteria:**
  - Adjacent nodes are close.
  - Non-adjacent nodes are far.
  - The preservation of edge length: **similar length**.
  - Densely connected nodes tend to close.
  - Draw  $G$  with as few crossings as possible.

Let's take an example



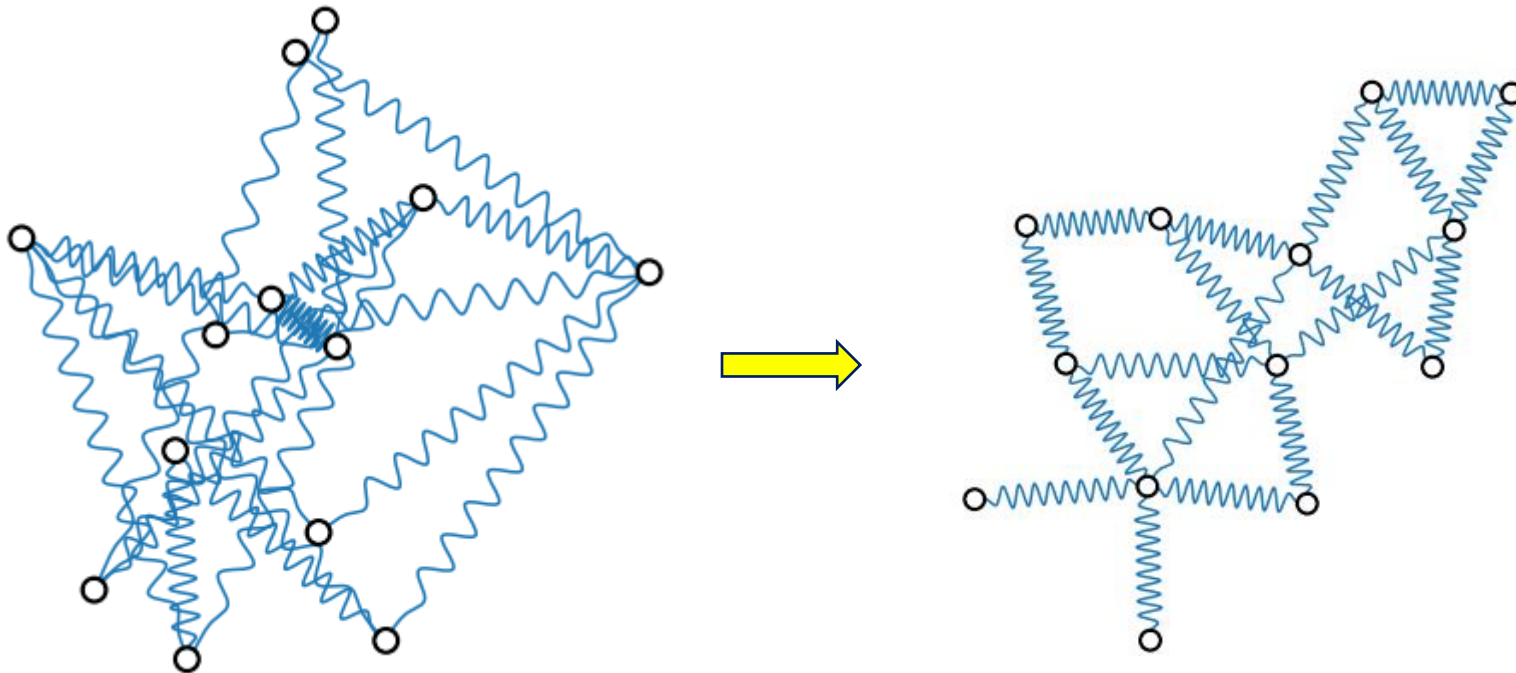
- **Input:** Graph  $G = (V, E)$ , required edge length  $l(e)$ ,  $\forall e \in E$
- **Output:** Drawing of  $G$  which realizes all the edge lengths



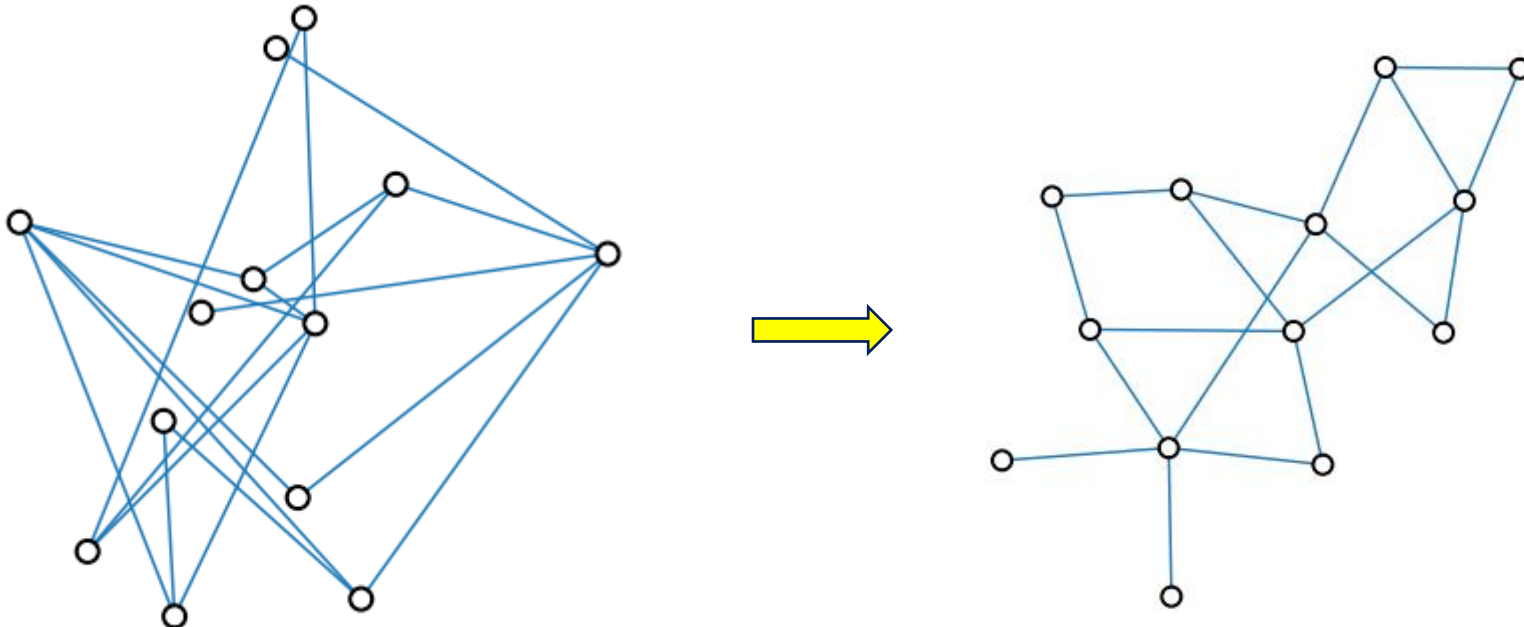
- **NP-hard problem** for:
  - Uniform edge lengths in any dimension.
  - Uniform edge lengths in planar drawing.



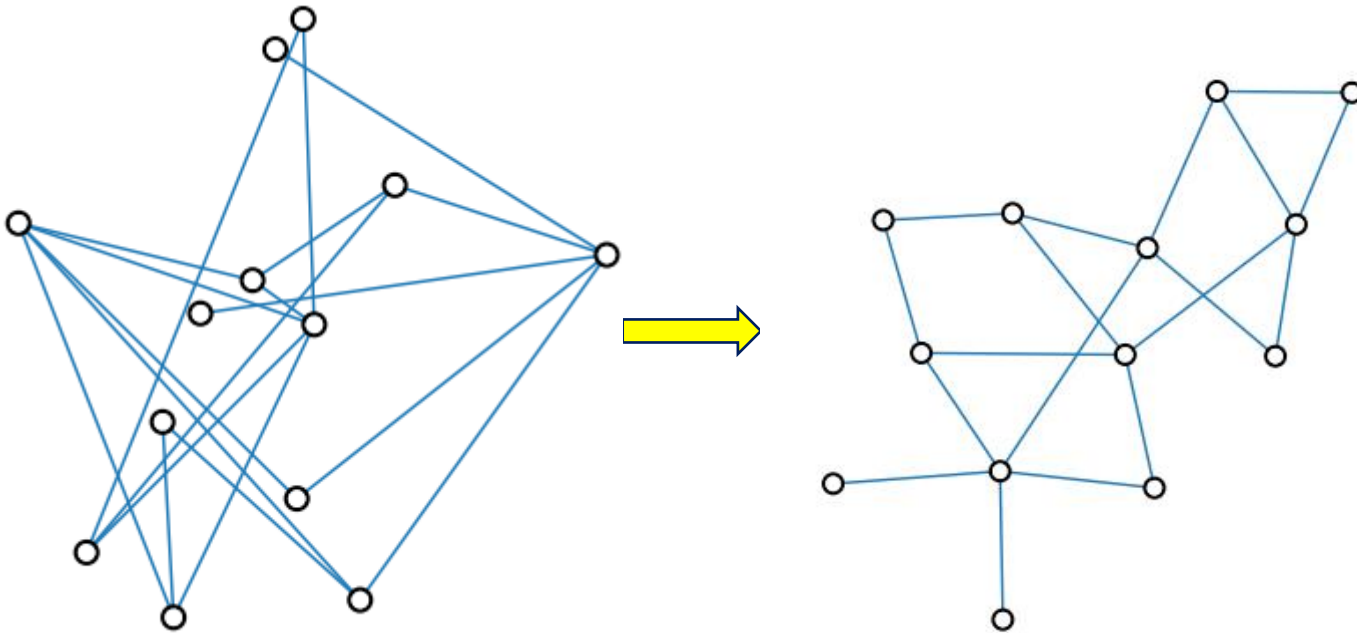
- To embed a graph, we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system.
  - The nodes are placed in some initial layout.
  - The spring forces on the rings move the system to a minimal energy state.



- To embed a graph, we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system.
  - The nodes are placed in some initial layout.
  - The spring forces on the rings move the system to a minimal energy state.



- To embed a graph, we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system.
  - The nodes are placed in some initial layout.
  - The spring forces on the rings move the system to a minimal energy state.



- Adjacent nodes  $u$  and  $v$ :  $f_{spring}$



- Repulsive forces:  
non-adjacent nodes  $u$  and  $v$ :  $f_{rep}$



- Repulsive force between two non-adjacent node pairs  $v_i$  and  $v_j$ :

$$f_{rep}(p_i, p_j) = \frac{c_{rep}}{\|p_i - p_j\|^2} \cdot p_i \vec{p}_j$$

- Attractive force between two adjacent vertices  $v_i$  and  $v_j$ :

$$f_{spring}(p_i, p_j) = c_{spring} \log \frac{\|p_i - p_j\|}{l} \cdot p_i \vec{p}_j$$

- Resulting displacement vector for node  $v_i$

$$F_i(t) \leftarrow \sum_{(v_i, v_j) \notin E} f_{rep}(p_j, p_i) + \sum_{(v_i, v_j) \in E} f_{spring}(p_j, p_i)$$

Where:

- $l = l(e)$ : the ideal spring length of edge  $e$ .
- $\|p_i - p_j\|$ : Distance between  $v_i$  and  $v_j$ .
- $p_i \vec{p}_j$ : unit vector pointing from  $v_i$  to  $v_j$ .
- $c_{rep}$ : repulsion constant (e.g. 1.0).
- $c_{spring}$ : spring constant (e.g. 2.0)

[Fruchterman, Reingold 1991] Graph drawing by force-directed placement

Initial layout with random positions of nodes in the layout

---

**Algorithm 1:** SpringEmbedder

---

$G = (V, E)$   $p = (p_i)$   $v_i \in V, \epsilon > 0, K \in N$

---

**Input:**  $p$ : initial layout,  $\epsilon$ : threshold

**Output:**  $p$ : is end layout

1

2

3

4

5

6

7

8 Return  $p$

---

End layout

- Spring forces:  
Adjacent nodes  $u$  and  $v$ :  $f_{spring}$
- Repulsive forces:  
non-adjacent nodes  $u$  and  $v$ :  $f_{rep}$

Initial layout with random positions of nodes in the layout

---

**Algorithm 1:** SpringEmbedder

---

$G = (V, E)$   $p = (p_i)$   $v_i \in V, \epsilon > 0, K \in N$

---

**Input:**  $p$ : initial layout,  $\epsilon$ : threshold

**Output:**  $p$ : is end layout

1  $t \leftarrow 1$

2

3

4

5

6

7

8 Return  $p$

---

End layout

- Spring forces:  
Adjacent nodes  $u$  and  $v$ :  $f_{spring}$
- Repulsive forces:  
non-adjacent nodes  $u$  and  $v$ :  $f_{rep}$

Initial layout with random positions of nodes in the layout

## Algorithm 1: SpringEmbedder

$G = (V, E)$   $p = (p_i)$   $v_i \in V, \epsilon > 0, K \in \mathbb{N}$

**Input:**  $p$ : initial layout,  $\epsilon$ : threshold

**Output:**  $p$ : is end layout

```

1  $t \leftarrow 1$ 
2 while  $t < K$  and  $\max_{v_i \in V} \|F_i(t)\| > \epsilon$  do
3   for  $v \in V$  do
4      $F_i(t) \leftarrow \sum_{(v_i, v_j) \notin E} f_{rep}(p_j, p_i) + \sum_{(v_i, v_j) \in E} f_{spring}(p_j, p_i)$ 
5   for  $v \in V$  do
6      $p_i \leftarrow p_i + \delta(t) \cdot F_i(t)$ 
7    $t \leftarrow t + 1$ 
8 Return  $p$ 
    
```

Update new location of node

cooling factor

End layout

- Spring forces:  
Adjacent nodes  $u$  and  $v$ :  $f_{spring}$
- Repulsive forces:  
non-adjacent nodes  $u$  and  $v$ :  $f_{rep}$

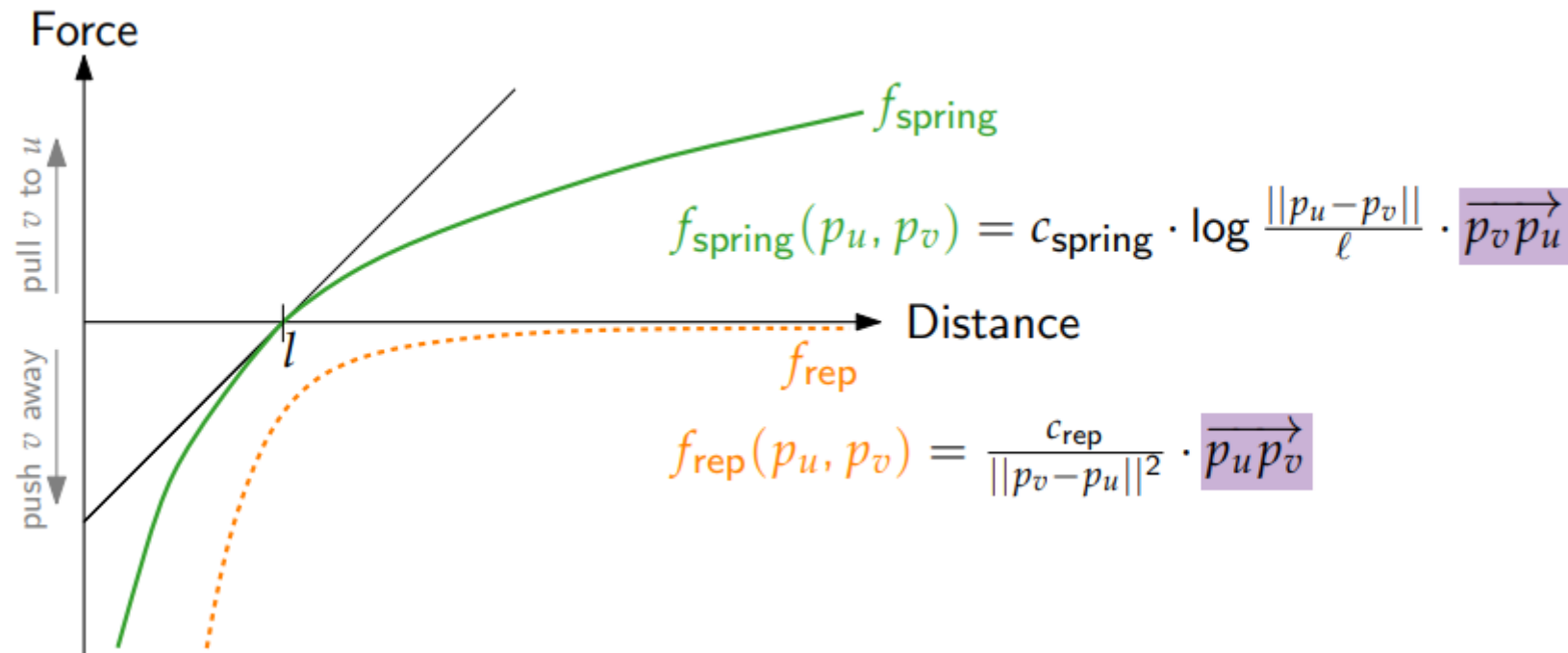
Where:

- $l = l(e)$ : the ideal spring length of edge  $e$ .
- $\|p_i - p_j\|$ : Distance between  $v_i$  and  $v_j$ .
- $p_i \vec{p}_j$ : unit vector pointing from  $v_i$  to  $v_j$ .
- $c_{rep}$ : repulsion constant (e.g. 1.0).
- $c_{spring}$ : spring constant (e.g. 2.0)

$$f_{rep}(p_i, p_j) = \frac{c_{rep}}{\|p_i - p_j\|^2} \cdot p_i \vec{p}_j$$

$$f_{spring}(p_i, p_j) = c_{spring} \log \frac{\|p_i - p_j\|}{l} \cdot p_i \vec{p}_j$$

- Spring forces ( $f_{spring}$ ): pull node  $v$  close to node  $u$  ( $u$  and  $v$  are adjacent)
- Repulsive forces ( $f_{rep}$ ): push node  $v$  far away node  $u$  ( $u$  and  $v$  are non-adjacent)





- Advantages.
  - Simple algorithm.
  - Good results for small and medium-sized graphs.
  - Good representation of symmetry and structure.
- Disadvantages.
  - System is not stable at the end.
  - Converging to local minimal.

- Repulsive force between **all** vertex pairs  $v_i$  and  $v_j$ :

$$f_{rep}(p_i, p_j) = \frac{l}{||p_i - p_j||^2} \cdot p_i \vec{p}_j$$

- Attractive force between two adjacent vertices  $v_i$  and  $v_j$ :

$$f_{attactive}(p_i, p_j) = \frac{||p_i - p_j||^2}{l} \cdot p_i \vec{p}_j$$

- Resulting force between adjacent vertices  $v_i$  and  $v_j$ :

$$f_{spring}(p_i, p_j) = f_{rep}(p_i, p_j) + f_{attactive}(p_i, p_j)$$

---

**Algorithm 1:** SpringEmbedder

$G = (V, E), p = (p_i), v_i \in V, \epsilon > 0, K \in \mathbb{N}$

---

**Input:**  $p$ : initial layout,  $\epsilon$ : threshold

**Output:**  $p$ : is end layout

```

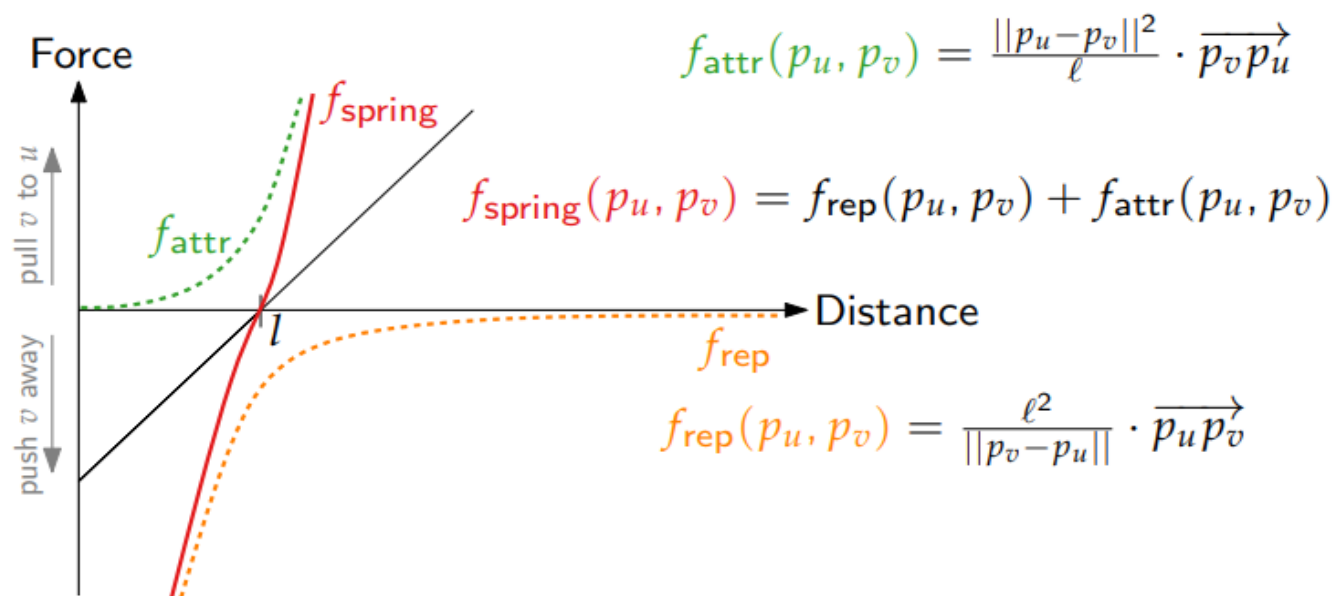
1  $t \leftarrow 1$ 
2 while  $t < K$  and  $\text{MAX}_{v_i \in V} ||F_i(t)|| > \epsilon$  do
3   for  $v \in V$  do
4      $F_i(t) \leftarrow \sum_{(v_i, v_j) \notin E} f_{rep}(p_j, p_i) + \sum_{(v_i, v_j) \in E} f_{spring}(p_j, p_i)$ 
5   for  $v \in V$  do
6      $p_i \leftarrow p_i + \delta(t) \cdot F_i(t)$ 
7    $t \leftarrow t + 1$ 
8 Return  $p$ 

```

---

There are three forces:

- Spring forces ( $f_{spring}$ ): pull node  $v$  close to node  $u$  ( $u$  and  $v$  are adjacent)
- Attractive force between two adjacent nodes  $v_i$  and  $v_j$  ( $f_{attr}$ ): pull node  $v$  close to node  $u$  ( $u$  and  $v$  are adjacent)
- Repulsive forces ( $f_{rep}$ ): push node  $v$  far away node  $u$  ( $u$  and  $v$  are non-adjacent)



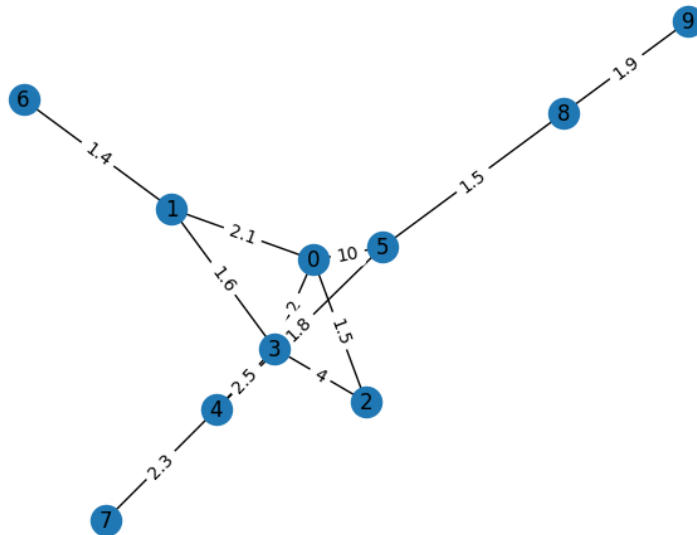
## ➤ Spring layout

```
# Instantiate the graph
G = nx.Graph()

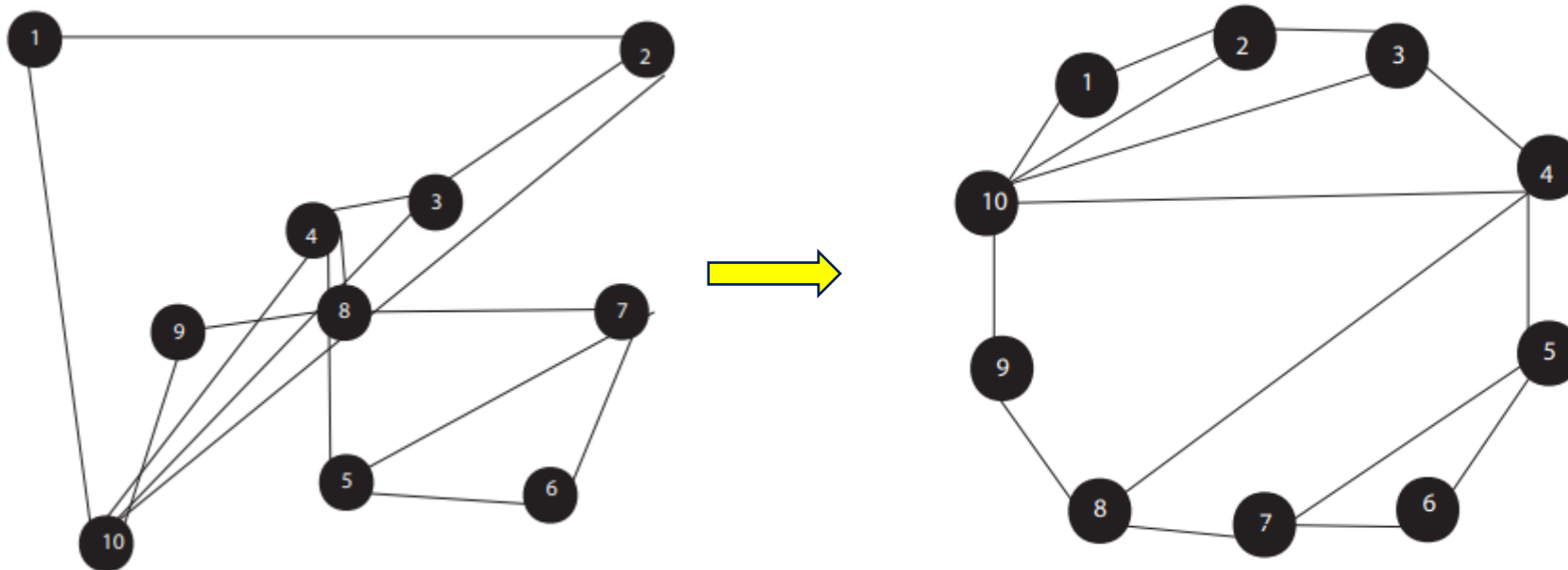
edges = [(0, 1, 2.1), (0, 2, 1.5), (0, 3, 2), (0, 5, 10), (1, 3, 1.6), (1, 6, 1.4),
         (2, 3, 4), (3, 4, 2.5), (4, 5, 1.8), (4, 7, 2.3), (5, 8, 1.5), (8, 9, 1.9)]
# add node/edge pairs
G.add_weighted_edges_from(edges)

pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels = True)

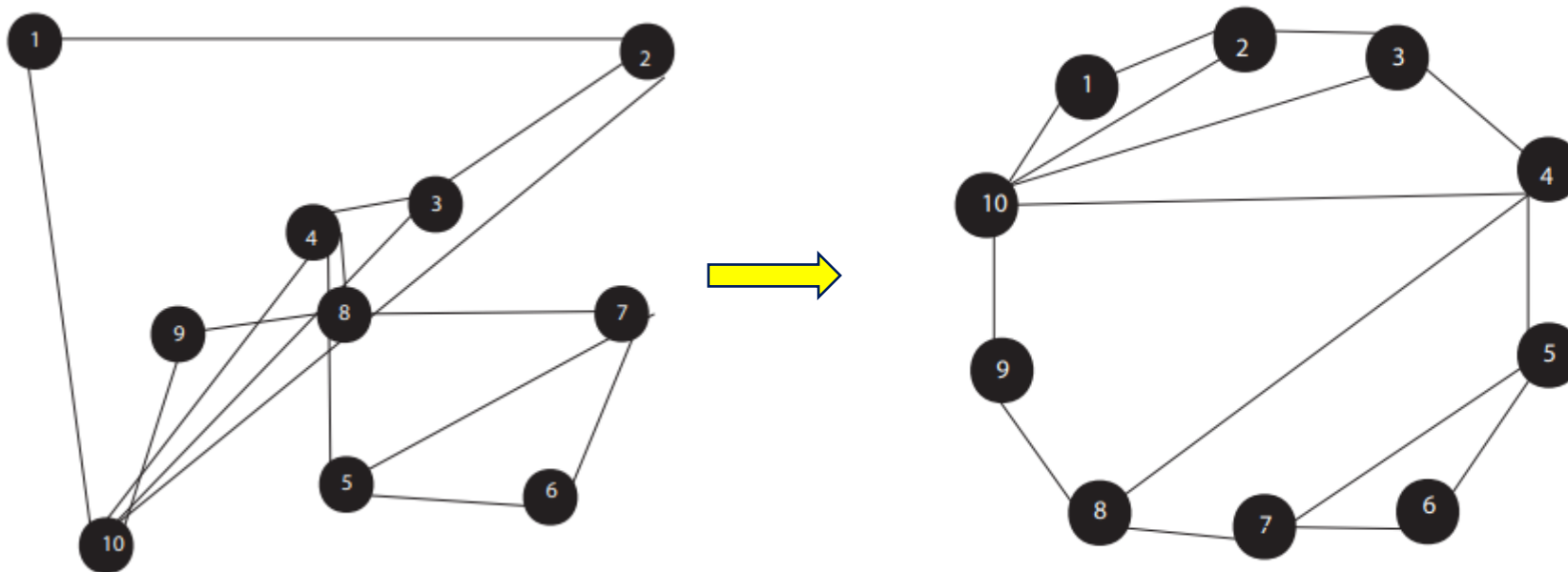
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
```



- **Input:** Graph  $G = (V, E)$
- **Output:** Creating clear and readable drawings of graph  $G$ .



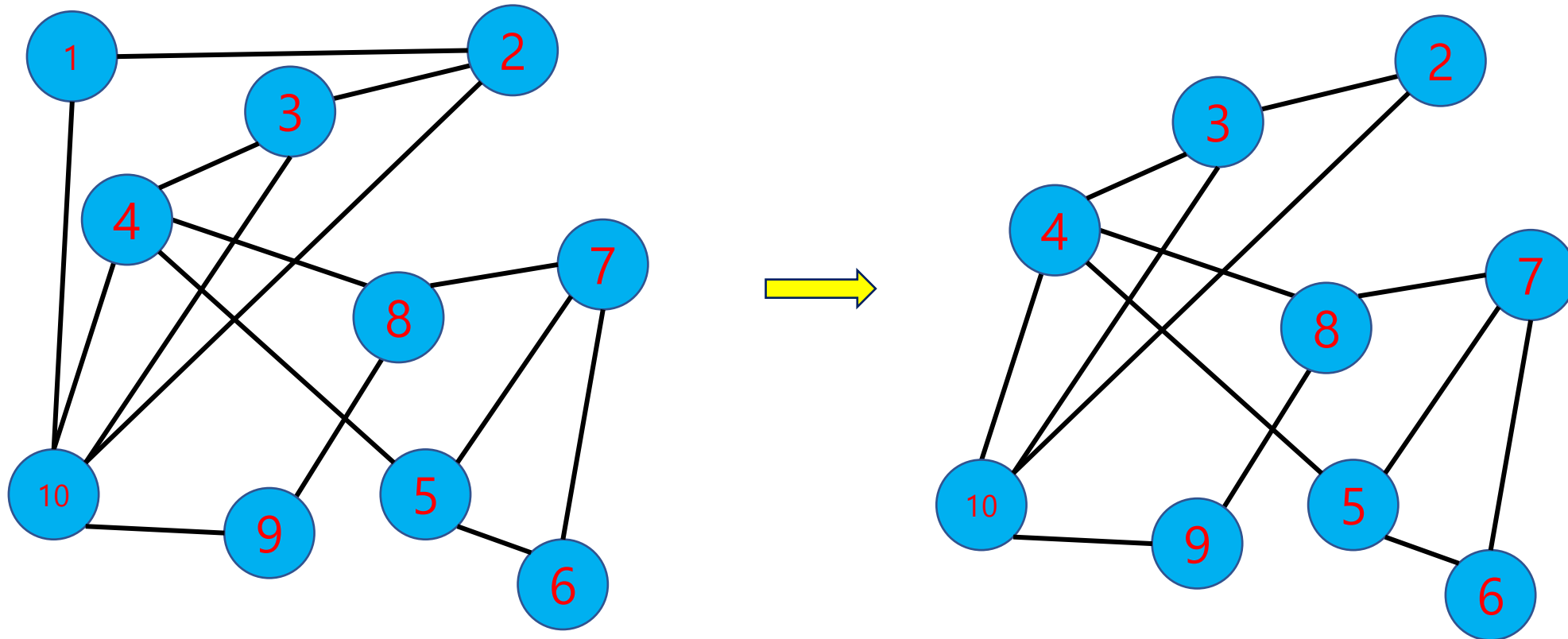
- Input: A biconnected graph,  $G = (V, E)$ .
- Output: A circular drawing  $\Gamma$  of  $G$  such that each node in  $V$  lies on the periphery of a single embedding circle.



- In circular graphs, close nodes should not be connected:
  - Idea: Finding and store nodes that have two non-connected neighbours by using BFS algorithm.
  - Implement:
    - Starting at a random node, store nodes that do not have non-connected neighbours in a stack.
    - Restore the graph to generate a circle graph

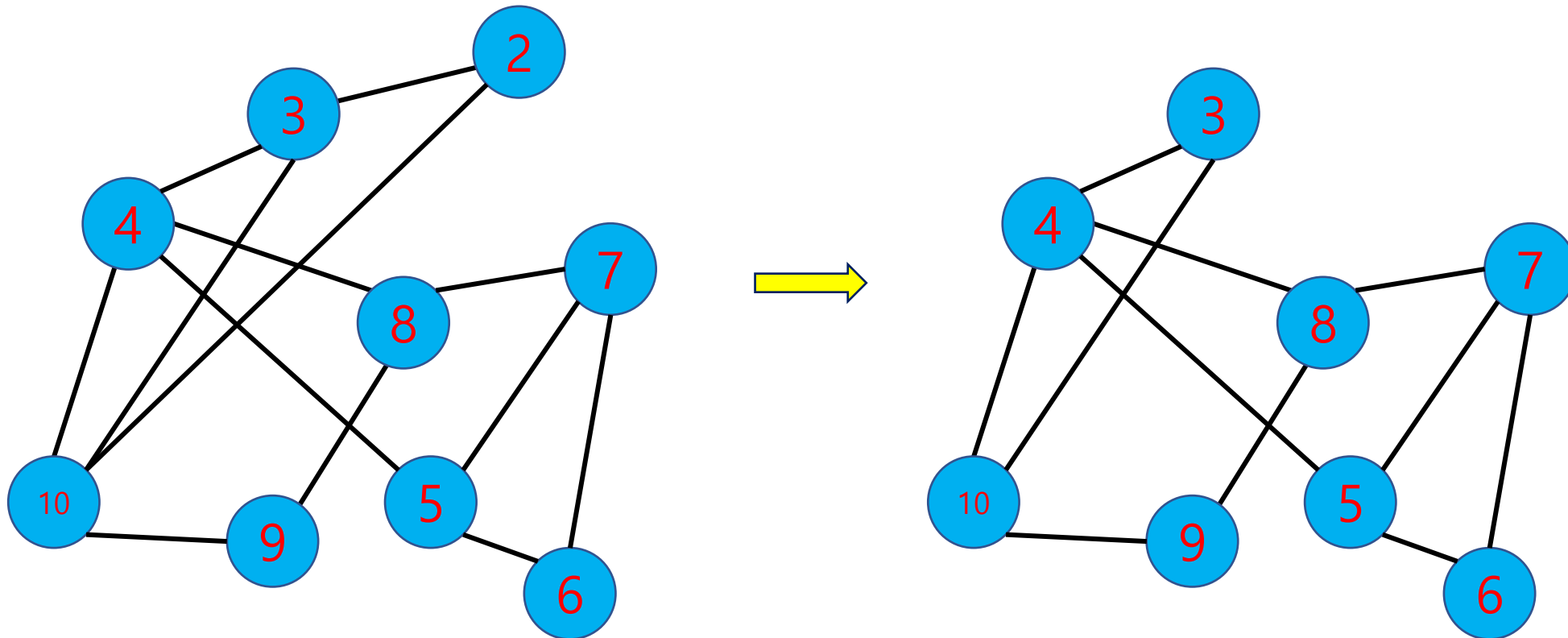
1. Bucket sort the nodes by ascending degree into a table  $T$ .
2. Set  $counter$  to 1.
3. While  $counter \leq n - 3$ 
  4. If a wave front node  $u$  has lowest degree then  $currentNode = u$ .
  5. Else If a wave center node  $v$  has lowest degree then  
 $currentNode = v$ .
  6. Else set  $currentNode$  to be some node with lowest degree.
  7. Visit the adjacent nodes consecutively. For each two nodes,
    8. If a pair edge exists place the edge into  $removalList$ .
    9. Else place a triangulation edge between the current pair of neighbors and also into  $removalList$ .
  10. Update the location of  $currentNode$ 's neighbors in  $T$ .
  11. Remove  $currentNode$  and incident edges from  $G$ .
  12. Increment  $counter$  by 1.
13. Restore  $G$  to its original topology.
14. Remove the edges in  $removalList$  from  $G$ .
15. Perform a DFS (or a longest path heuristic) on  $G$ .
16. Place the resulting longest path onto the embedding circle.
17. If there are any nodes which have not been placed then place the remaining nodes into the embedding order with the following priority:
  - (i) between two neighbors, (ii) next to one neighbor, (iii) next to zero neighbors.

- First, we choose randomly Node 1.
- We check for edge(2,10), which exists. We store it and remove Node 1.

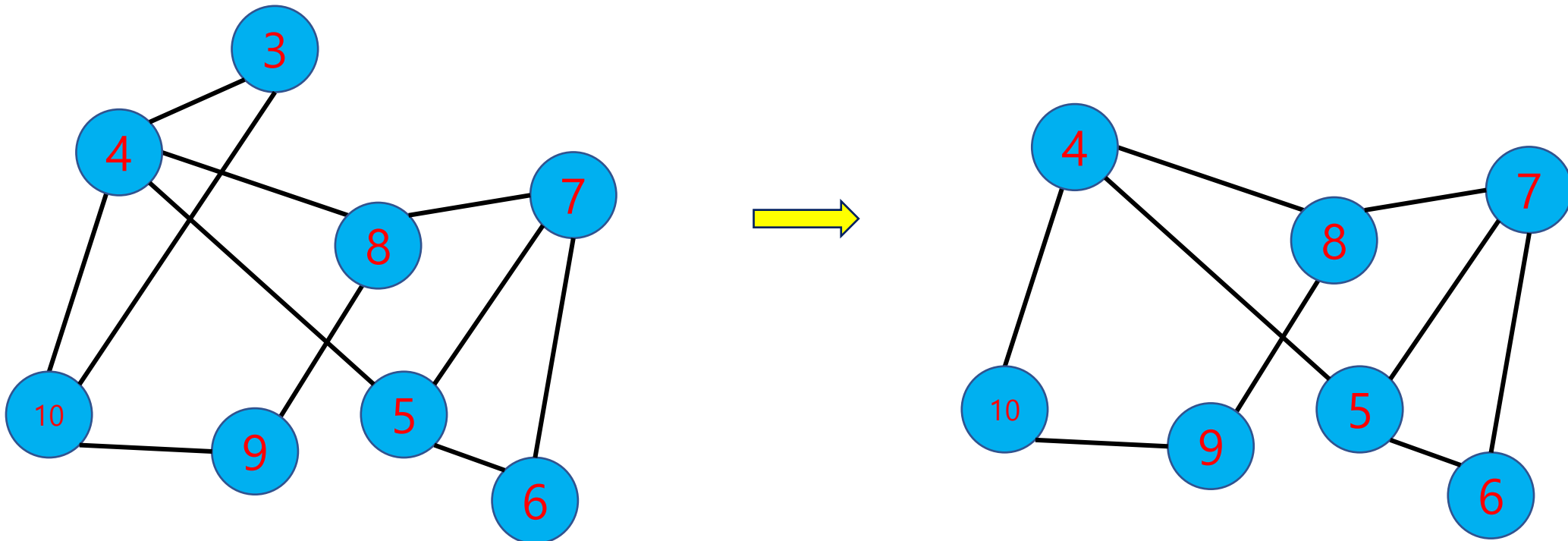




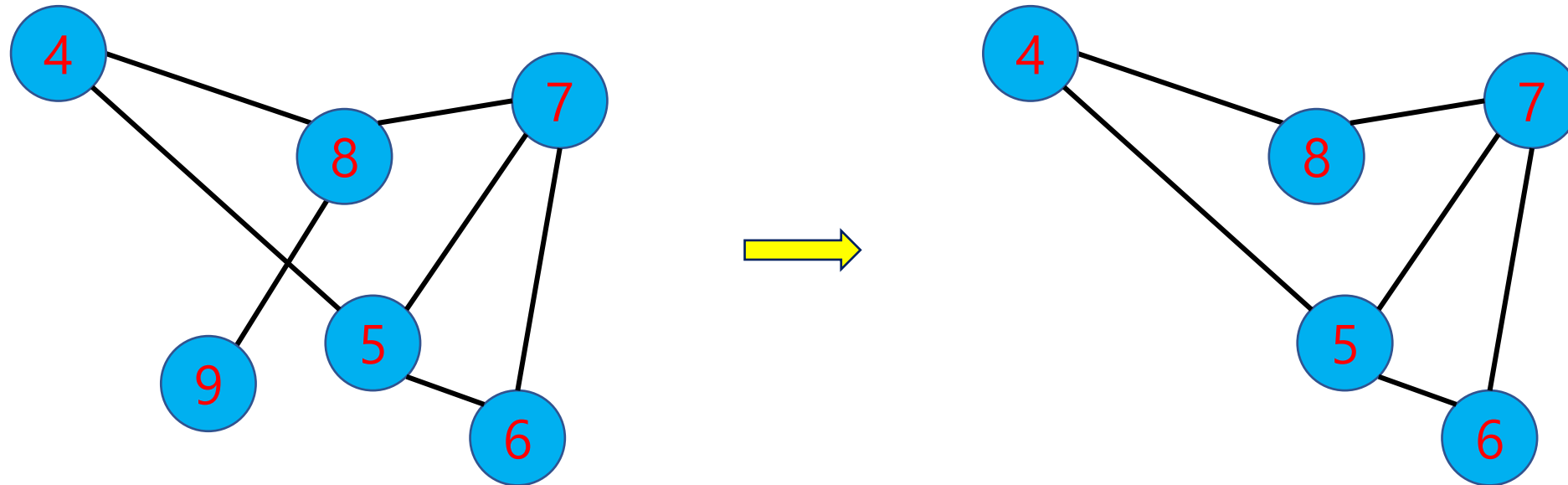
- Next, we choose a lowest-degree neighbor of the removed Node 1, which is 2.
- Check for edge (3,10) which exists. We store it and remove Node 2.



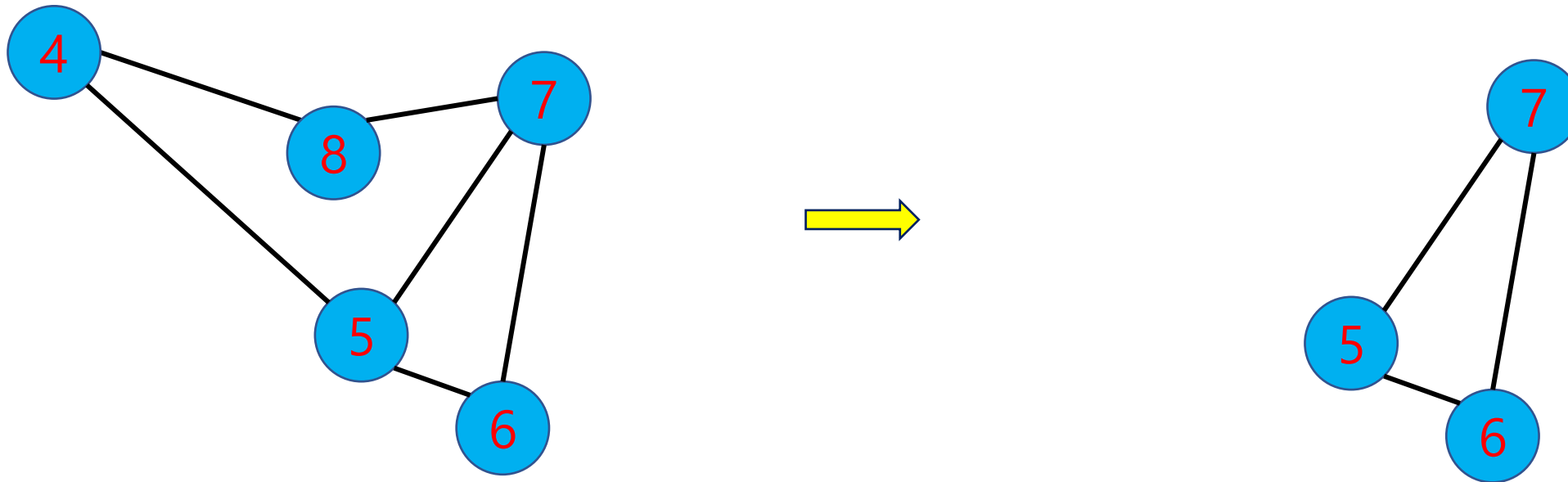
- Next we select a lowest degree neighbor of Node 2. This is Node 3. We check for edge (4,10).
- It exists so we store it and remove Node 3.



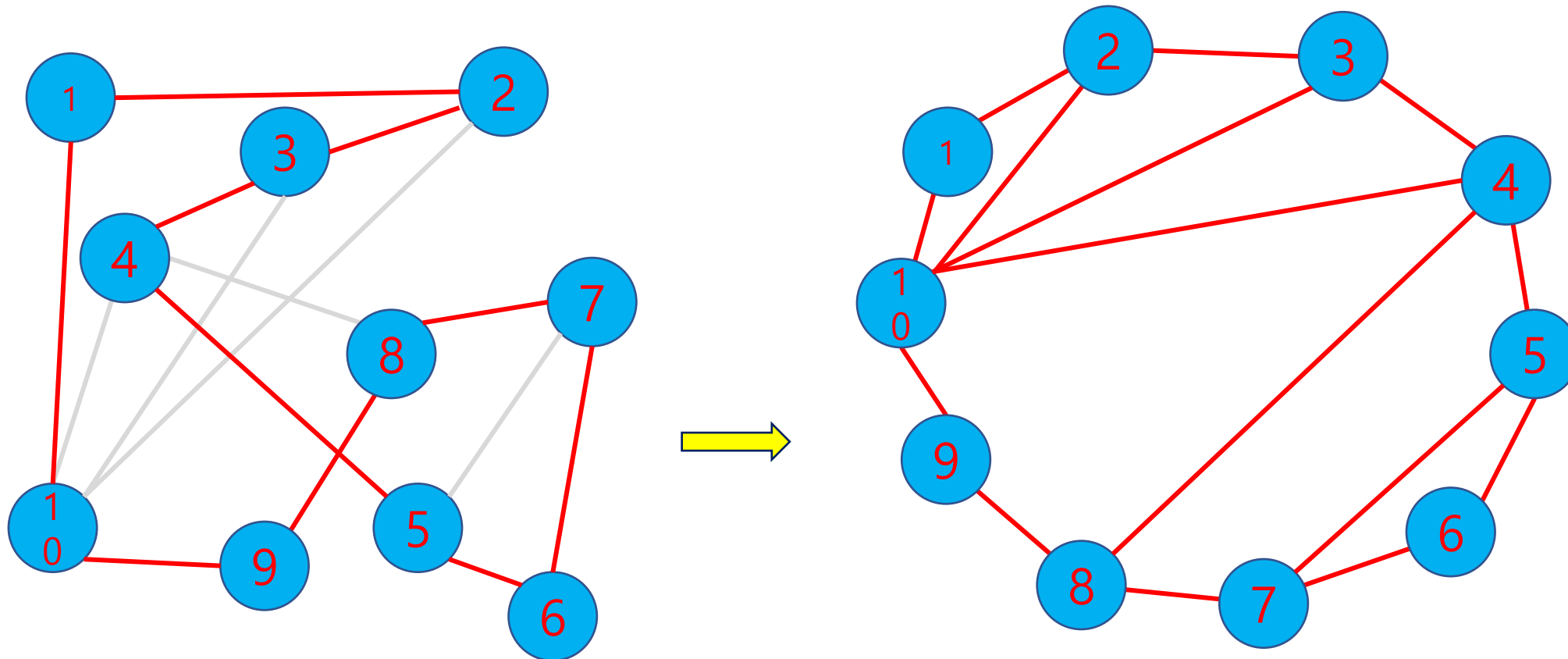
- Similarly we can select Node 10 and check for edge (4,9). It does not exist. So we add edge (4,9) which is a triangulation edge, store it and remove Node 10.
- We continue choosing Node 9, and check for edge (4,8). It exists so we store it and remove Node 9. Next, for Node 8 we check for edge(4,7) which does not exist. We add it to the graph and store it. After this, we remove Node 8.



- In the same way, we select vertex 4 and check for edge (5,7), which exists. So we mark.
- Now we have only three vertices left, so this phase of the algorithm is completed.



- Now we restore the graph and remove all stored edges.
- Since the graph is outerplanar, we have the Hamilton circle left.



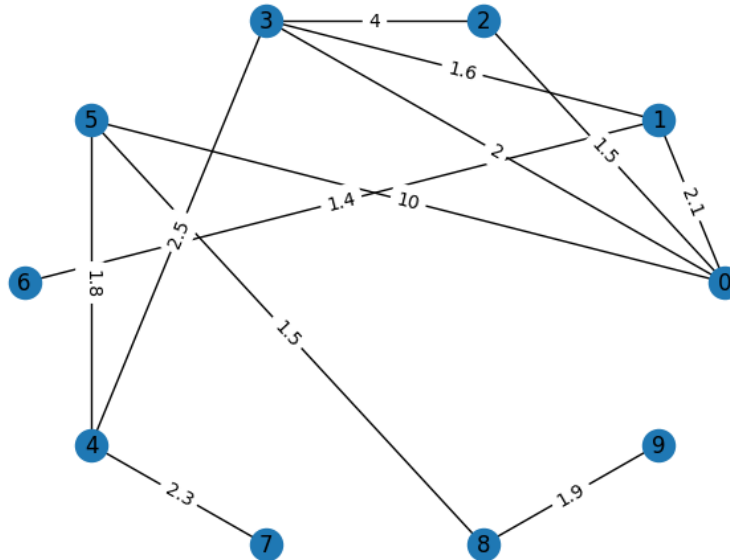
## ➤ Circular layout

```
# Instantiate the graph
G = nx.Graph()

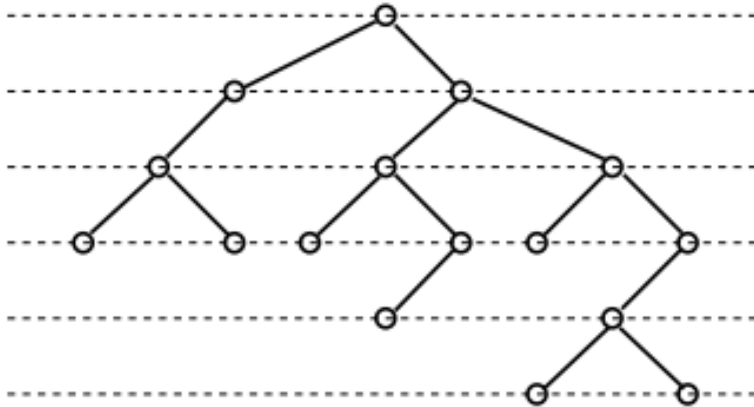
edges = [(0, 1, 2.1), (0, 2, 1.5), (0, 3, 2), (0, 5, 10), (1, 3, 1.6), (1, 6, 1.4),
         (2, 3, 4), (3, 4, 2.5), (4, 5, 1.8), (4, 7, 2.3), (5, 8, 1.5), (8, 9, 1.9)]
# add node/edge pairs
G.add_weighted_edges_from(edges)

pos = nx.circular_layout(G)
nx.draw(G, pos, with_labels = True)

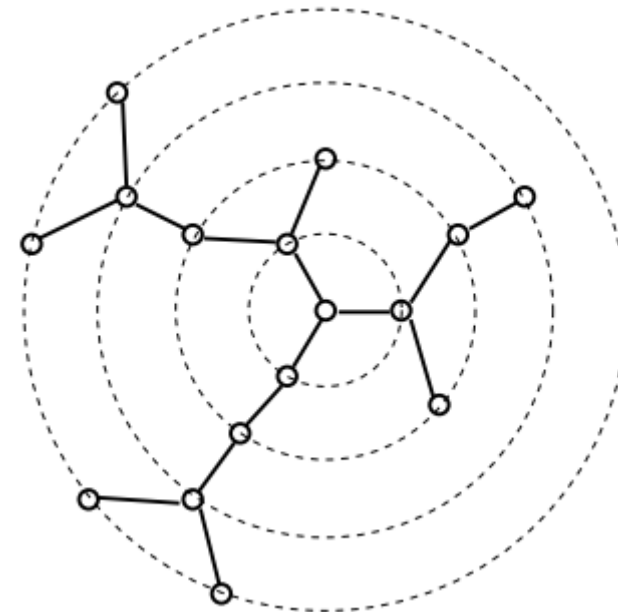
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
```



- Trees:
  - Requirements:
    - No two edges cross
    - A child should be placed below its parent in the y-direction.
    - Strongly order-preserving drawings

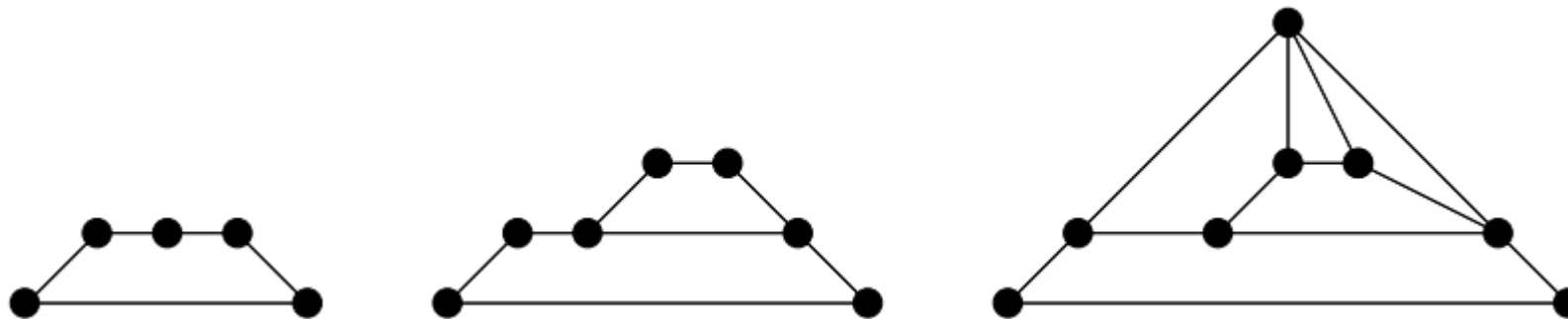


a layered tree drawing.



a radial tree drawing.

- Given an input graph  $G = (V, E)$ :
- Kant used the canonical ordering approach to develop straight-line algorithm.
- The algorithm aims to form a chain and give them the same  $y$ -coordinate



An example for the straight-line algorithm of Kant.

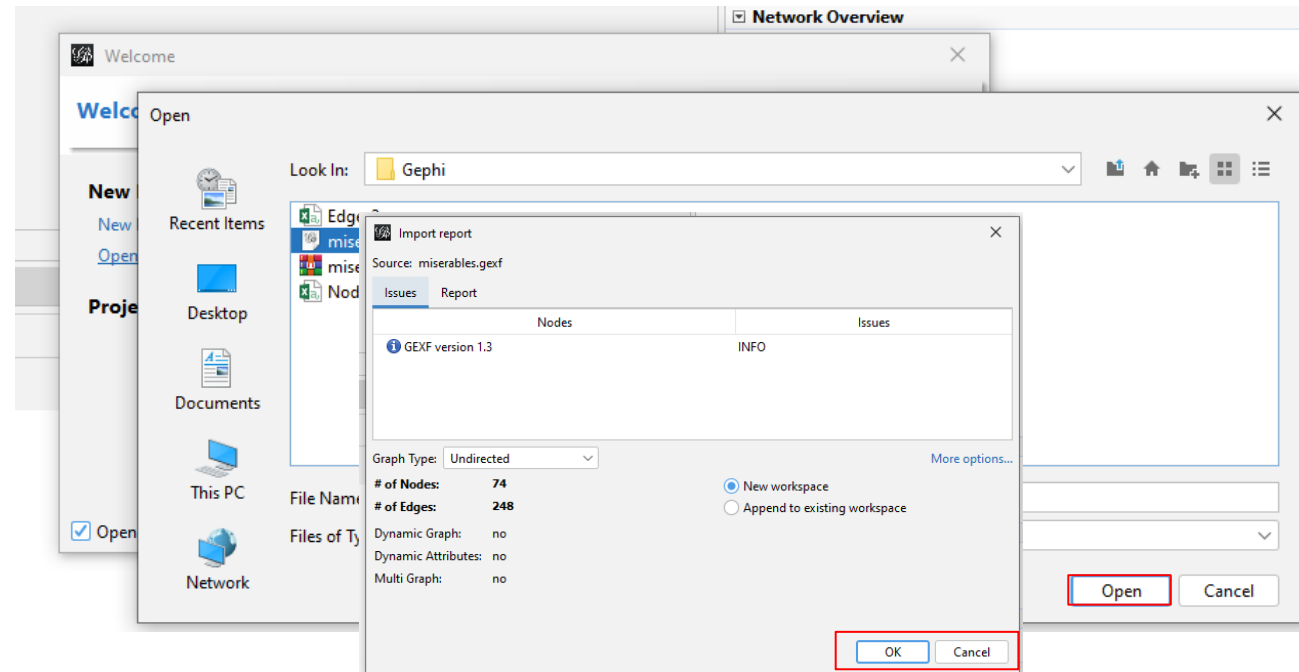
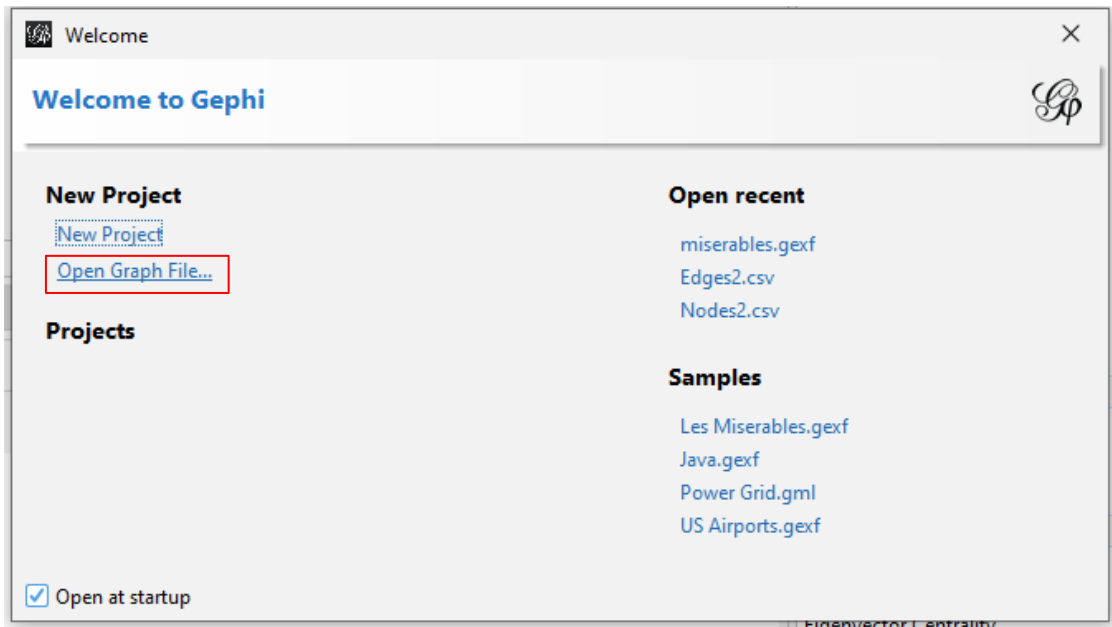


- There are several open source tools for network analysis:
  - NetworkX
  - iGraph packages in R
  - Gephi
  - Cytoscape

- An open-source visualization exploration software without having coding skills.
- Functions:
  - Interaction with the final representation
  - Manipulation with structures
  - Appearance properties
  - Understand patterns in visualization.

- Applications of Gephi:
  - Exploratory Data Analysis
  - Link Analysis
  - Social Network Analysis
  - Biological Network Analysis
  - Poster Creation
- Different layouts:
  - Circular Layout
  - Noverlap Layout
  - Expansion...

- Prepare:
  - Sample graph: `miserables.gexf` (download in class github)
  - Open Gephi.
  - On the Welcome screen that appears, click on Open Graph File.
  - Open `miserables.gexf` and click OK



## ➤ Gephi's interface:

The screenshot shows the Gephi 0.10.1 interface with a network graph visualization in the center. The interface is divided into several panels and toolbars. Red arrows point from numbered callouts to specific parts of the interface:

- 1. Overview:** where we can explore the graph visually (points to the Overview tab).
- 2. Data Laboratory:** provides an "Excel" table view of the data in network (points to the Data Laboratory tab).
- 3. Preview:** where we polish the visualization before exporting it as a picture or pdf (points to the Preview tab).
- 4. "Filters":** where we can hide different parts of the network under a variety of conditions (points to the Filters panel on the right).
- 5. "Statistics":** where we can compute metrics on the network (points to the Statistics panel on the right).
- 6. "Appearance":** where we can change colors and sizes in interesting ways (points to the Appearance panel on the left).
- 7. "Layouts":** where we can apply automated procedures to change the position of the network (points to the Layout panel on the left).
- 8. A series of icons to add / colorize nodes and links manually, by clicking on them** (points to the toolbar icons).
- 9. Options and sliders to change the size of all nodes, links, or labels** (points to the bottom toolbar).

## ➤ Gephi's layout:

The image shows two screenshots of the Gephi 0.10.1 interface, illustrating the steps to visualize a graph layout.

**Step 1:** The 'Layout' panel is open, showing a list of layout algorithms. The 'Fruchterman Reingold' algorithm is selected.

**Step 2:** The 'Fruchterman Reingold' algorithm is chosen from the list of layouts.

**Step 3:** The 'Run' button is clicked to execute the layout algorithm.

**Step 4:** The 'Stop' button is clicked to freeze the graph.

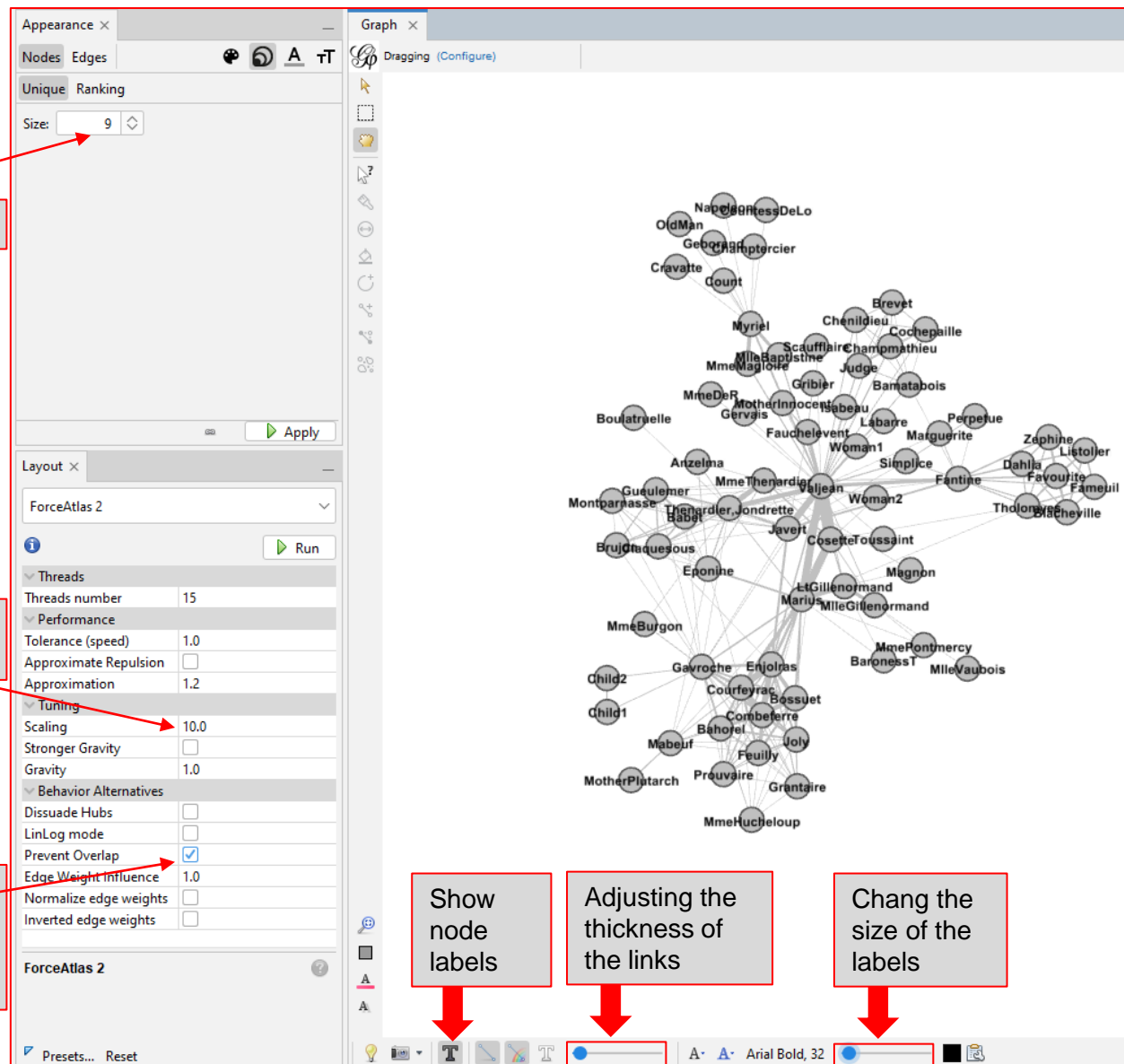
The final result is a network graph visualization showing a complex, interconnected structure of nodes and edges.

## ➤ Force atlas 2 layout:

1. Size: change node size.

2. Scaling: a parameter to control how “wide” the graph will be.

3. Prevent overlap: a parameter to avoid that nodes are on top of each other. To make it easier to read the network. Check the box.



Force Atlas 2 is a layout which:

1. brings together nodes which are connected

2. spreads apart unconnected nodes

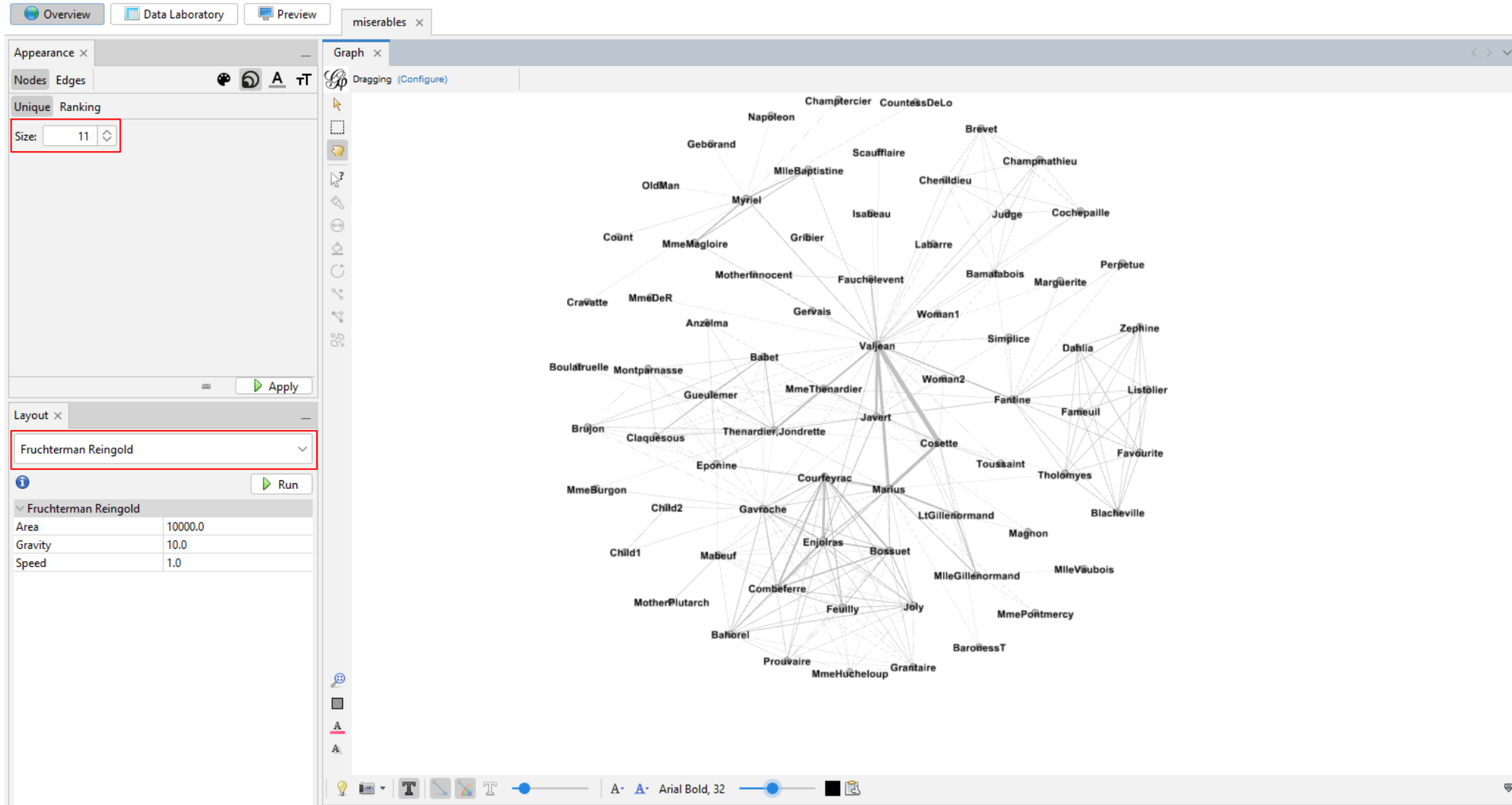
As an effect, we can easily detect communities of nodes

Show node labels

Adjusting the thickness of the links

Change the size of the labels

- Fruchterman and Reingold relies on spring layout





- Circular layout: to use this layout, you have to install new plugin

1. Click on "Tools"

2. Select "Plugins"

3. Search plugin with keyword "circular"

4. Check on the box

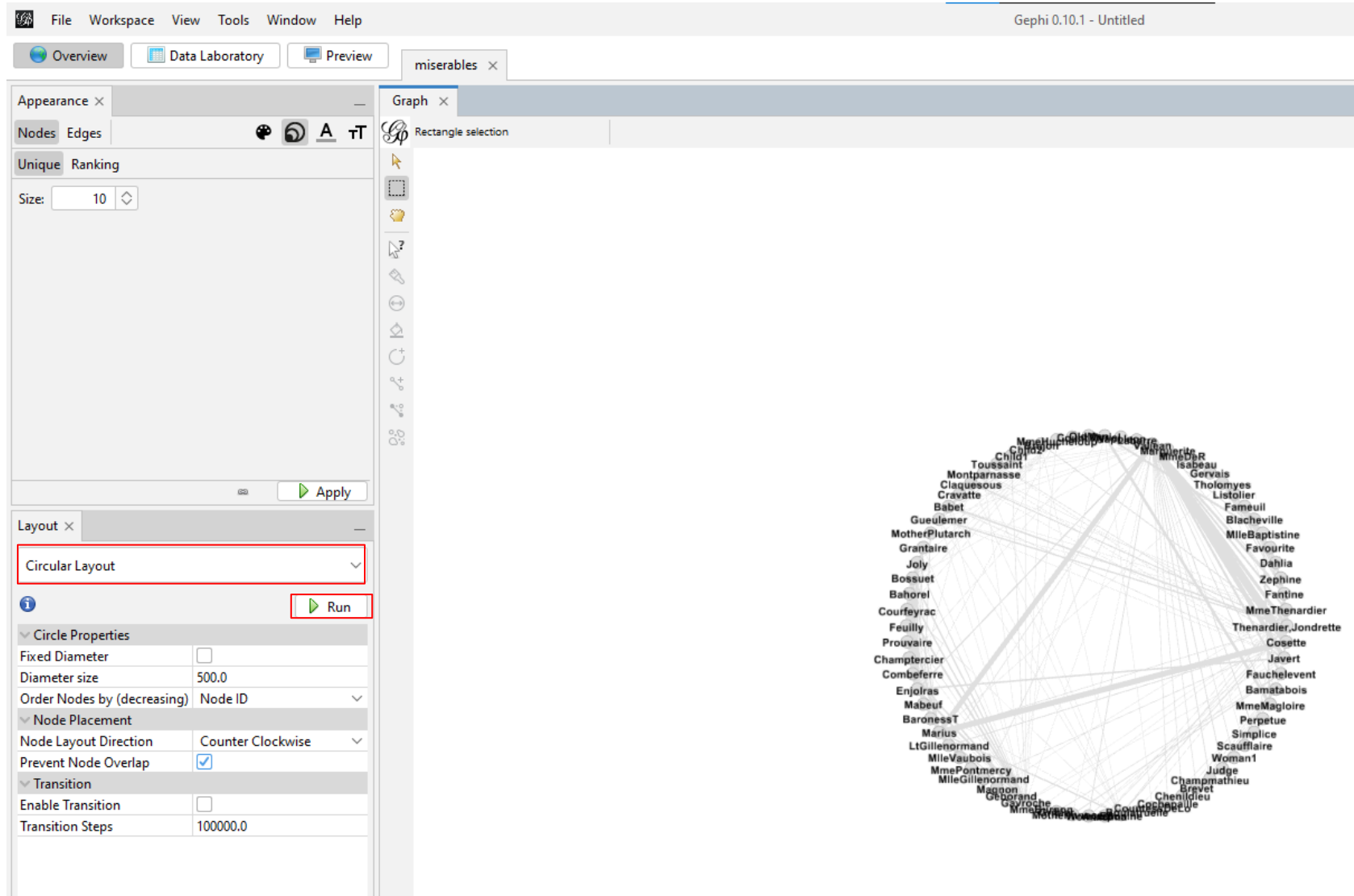
5. Click "Next"

6. Accept the terms of plugin

7. Click "Install"

8. Click "Finish" with restart to apply plugin. Don't forget to save your project before restarting!

➤ Circular layout:



## ➤ Preview graph

1. Click "Preview"

2. Select "Presets" to display you graph

3. Custom graph style

4. Click on "Refresh" to apply

Export image

Zoom graph

The screenshot shows the Gephi 0.10.1 interface. The 'Preview' tab is selected, displaying a network graph of characters from Les Misérables. The 'Preview Settings' panel on the left is open, showing various options for customizing the graph's appearance. Red boxes and arrows highlight the following steps:

- 1. Click "Preview": The 'Preview' tab in the top toolbar is highlighted.
- 2. Select "Presets" to display you graph: The 'Presets' dropdown menu in the 'Preview Settings' panel is highlighted.
- 3. Custom graph style: The 'Nodes' and 'Edges' sections in the 'Preview Settings' panel are highlighted, showing various customization options like border color, font, and edge thickness.
- 4. Click on "Refresh" to apply: The 'Refresh' button at the bottom of the 'Preview Settings' panel is highlighted.
- Export image: The 'Export' button at the bottom left of the 'Preview' window is highlighted.
- Zoom graph: The 'Reset zoom' button at the bottom right of the 'Preview' window is highlighted.

## ➤ Switching the view to the data laboratory:

The screenshot displays the Gephi Data Laboratory interface. At the top, there are tabs for 'Overview', 'Data Laboratory', and 'Preview'. The 'Data Laboratory' tab is active. Below the tabs, there is a 'Data Table' window showing a table of node data. The table has columns for 'Label', 'Interval', and 'Gender'. The 'Label' column contains names like Myriel, Napoleon, MlleBaptistine, etc. The 'Interval' column is empty. The 'Gender' column contains 'M' or 'F'. The table is filtered by 'Id'. Annotations with red boxes and arrows point to specific parts of the interface:

- An arrow points to the 'Data Laboratory' tab with the text 'Click on "Data laboratory"'.  
Node labels →
- Node attributes →
- Each row contains one node information →

|    | Label          | Interval | Gender |
|----|----------------|----------|--------|
| 0  | Myriel         |          | M      |
| 1  | Napoleon       |          | M      |
| 2  | MlleBaptistine |          | F      |
| 3  | MmeMagloire    |          | F      |
| 4  | CountessDeLo   |          | F      |
| 5  | Geborand       |          | F      |
| 6  | Champiercier   |          | M      |
| 7  | Cravatte       |          | M      |
| 8  | Count          |          | M      |
| 9  | OldMan         |          | M      |
| 10 | Labarre        |          | M      |
| 11 | Valjean        |          | M      |
| 12 | Marguerite     |          | F      |
| 13 | MmeDeR         |          | F      |
| 14 | Isabeau        |          | M      |
| 15 | Gervais        |          | M      |
| 16 | Tholomyes      |          | M      |
| 17 | Listolier      |          | M      |
| 18 | Fameuil        |          | M      |
| 19 | Blacheville    |          | M      |
| 20 | Favourite      |          | F      |
| 21 | Dahlia         |          | F      |
| 22 | Zephine        |          | F      |
| 23 | Fantine        |          | F      |
| 24 | MmeThenardier  |          | F      |
| 26 | Cosette        |          | F      |
| 27 | Javert         |          | M      |
| 28 | Fauchelevant   |          | M      |
| 29 | Bamatabois     |          | M      |
| 30 | Perpetue       |          | F      |
| 31 | Simplice       |          | F      |
| 32 | Scaufflaire    |          | M      |
| 33 | Woman1         |          | F      |
| 34 | Judge          |          | M      |
| 35 | Champmathieu   |          | M      |
| 36 | Brevet         |          | M      |
| 37 | Chenildieu     |          | M      |
| 38 | Cochepaille    |          | M      |
| 40 | Boulatruelle   |          | M      |
| 41 | Eponine        |          | F      |
| 42 | Anzelma        |          | F      |
| 43 | Woman2         |          | F      |

## ➤ Computing betweenness centrality with Gephi:

1. Click on "Statistics"

2. Run "Network Diameter"

3. Click OK

4. Close

**HTML Report Results:**  
Diameter: 5  
Radius: 3  
Average Path length: 2.587189929655683

**Betweenness Centrality Distribution**

Count

Value

Print Copy Save Close

Graph Distance settings

Distance

The average graph-distance between all pairs of nodes. Connected nodes have graph distance 1. The diameter is the longest graph distance between any two nodes in the network. (i.e. How far apart are the two most distant nodes).

☐ Directed ☒ Undirected ☐ Normalize Centralities in [0,1]

**Betweenness Centrality:** Measures how often a node appears on shortest paths between nodes in the network.

**Closeness Centrality:** The average distance from a given starting node to all other nodes in the network.

**Eccentricity:** The distance from a given starting node to the farthest node from it in the network.

OK Cancel

Context

Nodes: 74  
Edges: 248  
Undirected Graph

Filters Statistics

Settings

Network Overview

Avg. Degree 6.703 Run

Weighted Degree Run

Network Diameter Run

Graph Density Run

HITS Run

PageRank Run

Connected Components Run

Community Detection

Modularity Run

Statistical Inference Run

Node Overview

Avg. Clustering Coefficient Run

Eigenvector Centrality Run

Edge Overview

Avg. Path Length Run

Dynamic

# Nodes Run

# Edges Run

Degree Run

Clustering Coefficient Run

## ➤ View graph attribute: Betweenness Centrality

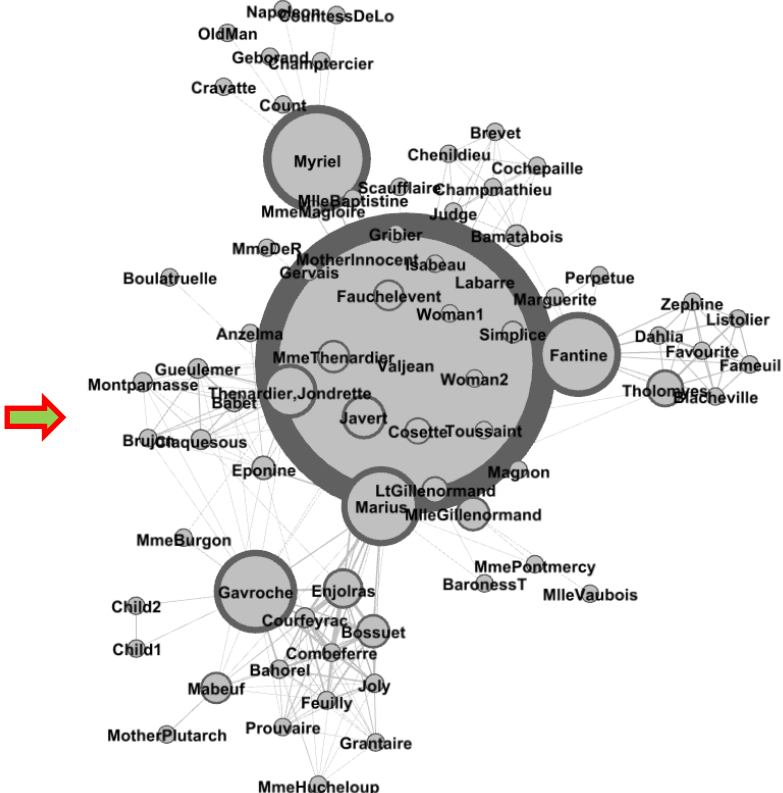
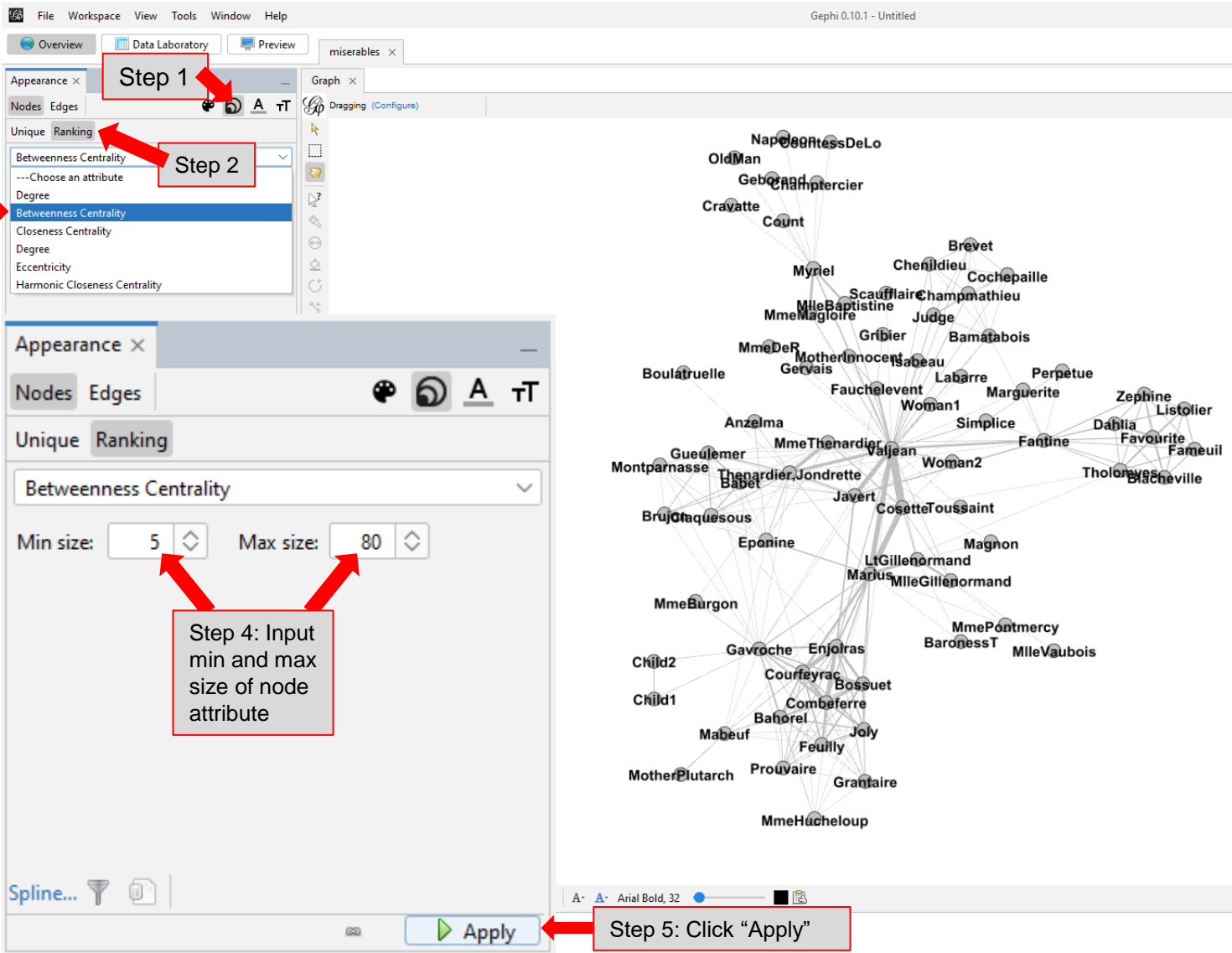
Step 1: Select the 'Data Laboratory' tab in the top menu.

Step 2: In the 'Appearance' panel, under the 'Nodes' tab, select 'Betweenness Centrality' from the 'Ranking' dropdown menu.

Step 3: Choose the attribute 'Betweenness Centrality' from the list.

Step 4: Input the minimum and maximum size of the node attribute. Set 'Min size' to 5 and 'Max size' to 80.

Step 5: Click the 'Apply' button at the bottom of the 'Appearance' panel.







네트워크 과학연구실  
NETWORK SCIENCE LAB



가톨릭대학교  
THE CATHOLIC UNIVERSITY OF KOREA

