

Graph Database and Visualization

Prof. O-Joun Lee

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



네트워크 과학 연구실
NETWORK SCIENCE LAB



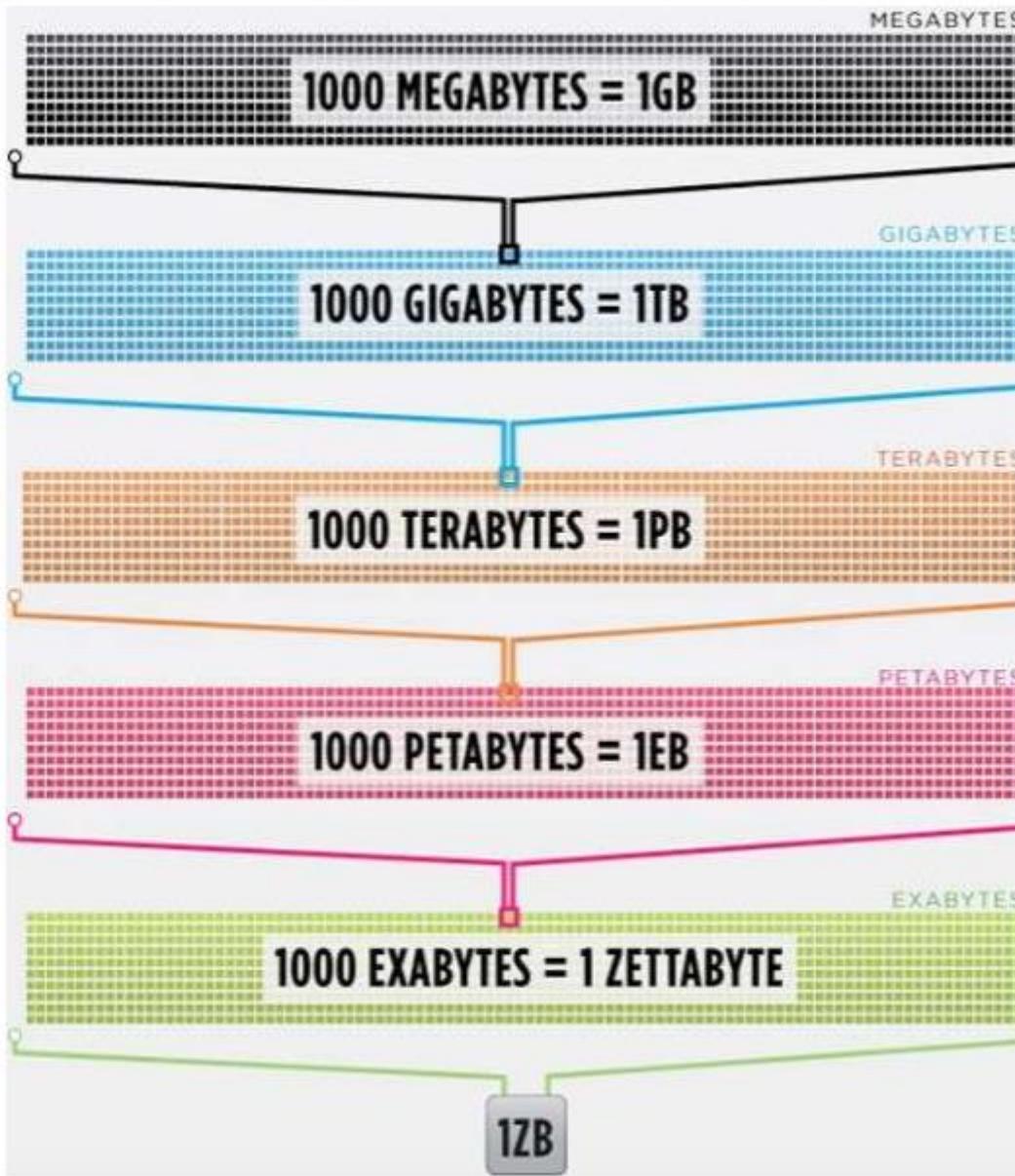
가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA



Contents



- Graph databases and storage systems
 - Trends in graph database
 - Graph database engines and example
- Graph visualization:
 - Visualization techniques:
 - Spring-embedded
 - Circular, etc.
 - Tools for graph exploration and visualization:
 - Gephi, Cytoscape, etc.



➤ Data is getting bigger:

“Every 2 days we create as much information as we did up to 2003”

– Eric Schmidt, Google

Data is more connected

- Text (content)
- HyperText (added pointers)
- RSS (joined those pointers)
- Blogs (added pingbacks)
- Tagging (grouped related data)
- RDF (described connected data)
- GGG (content + pointers + relationships + descriptions)

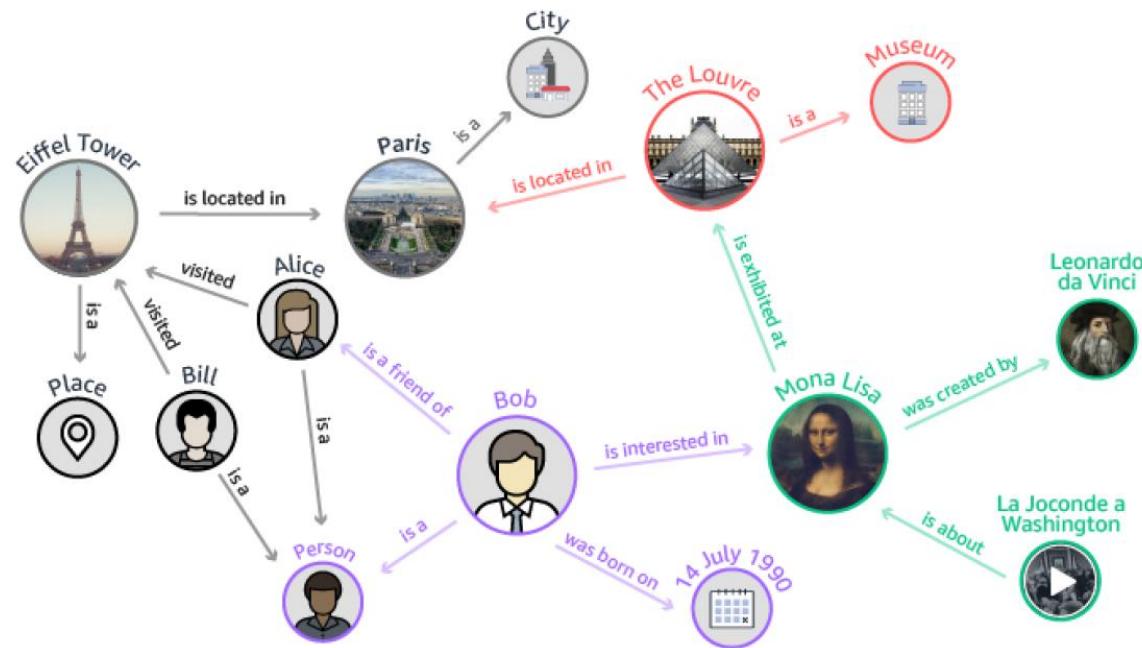
Data is more Semi-Structured

- If you tried to collect all the data of every movie ever made, how would you model it?
 - Actors, Characters, Locations, Dates, Costs, Ratings, Showings, Ticket Sales, etc.



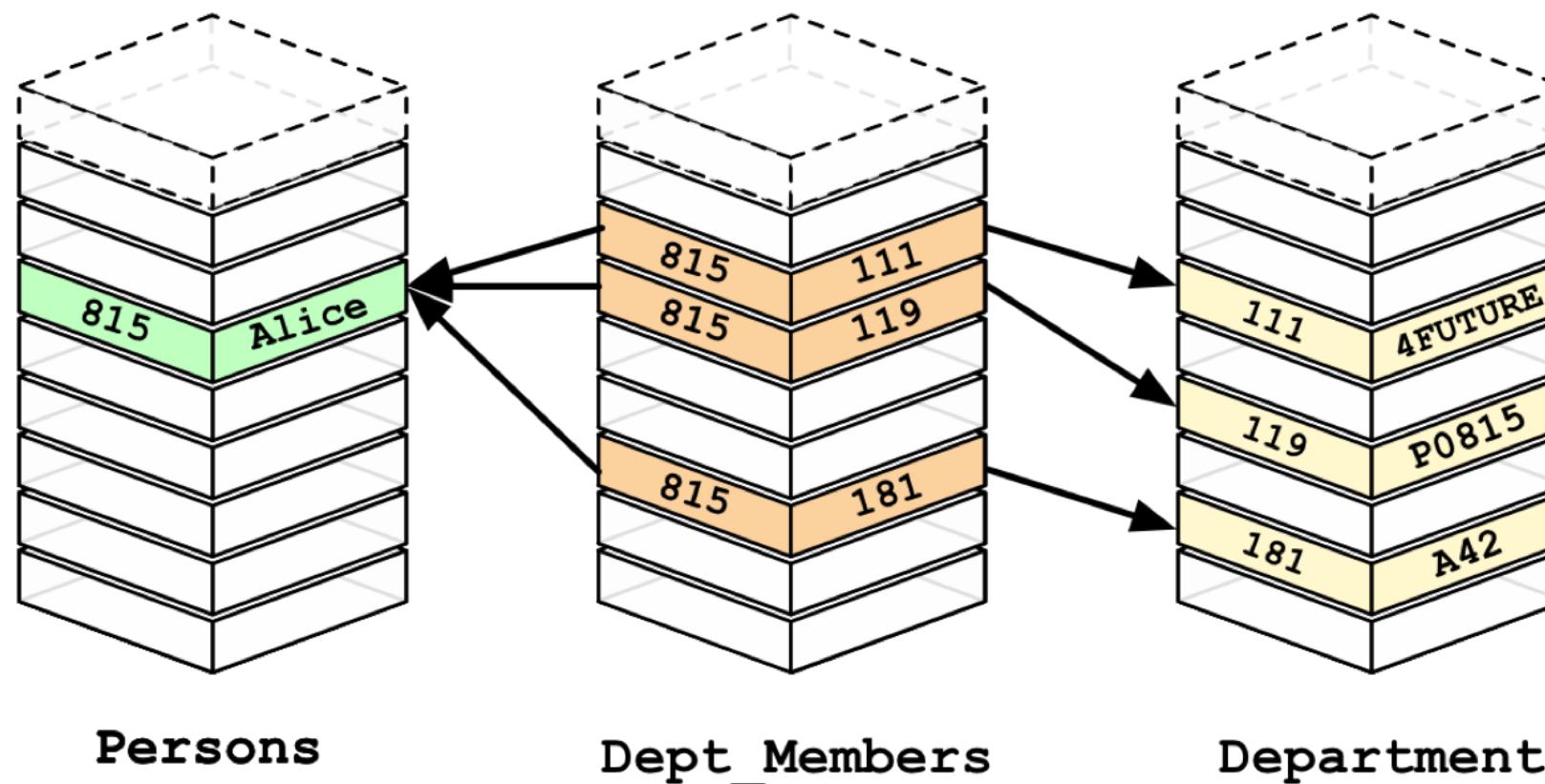
Graphs are important to model and represent those relational features.

- A database with an explicit graph structure.
- A graph database stores nodes and relationships instead of tables, or documents. Data is stored just like you might sketch ideas on a whiteboard.
- The data is stored without restricting it to a pre-defined model, allowing a very flexible way of thinking about and using it.

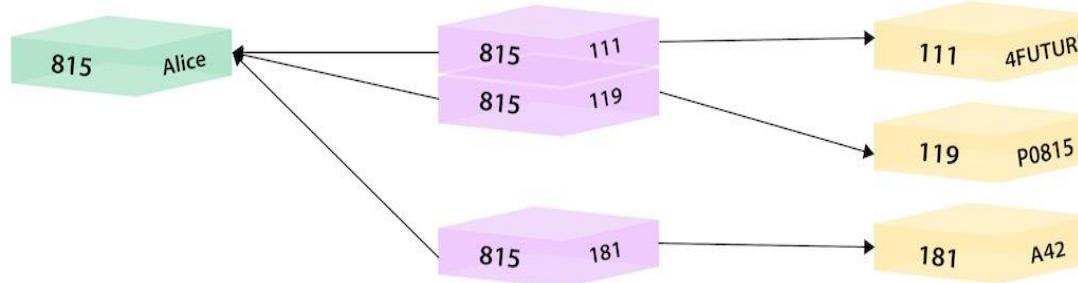


- Nodes are indexed for fast initial lookup.
- As the number of nodes increases, the cost of a local step (or hop) remains the same
- Property graph:
 - Each node/edges is uniquely identified.
 - Each node has a set of incoming and outgoing edges.
 - Each node/edge has a collection of properties.
 - Each edge has a label that defines the relationship between its two nodes.

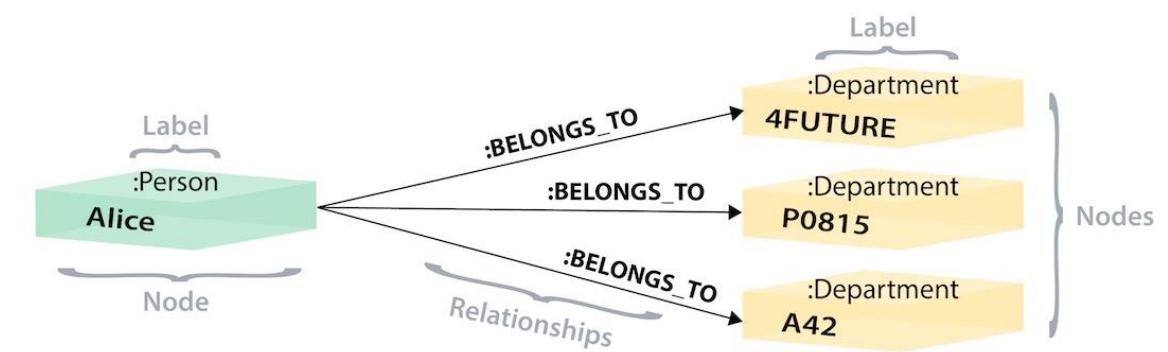
- Look at an example: There is a person named Alice with ID number is 815. Also, there are Department member and Department information.
- How to find the user Alice information belong to which department?



- To find the user Alice and her person ID of 815.
- we search the Person-Department table (orange middle table) to locate all the rows that reference Alice's person ID (815).
- Once we retrieve the 3 relevant rows, we go to the Department table on the right to search for the actual values of the department IDs (111, 119, 181).
- Now we know that Alice is part of the 4Future, P0815, and A42 departments.

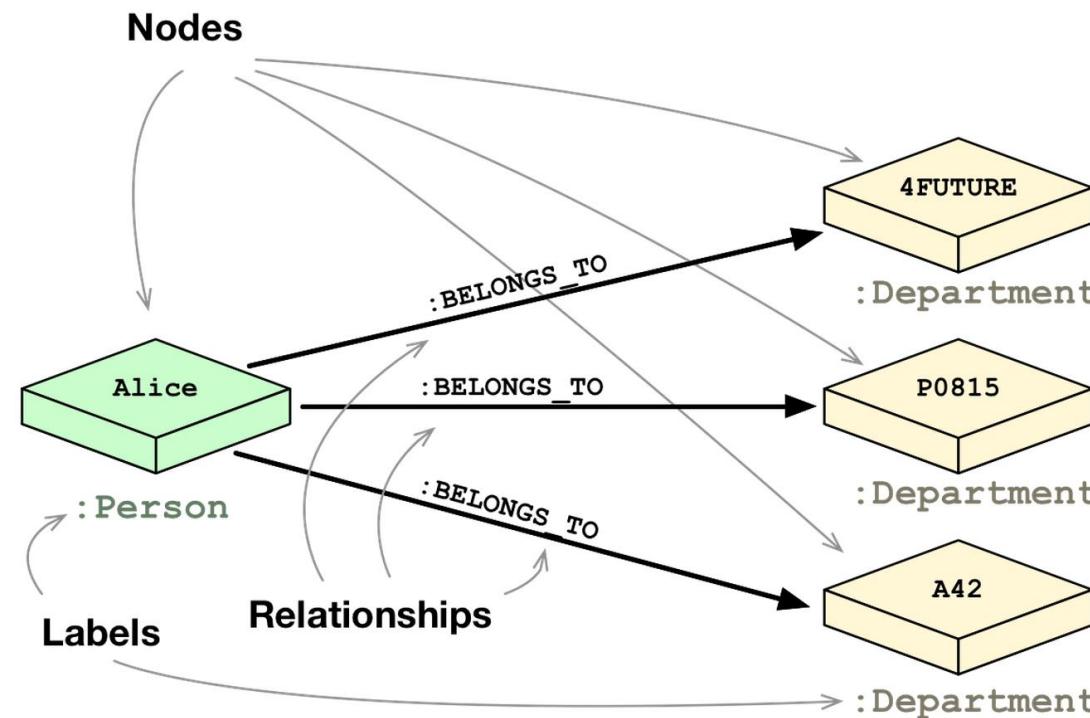


Relational database

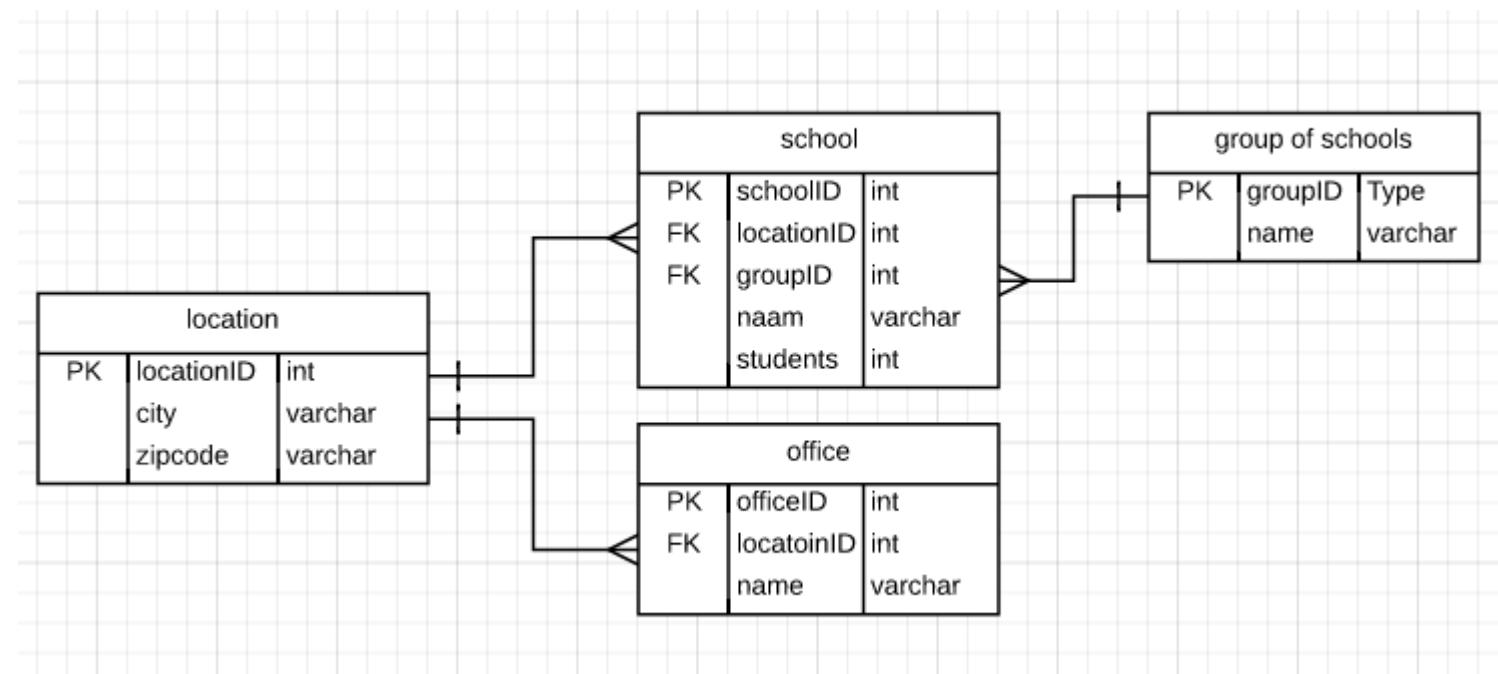


Graph database

- Nodes: Alice, 4FUTURE, P0815, A42
- Labels: Person, Department
- Relationships: BELONGS_TO



- A relational database is a collection of information that organizes data in predefined relationships where data is stored in one or more tables (or "relations") of columns and rows, making it easy to see and understand how different data structures relate to each other.
- Relationships are a logical connection between different tables, established based on interaction among these tables.



Source: Stack

➤ Pros:

- ❖ Schema Flexibility.
- ❖ More intuitive querying.
- ❖ Avoids “join bombs”.
- ❖ Local hops are not a function of total nodes.

➤ Cons:

- ❖ Not always advantageous.
- ❖ Query languages are not unified.



Azure



The #1 Database for Connected Data





➤ **Pros:**

- Runs complex distributed queries
- Scales out through sharded storage
- Returns data natively in JSON, making it ideally suited for web development
- Written on top of GraphQL.

➤ **Cons:**

- No native windows installation
- No support for windows in a production environment

➤ **Pros:**

- Runs on Windows, Mac, and Linux.
- Scalability: Handles large-scale graph data.

➤ **Cons:**

- Only support one model.

➤ Pros:

- Multi model DB – both graph and document DB
- Easily add users/roles
- Supports multiple databases

➤ Cons:

- No native windows service installation
- Requires more schema design up front

➤ Pros:

- Multi-model support.
- Integration with Azure services.

➤ Cons:

- Only run in cloud service.
- Complexity of configuration.



➤ **Pros:**

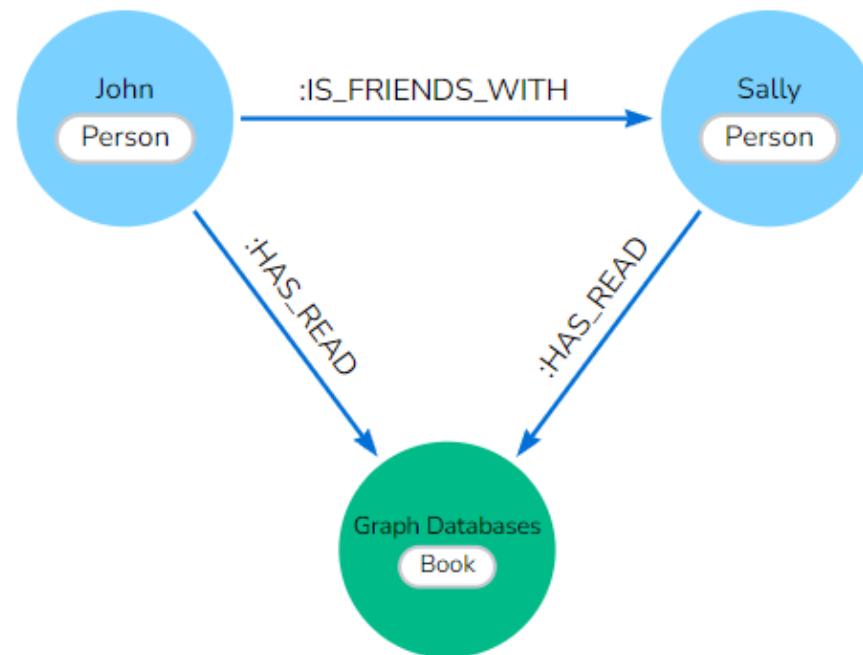
- Runs on Windows natively - in either a console or as a service
- 24/7 production support since 2003 – Mature
- Large and active user community

➤ **Cons:**

- Only one Project can be running on one port at a time

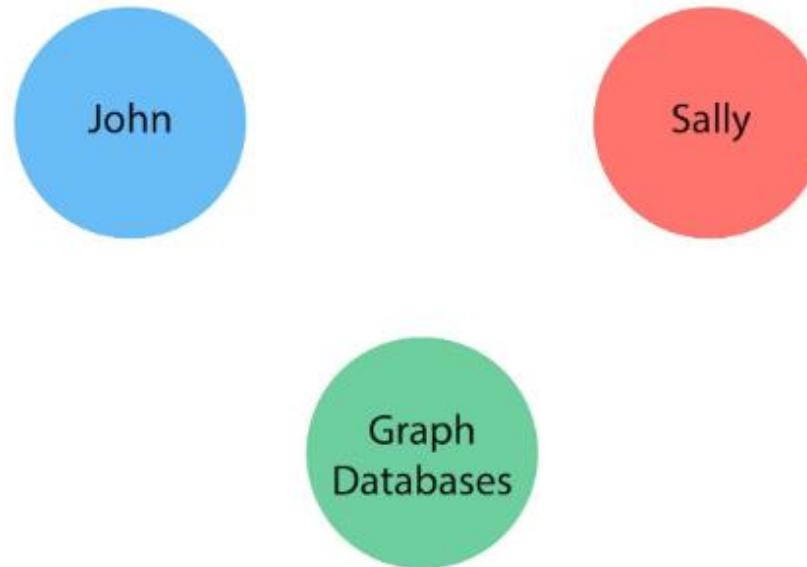
- Full ACID (atomicity, consistency, isolation, durability)
- REST API
- Property Graph
- Lucene Index
- High Availability (with Enterprise Edition)

- The SQL becomes more complex as the length of the relationships increase.
- Performance on the joins becomes an issue quickly.
- SQL is not well-suited to model rich domains.
- It's not easy to start at one row and follow relevant relationships along a path.

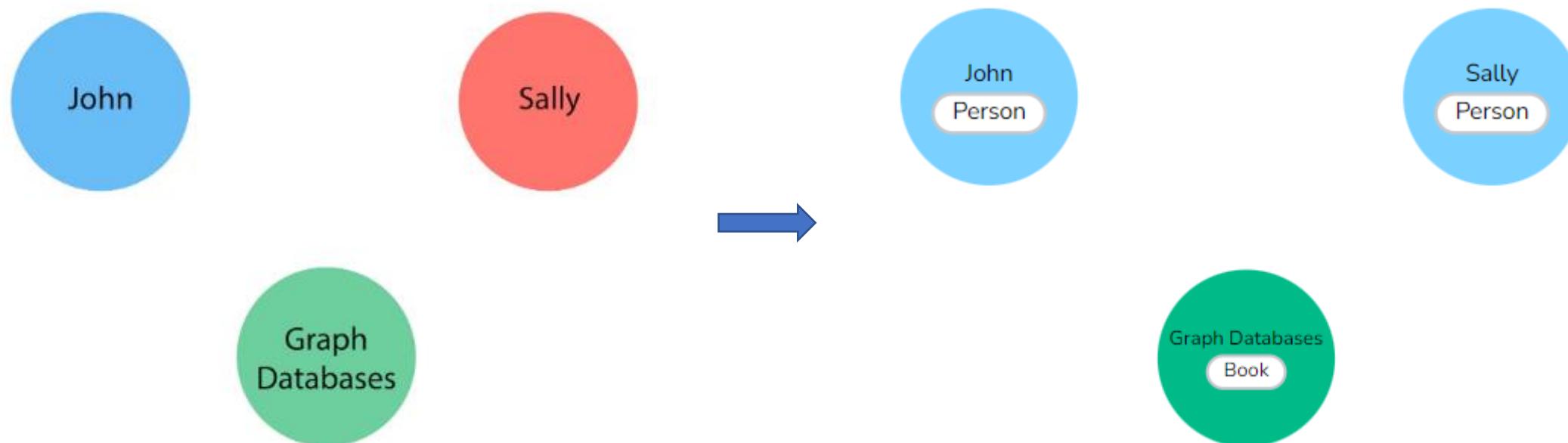


- Two people, Sally and John, are friends.
- Both John and Sally have read the book, Graph Databases.

- Two *people*, **Sally** and **John**, are friends. Both **John** and **Sally** have read the book, **Graph Databases**.
- Extracting the nodes: John, Sally, Graph Databases



- Two *people*, **Sally** and **John**, are friends. Both **John** and **Sally** have read the book, **Graph Databases**.
 - Extracting the nodes: John, Sally, Graph Databases
 - Extracting the labels



- Two people, **Sally** and **John**, are friends. Both **John** and **Sally** have read the book, **Graph Databases**.

- Extracting the nodes: John, Sally, Graph Databases

- Extracting the labels

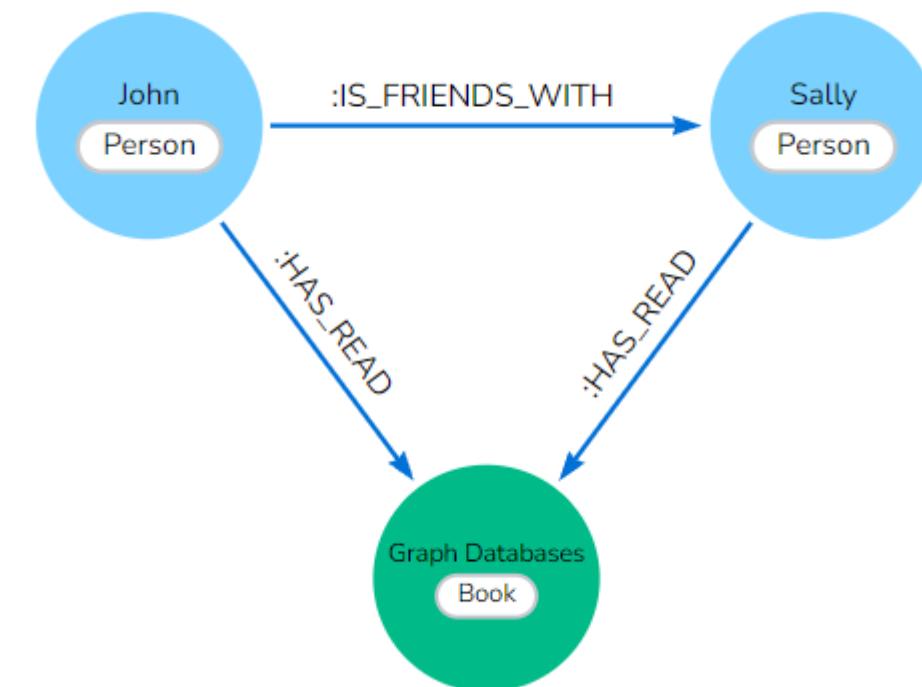
- Defining Relationships:

- John is friends with Sally

- Sally is friends with John

- John has read Graph Databases

- Sally has read Graph Databases



Neo4j

Graph Database

Install on Windows

DevNami

Neo4j Installation.

➤ Sample scripts:

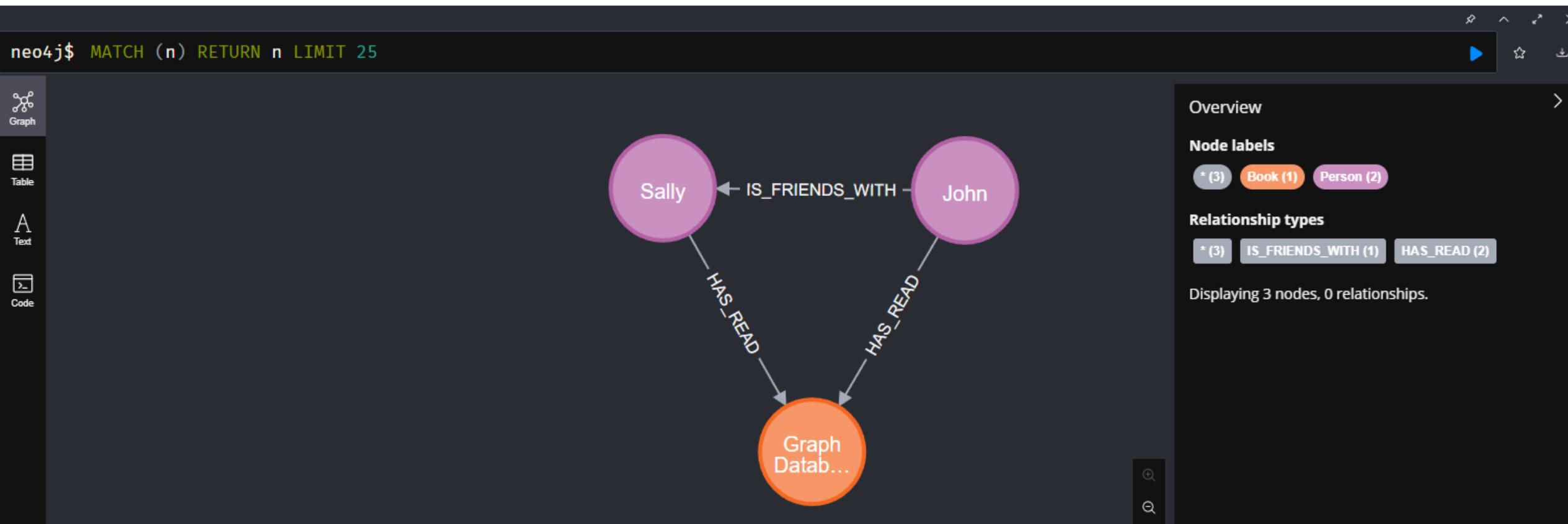
- Two people, John and Sally, are friends. Both John and Sally have read the book, Graph Databases.
- Create Nodes:

```
CREATE(:Person{name:'John',born:'Mar 8, 1998',linkedin:'@john'})  
CREATE(:Person{name:'Sally',born:'Oct 16, 1997',linkedin:'@sali'})  
CREATE(:Book{name:'Graph Databases',published_date:'Nov 16, 2015'})
```

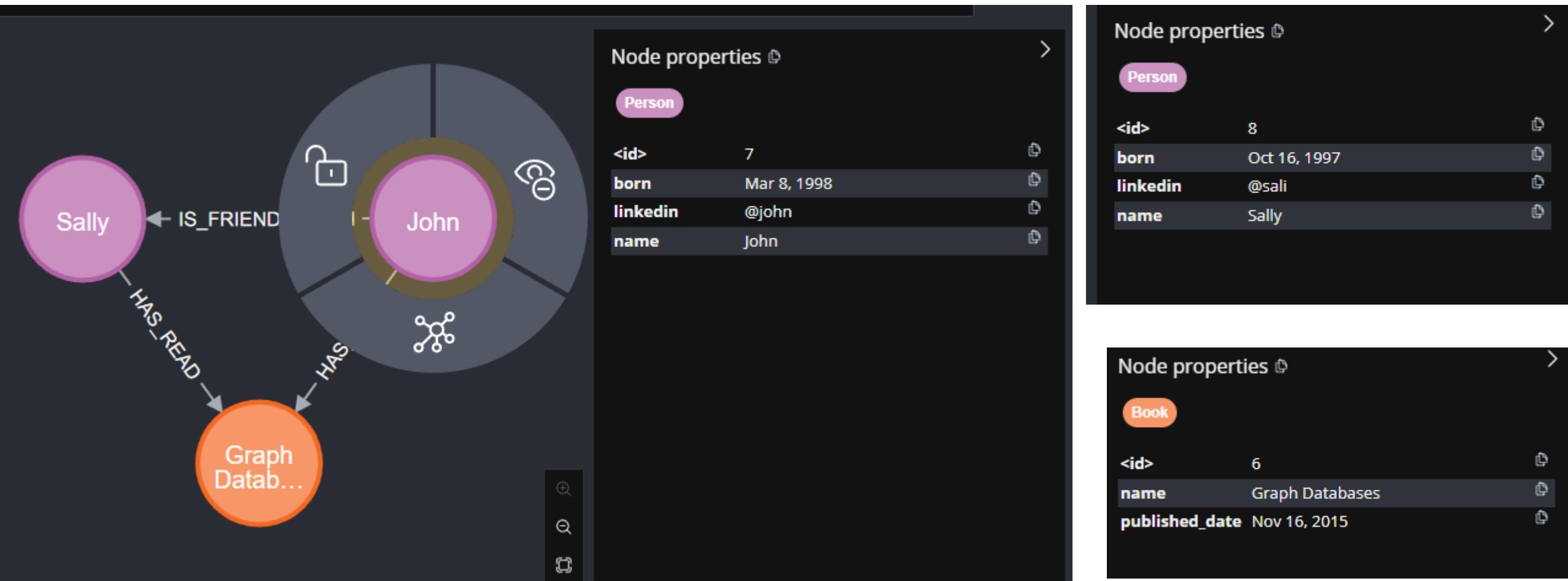
➤ Generate relationships:

```
MATCH (a:Person), (b:Person) WHERE (a.name = 'John' AND b.name = 'Sally') CREATE (a)-[r1  
:IS_FRIENDS_WITH]->(b);  
MATCH (a:Person), (b:Book) WHERE (a.name = 'John' OR a.name = 'Sally') AND b.name = '  
Graph Databases' CREATE (a)-[r2:HAS_READ]->(b);
```

➤ Show database:



➤ Node properties:



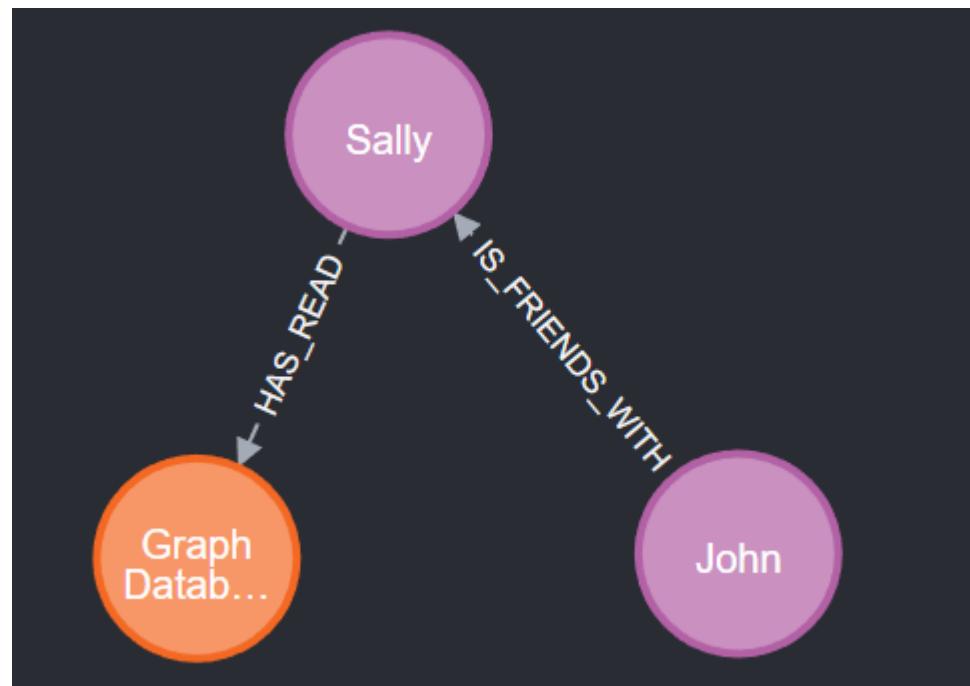
➤ Delete relationships:

```
neo4j$ MATCH (n:Person {name: 'John'})-[r:HAS_READ]->() DELETE r
```

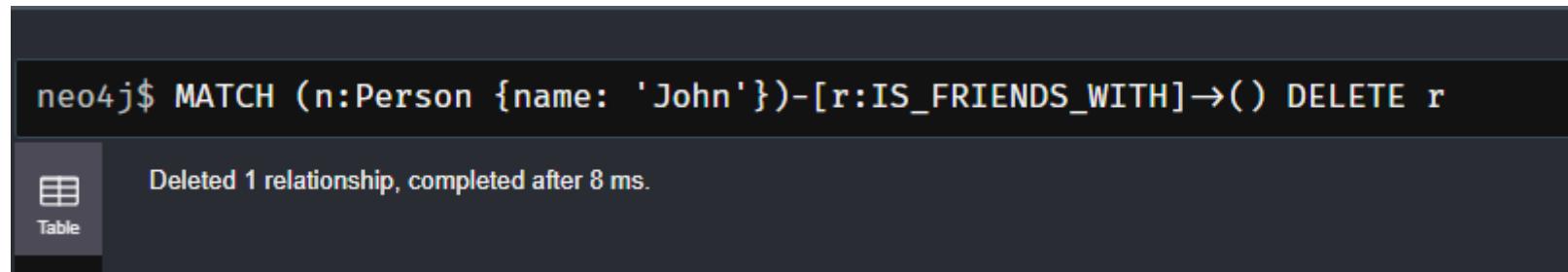
Deleted 1 relationship, completed after 17 ms.

Table

➤ Show database:



- Delete a node: to delete a node, all the relationships must be removed.
 - John has 2 relationship: one is has read with graph database. Another is a friend with Sally.
 - Remove both relationship.



The screenshot shows the Neo4j browser interface. A query is entered in the top text input field:

```
neo4j$ MATCH (n:Person {name: 'John'})-[r:IS_FRIENDS_WITH]→() DELETE r
```

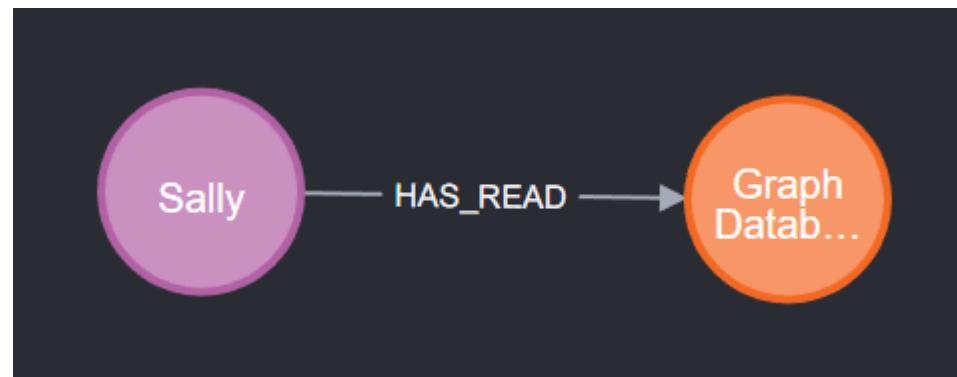
Below the query, the results are displayed in a table. The table has two columns: a small icon representing the result type and the text "Deleted 1 relationship, completed after 8 ms.".

Table	Deleted 1 relationship, completed after 8 ms.

- Remove a node:

```
neo4j$ MATCH (n:Person {name:'John'}) DELETE n
Table
Deleted 1 node, completed after 1 ms.
```

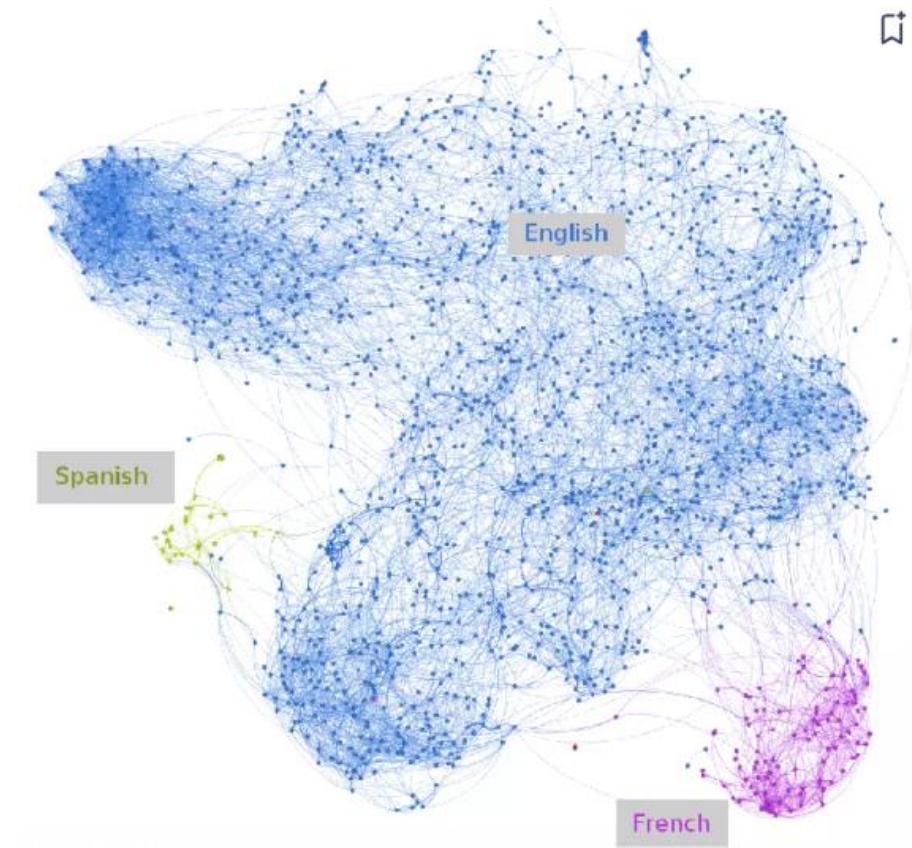
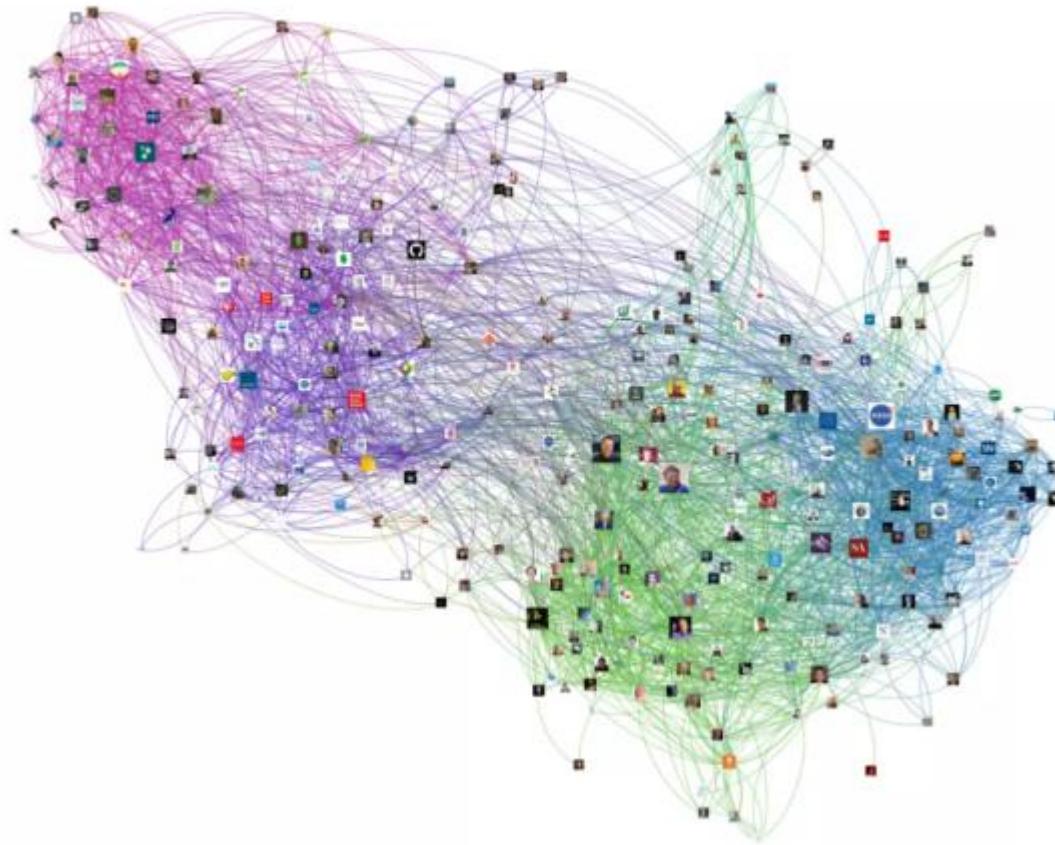
- Show database:



What is Visualization?

31

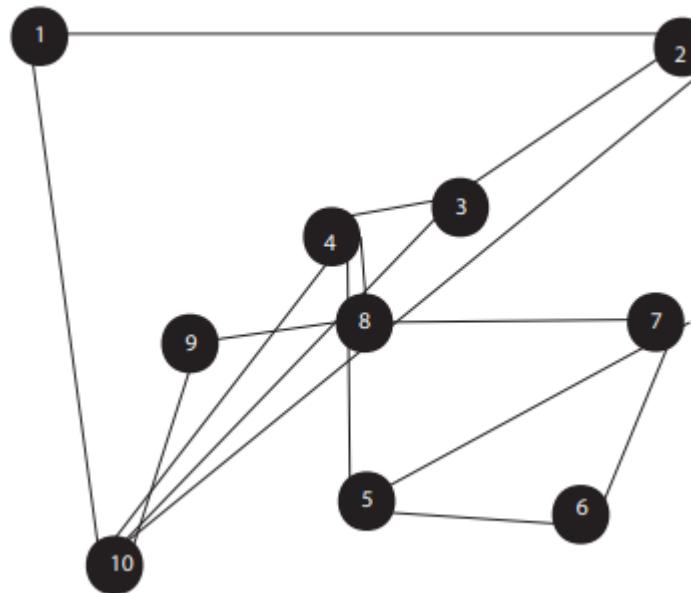
- A visual representation of connected nodes. It shows both individual entities and the relationships between them.



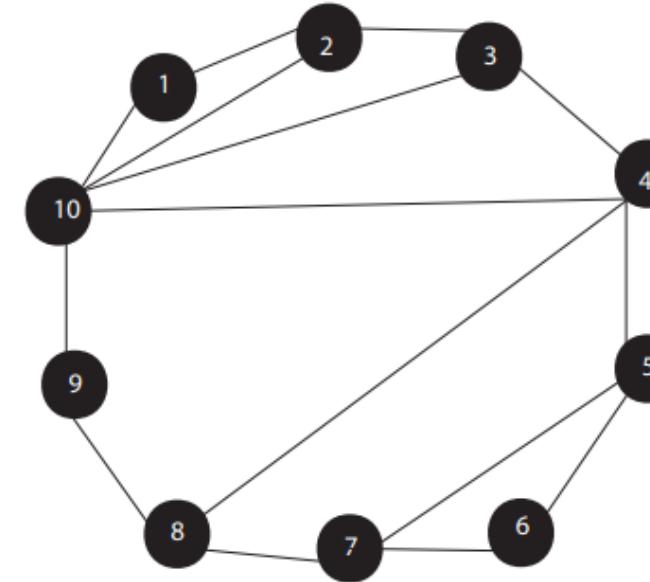
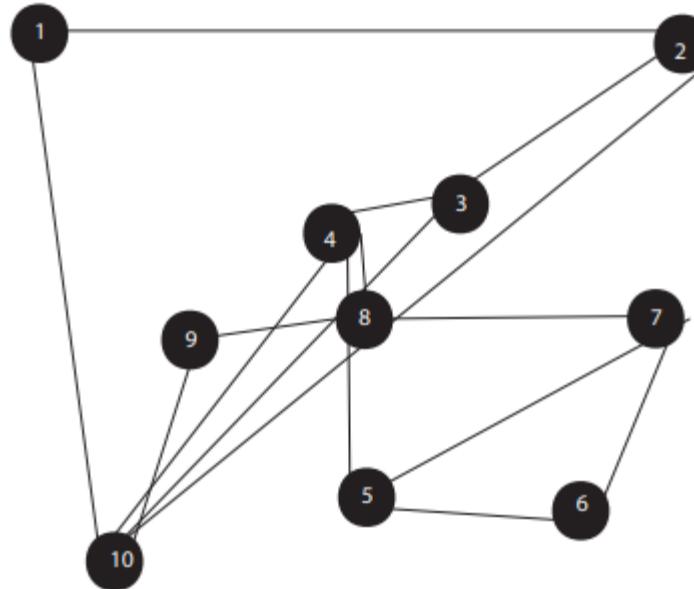
Why are Graph Visualization important?

32

- Input: $Graph G = (V, E)$



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

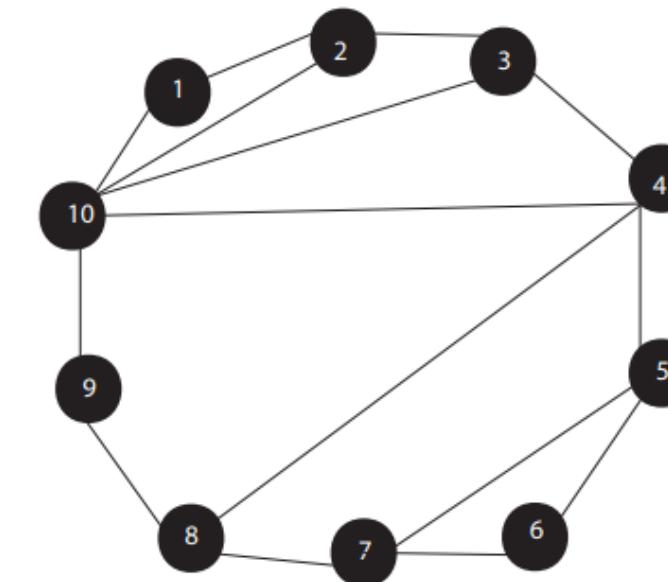


→ Which criteria would you optimize?

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

➤ Criteria:

- Adjacent nodes are close.
- Non-adjacent nodes are far apart.
- The preservation of edge length: edges short, straight-line, **similar length**.
- Densely connected nodes tend to close.
- Draw G with as few crossings as possible.
- Nodes distributed evenly.



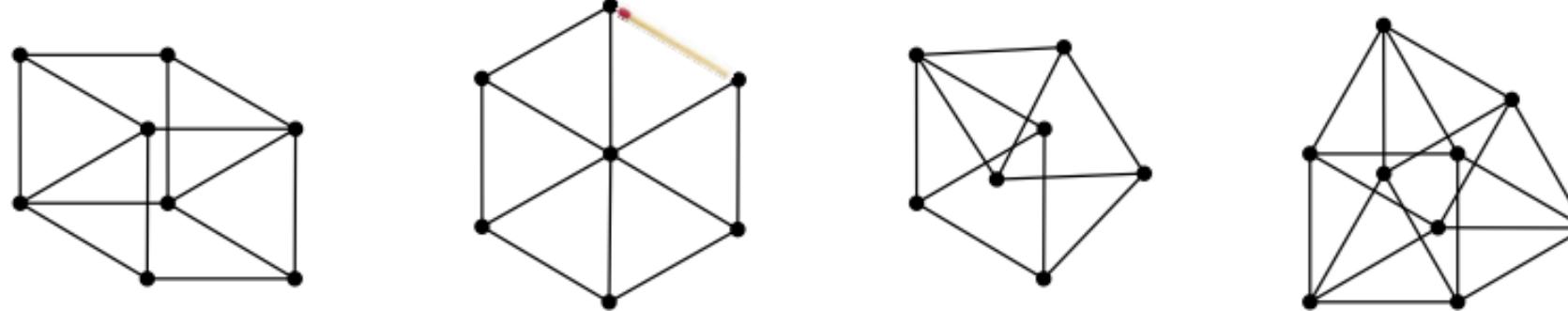
- **Input:** Graph $G = (V, E)$
- **Output:** Creating clear and readable drawings of graph G .
- **Criteria:**

- Adjacent nodes are close.
- Non-adjacent nodes are far.
- The preservation of edge length: edges short, straight-line, **similar length**.
- Densely connected nodes tend to close.
- Draw G with as few crossings as possible.
- Nodes distributed evenly.

Let's take an example

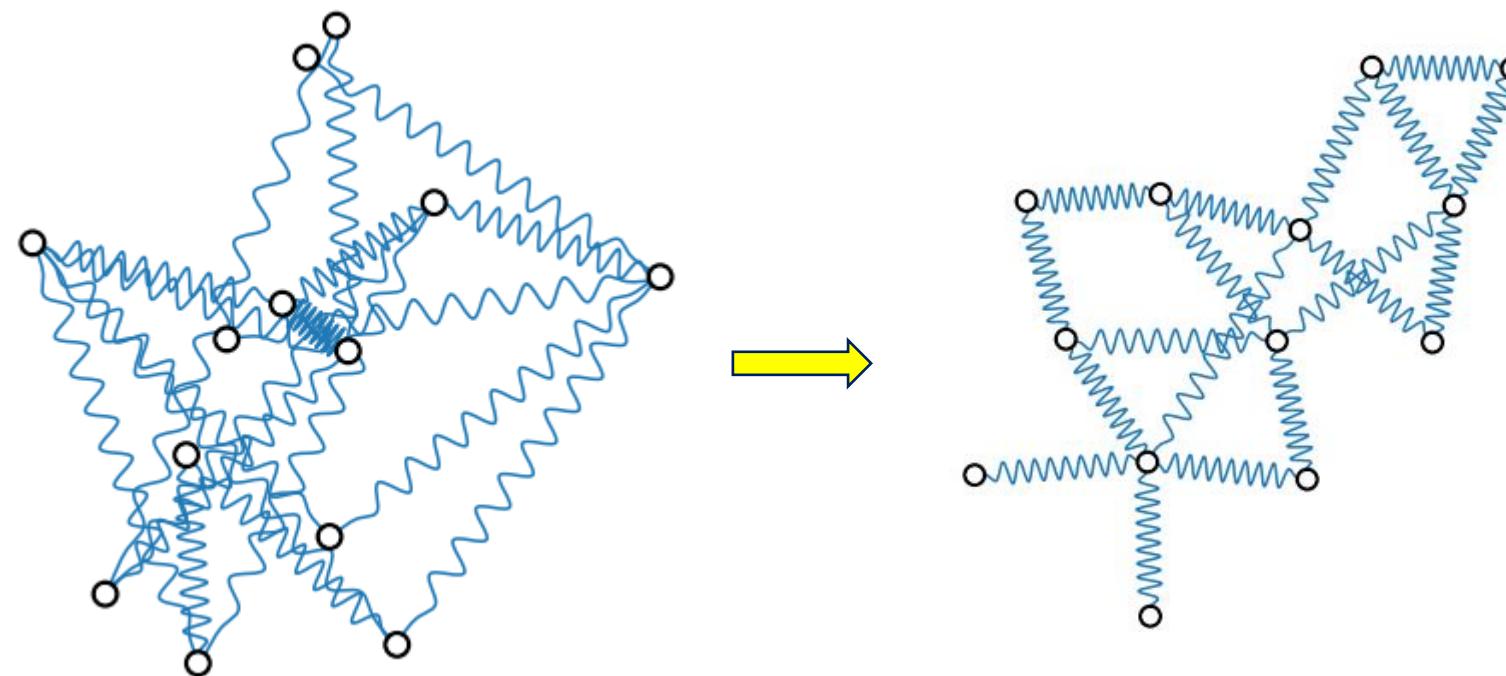


- **Input:** Graph $G = (V, E)$, required edge length $l(e), \forall e \in E$
- **Output:** Drawing of G which realizes all the edge lengths

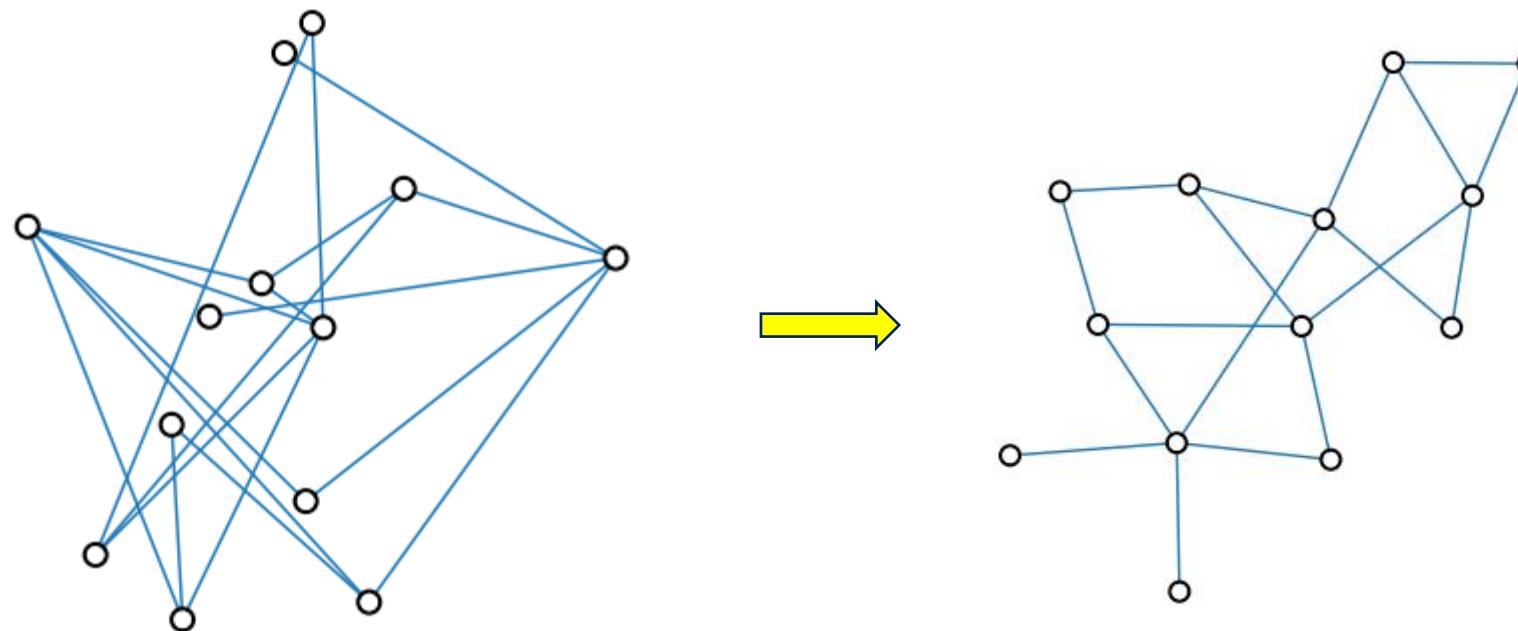


- **NP-hard problem** for:
 - Uniform edge lengths in any dimension.
 - Uniform edge lengths in planar drawing.

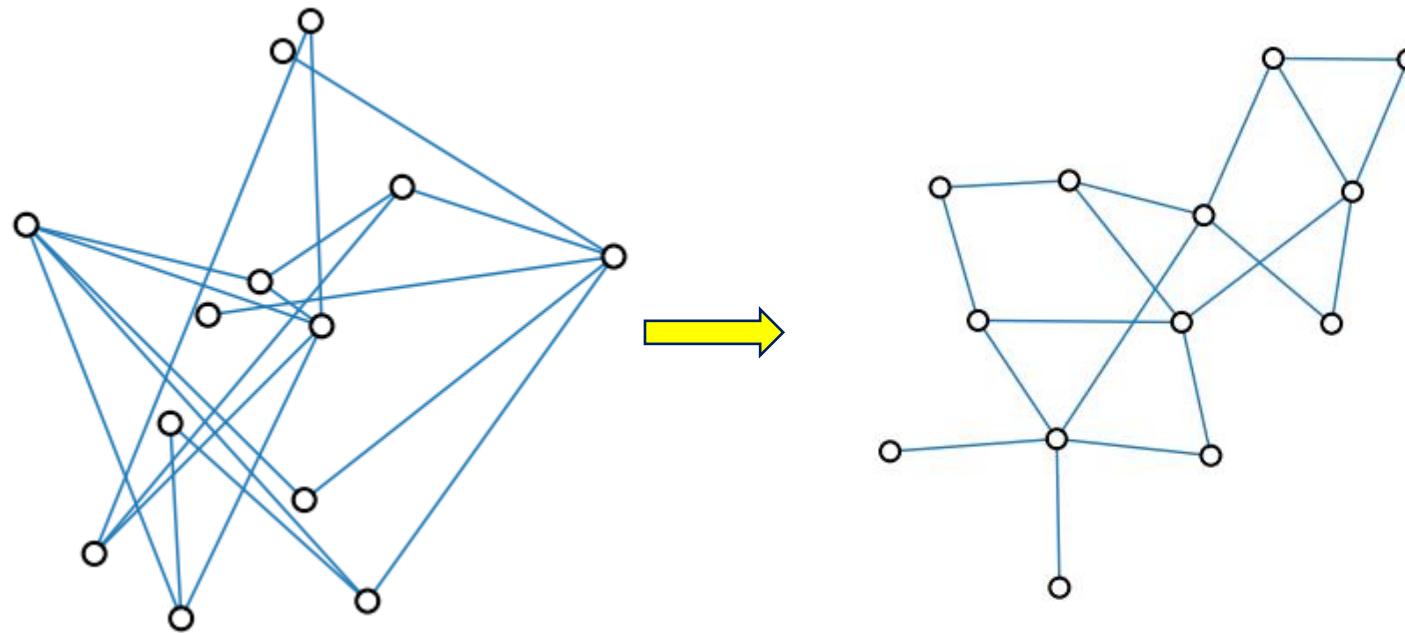
- To embed a graph, we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system.
 - The nodes are placed in some initial layout.
 - The spring forces on the rings move the system to a minimal energy state.



- 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}
- u o~~~~~o v
 f_{spring}
- Repulsive forces:
 non-adjacent nodes x and y : f_{rep}



Spring-embedded by Eades – main functions

- 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)

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, ϵ : 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}

Spring-embedded by Eades – Algorithm

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, ϵ : 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}

Spring-embedded by Eades – Algorithm

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, ϵ : 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      $p_i \leftarrow p_i + \delta(t) \cdot F_i(t)$ 
```

Update new location of node

```

7    $t \leftarrow t + 1$ 
```

```

8 Return  $p$ 
```

End layout

cooling factor

➤ Spring forces:

Adjacent nodes u and v : f_{spring}

➤ Repulsive forces:

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

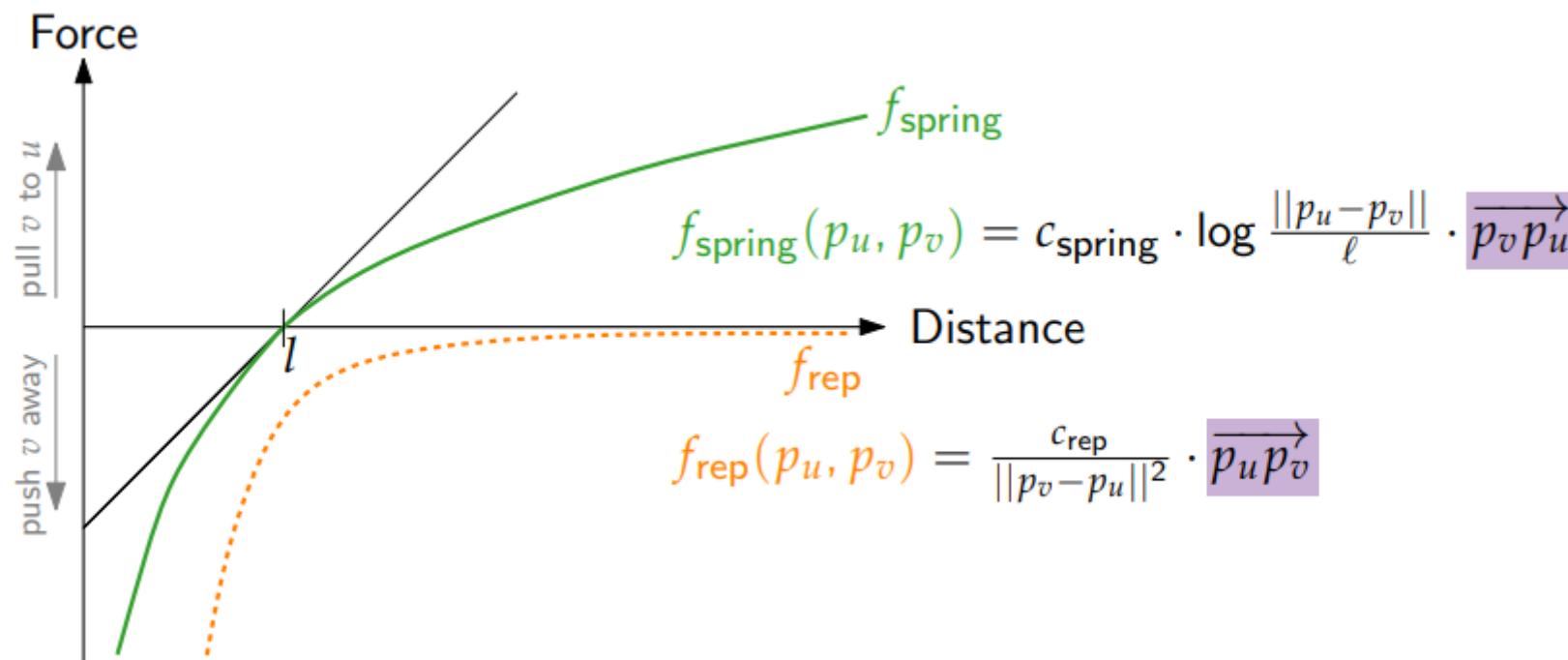
Where:

- $l = l(e)$: 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.

Variant by Fruchterman & Reingold

- 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

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

- Resulting force between adjacent vertices v_i and v_j

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

Algorithm 1: SpringEmbedder

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

Input: p : initial layout, ϵ : threshold

Output: p : is end layout

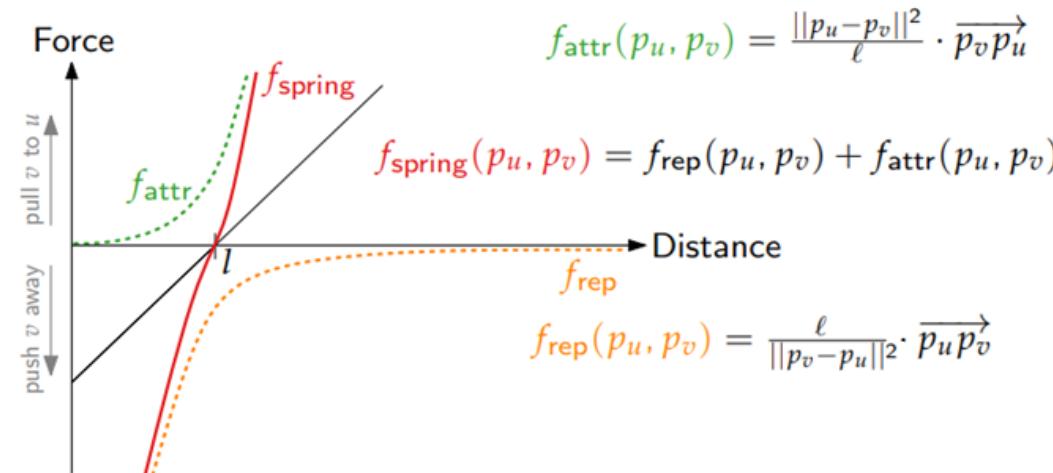
```

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 faraway node u (u and v are non-adjacent).



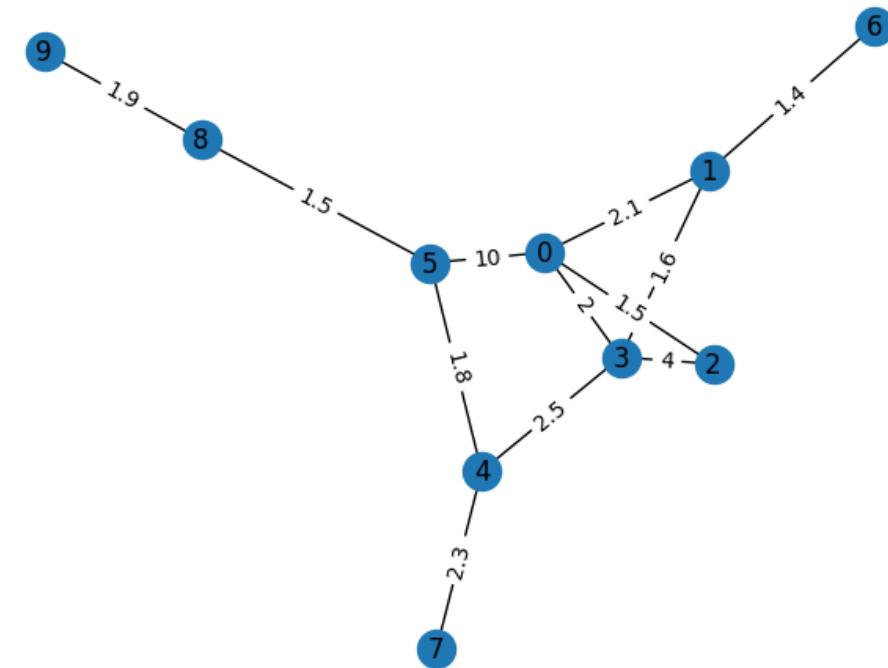
➤ Spring layout

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

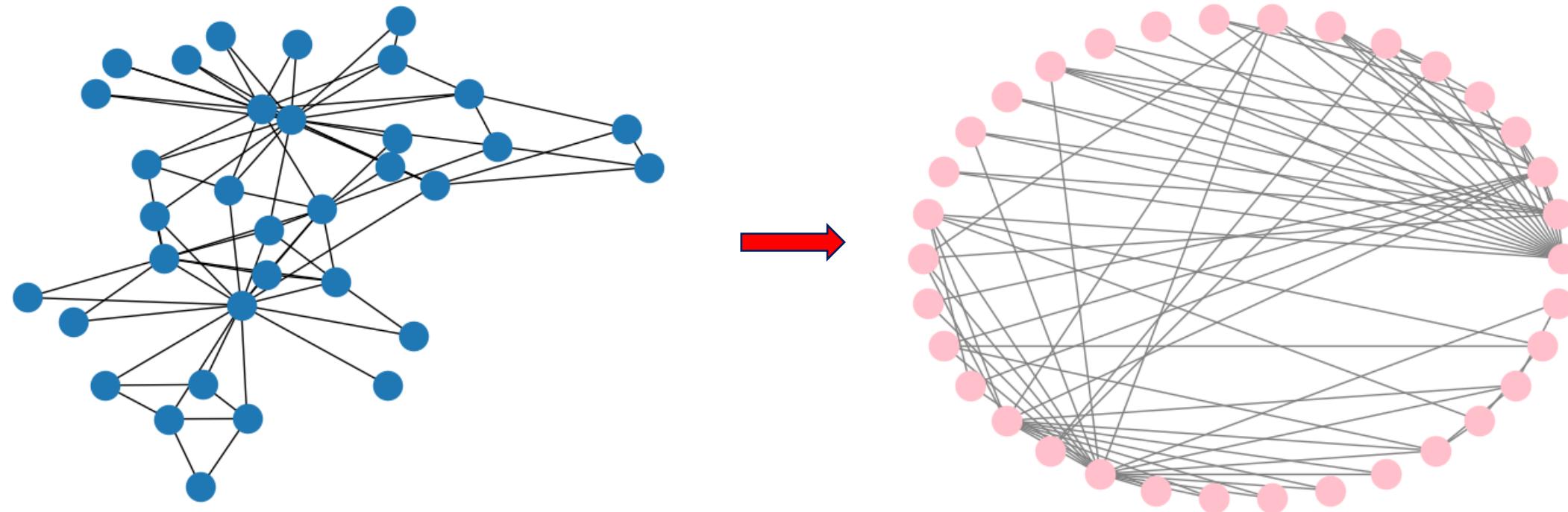
edges = [(0, 1, 2.1), (0, 2, 1.5), (0, 3, 2), (0, 5, 10), (1, 3, 1.6), (1, 6, 1.4),
         (2, 3, 4), (3, 4, 2.5), (4, 5, 1.8), (4, 7, 2.3), (5, 8, 1.5), (8, 9, 1.9)]
# add node/edge pairs
G.add_weighted_edges_from(edges)

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

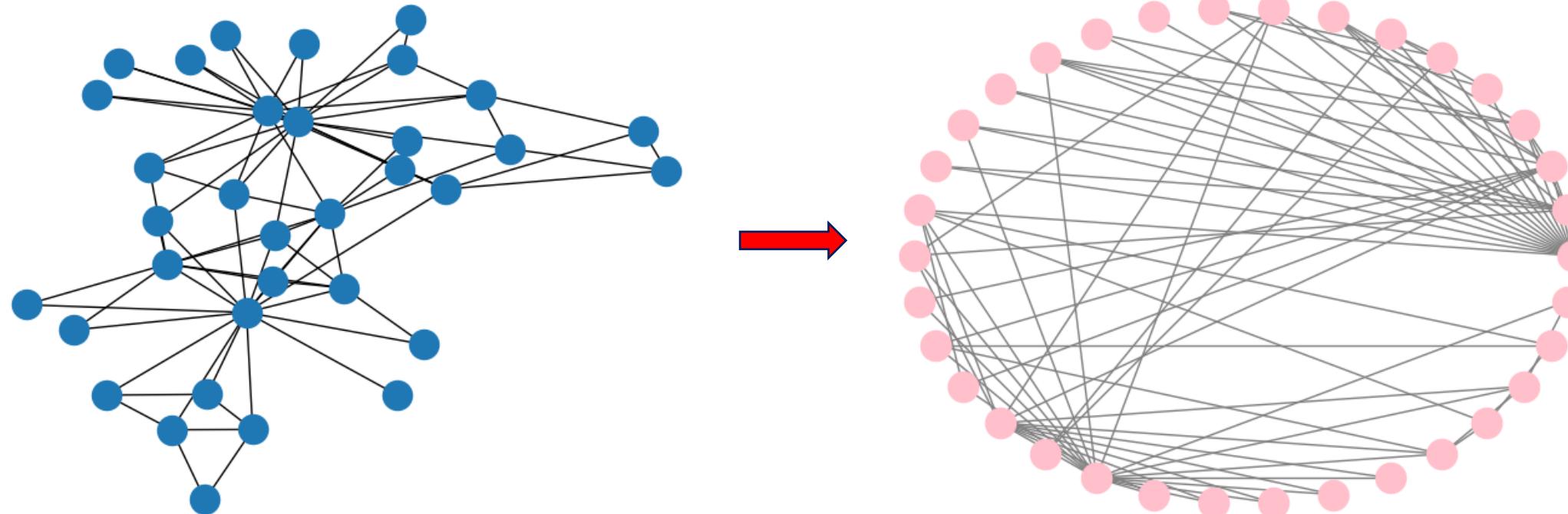
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
```



- Input: Graph $G = (V, E)$
- Output: Creating circular drawings of graph G .



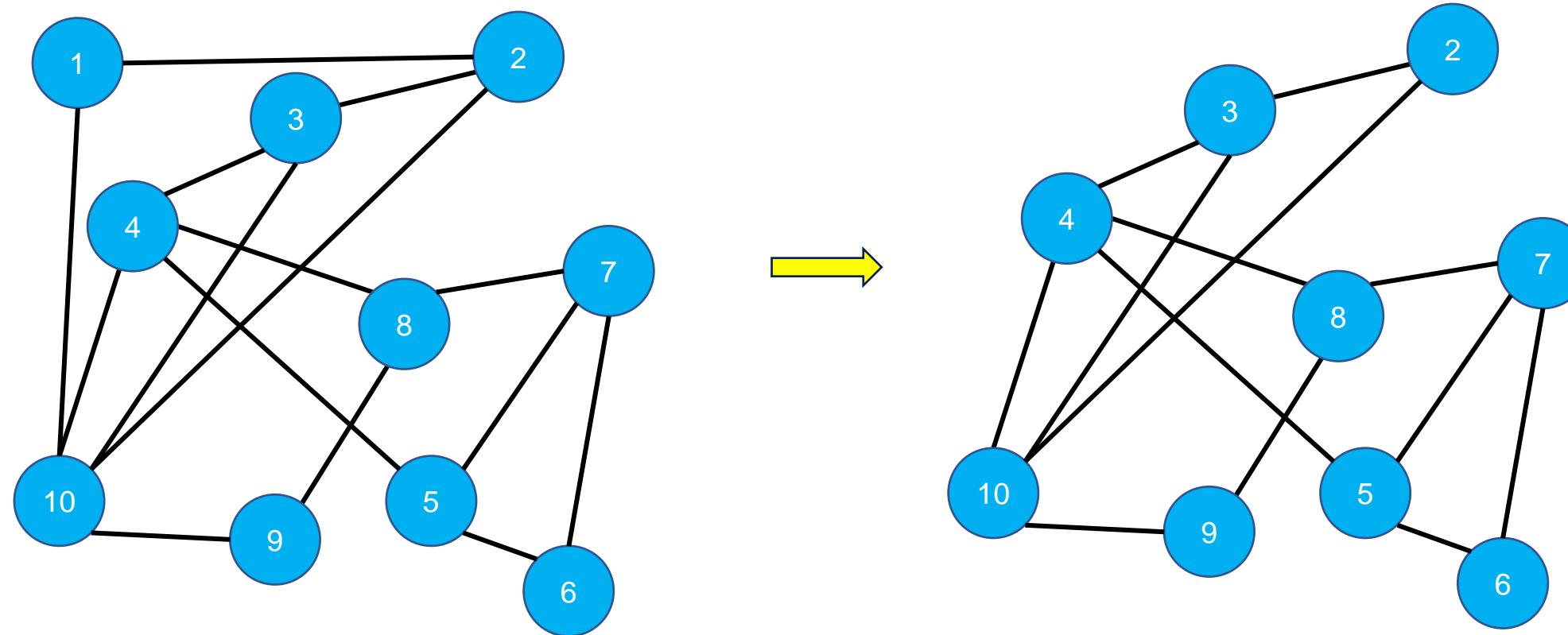
- **Input:** A biconnected graph, $G = (V, E)$.
- **Output:** A circular drawing Γ of G such that each node in V lies on the periphery of a single embedding circle.



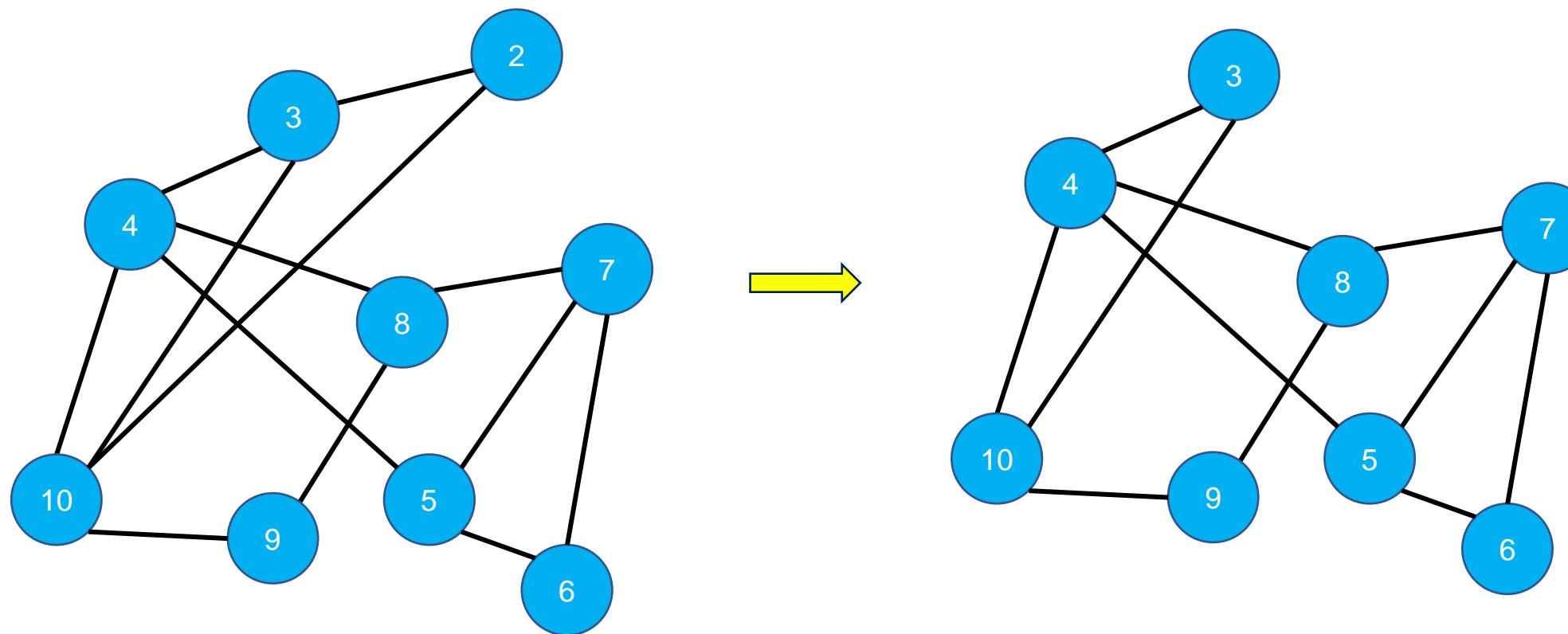
- In circular graphs, close nodes should not be connected:
 - Idea: Finding and store nodes that have two non-connected neighbours by using BFS algorithm.
 - Implement:
 - Starting at a random node, store nodes that do not have non-connected neighbours in a stack.
 - Restore the graph to generate a circle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

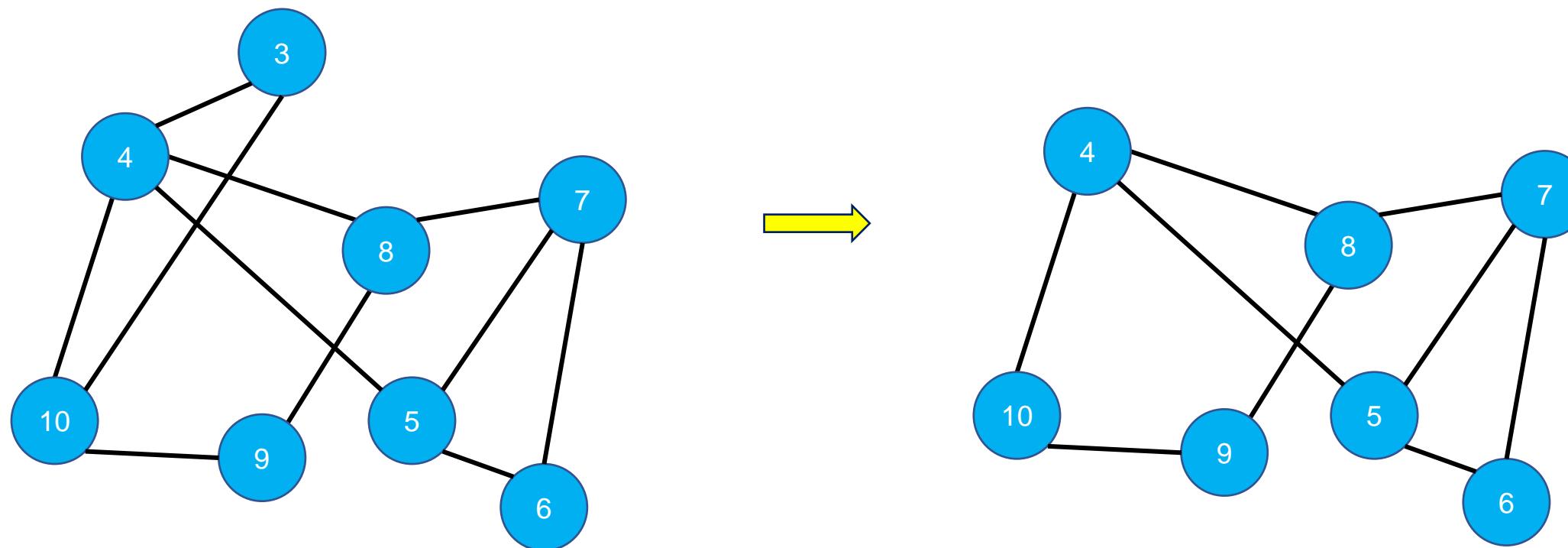
- Start with a graph G in the left side.
- First, we choose randomly Node 1.
- We check for edge(2,10), which exists. We store it and remove Node 1.



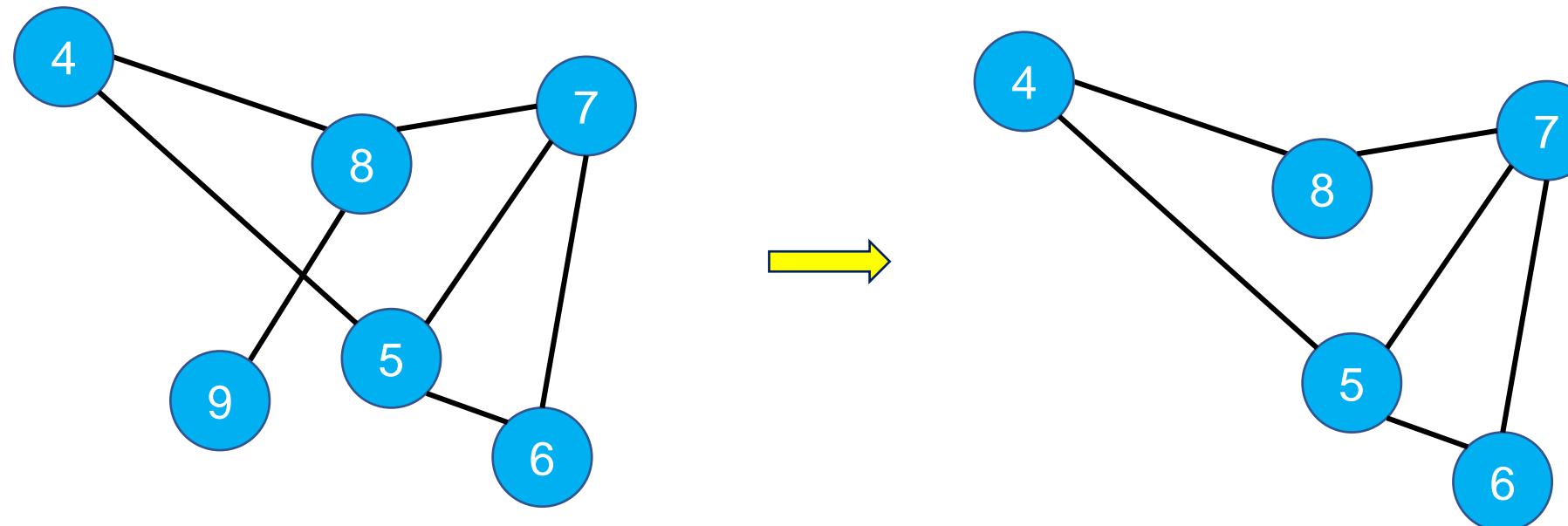
- Next, we choose a lowest-degree neighbor of the removed Node 1, which is 2.
- Check for edge (3,10) which exists. We store it and remove Node 2.



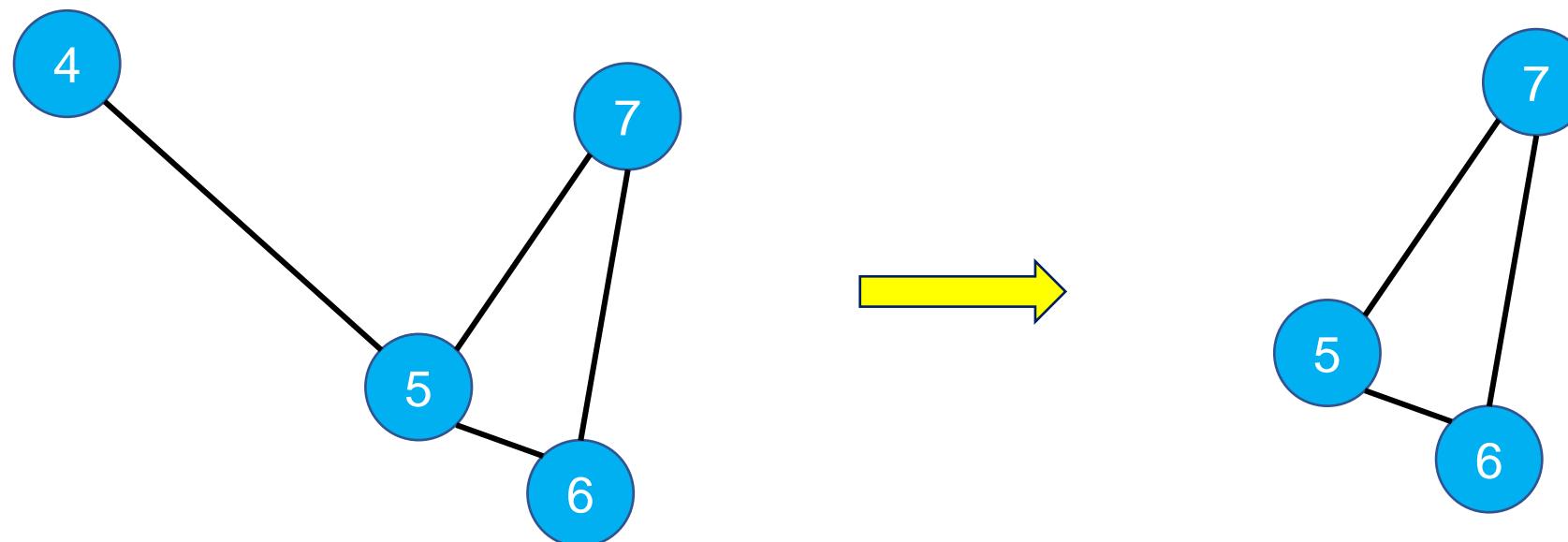
- Next, we select a lowest-degree neighbor of Node 2.
- This is Node 3. We check for edge (4,10).
- It exists so we store it and remove Node 3.



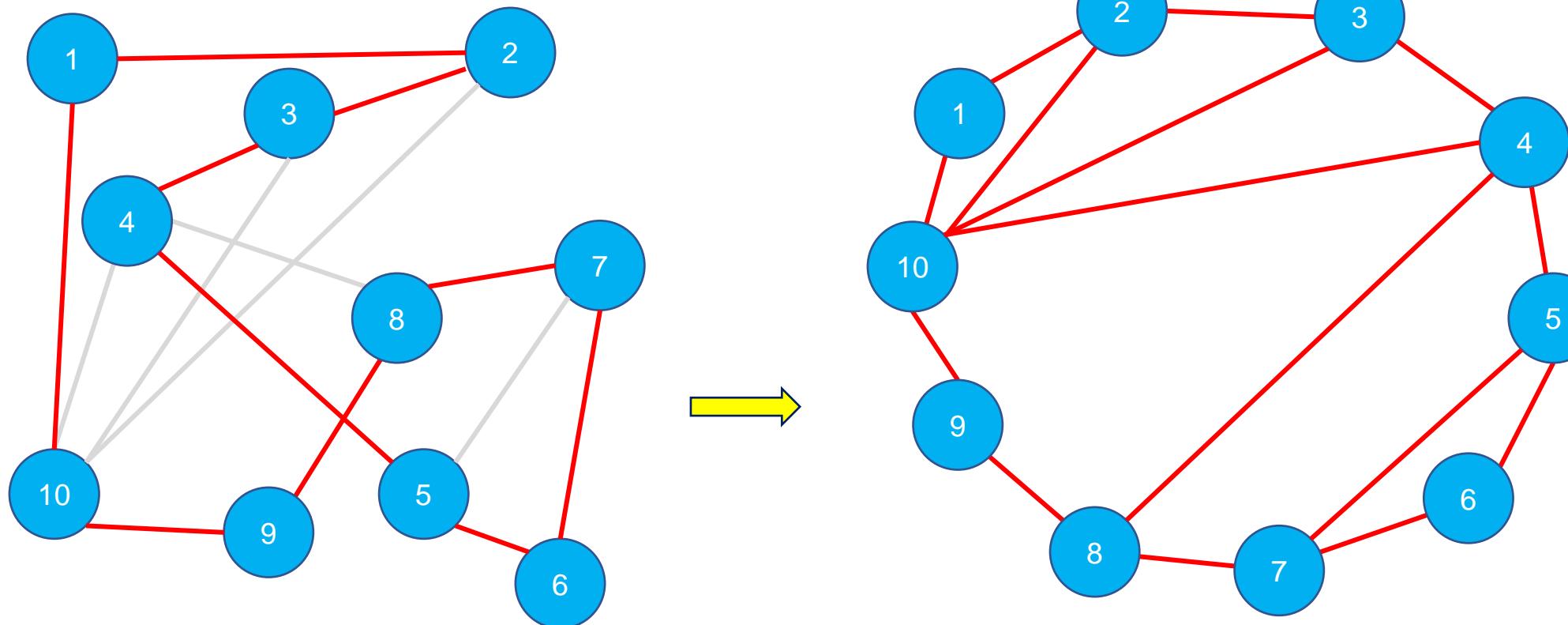
- Similarly, we can select Node 10 and check for edge (4,9).
 - It does not exist.
 - So, we add edge (4,9) which is a triangulation edge, store it and remove Node 10.
- We continue choosing Node 9 and check for edge (4,8). It exists so we store it and remove Node 9. Next, for Node 8 we check for edge(4,7)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



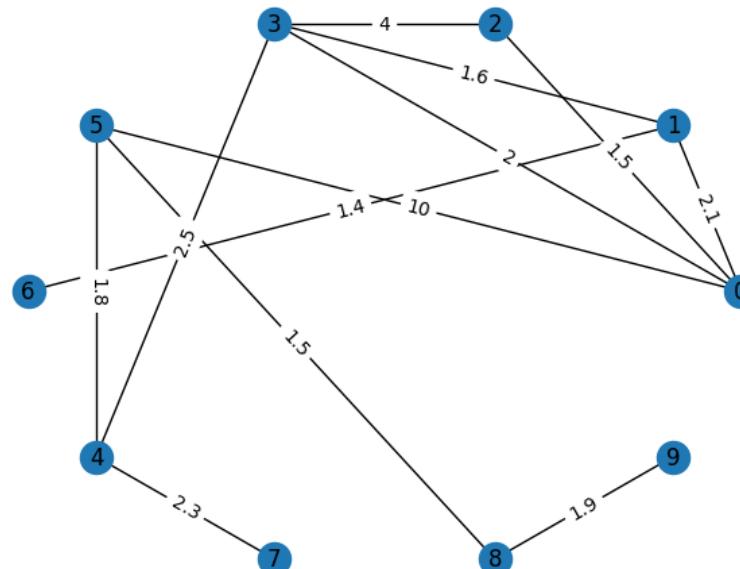
➤ Circular layout

```
# 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)

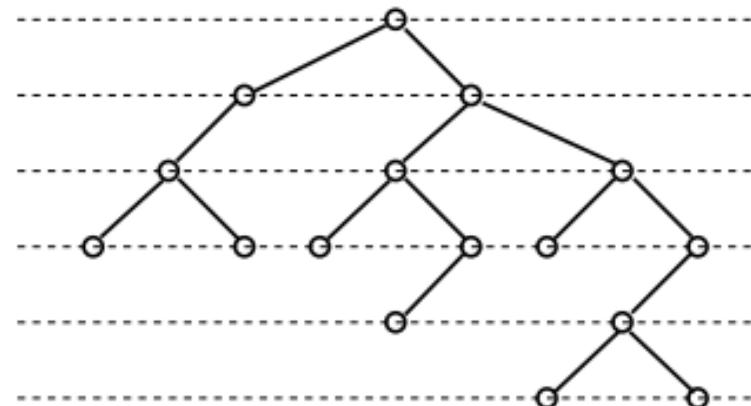
labels = nx.get_edge_attributes(G,'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
```



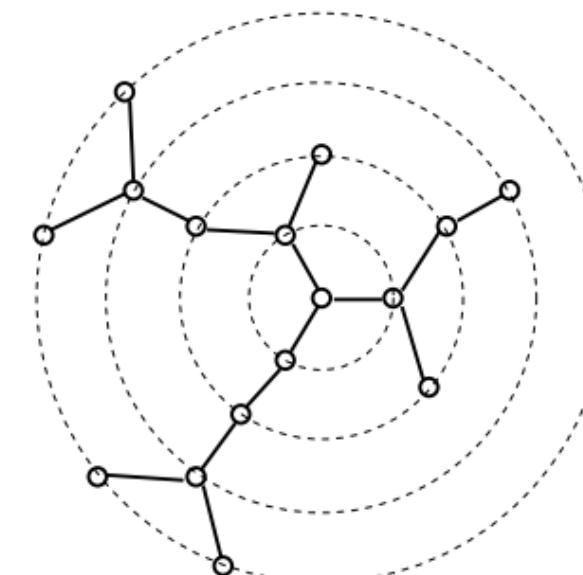
- Trees:

- Requirements:

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



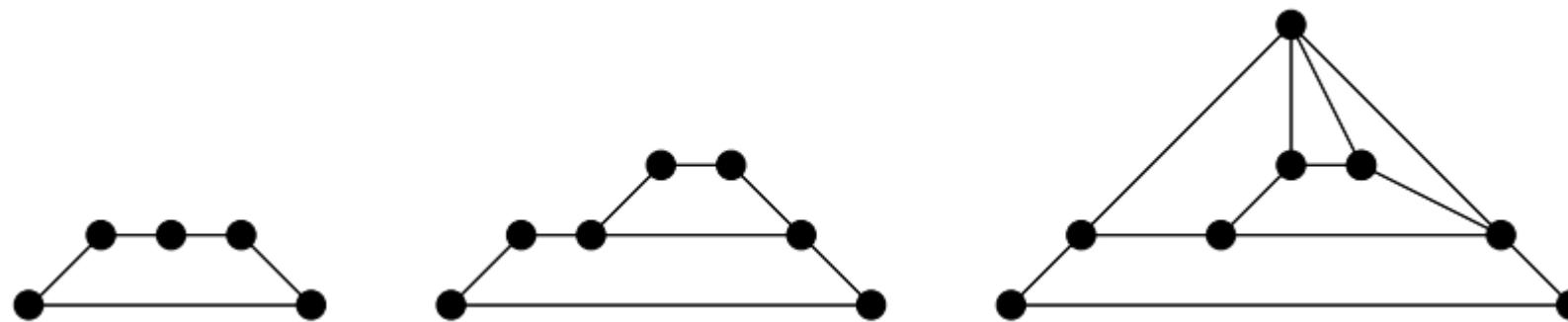
A layered tree drawing



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

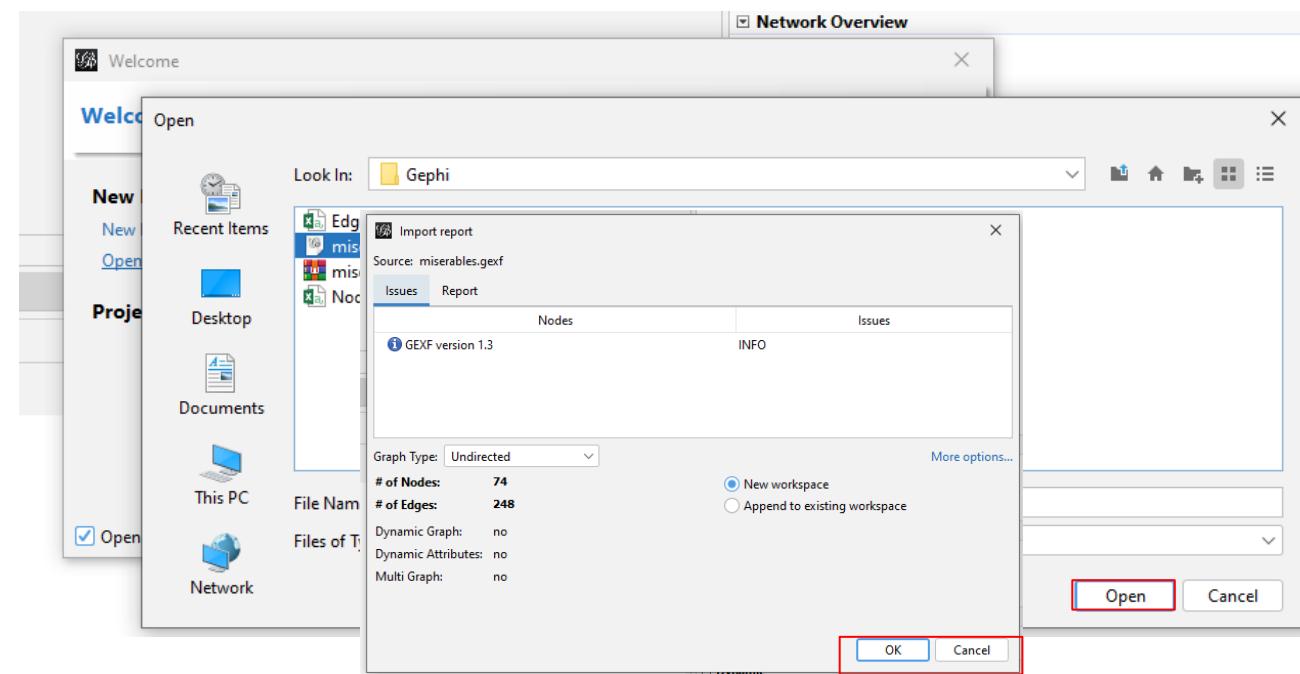
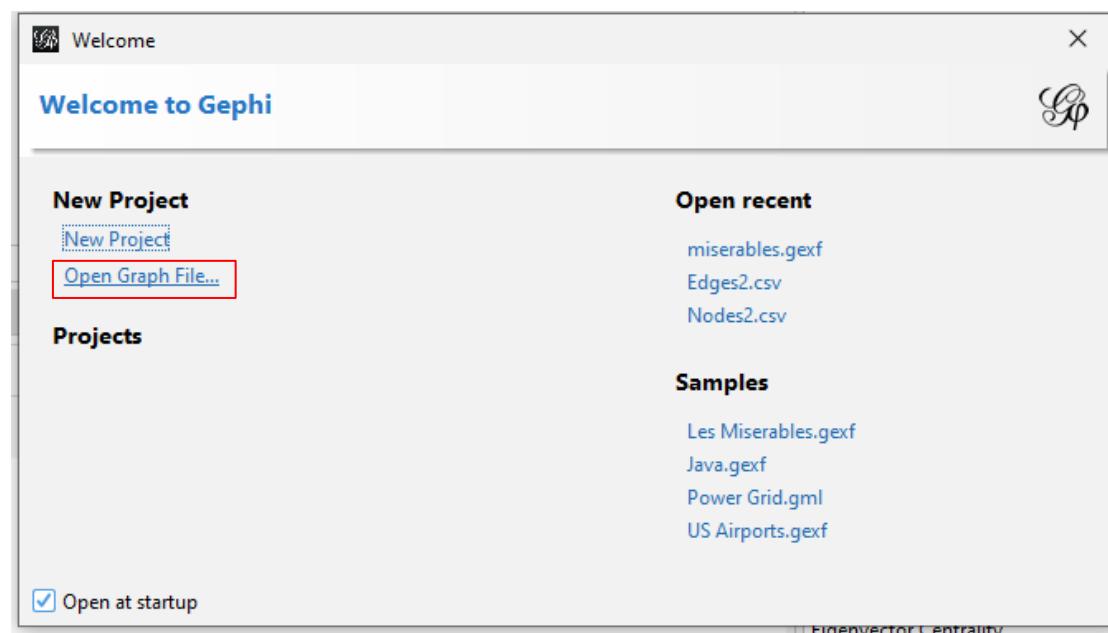
- There are several open-source tools for network analysis:
 - NetworkX in Python.
 - iGraph packages in R.
 - Gephi.
 - Cytoscape.
 - NodeXL.
 - Graphia.app.
 - Gephisto.
 - Ucinet.
 - Graphviz.
 - Etc...

- An open-source visualization exploration software without having coding skills.
- A tool for data analysts and scientists keen to explore and understand **graphs**.
- Functions:
 - Real time visualization.
 - Manipulation with Excel structures.
 - Appearance properties with metrics.
 - Understand patterns in visualization with dynamic filtering and layout.
 - Extensible plugins.

- Applications of Gephi:
 - Exploratory Data Analysis.
 - Link Analysis.
 - Social Network Analysis.
 - Biological Network Analysis.
 - Poster Creation.
- Different layouts:
 - CircularLayout
 - GeoLayout
 - Geometric transformation
 - Noverlap
 - OpenOrdLayout

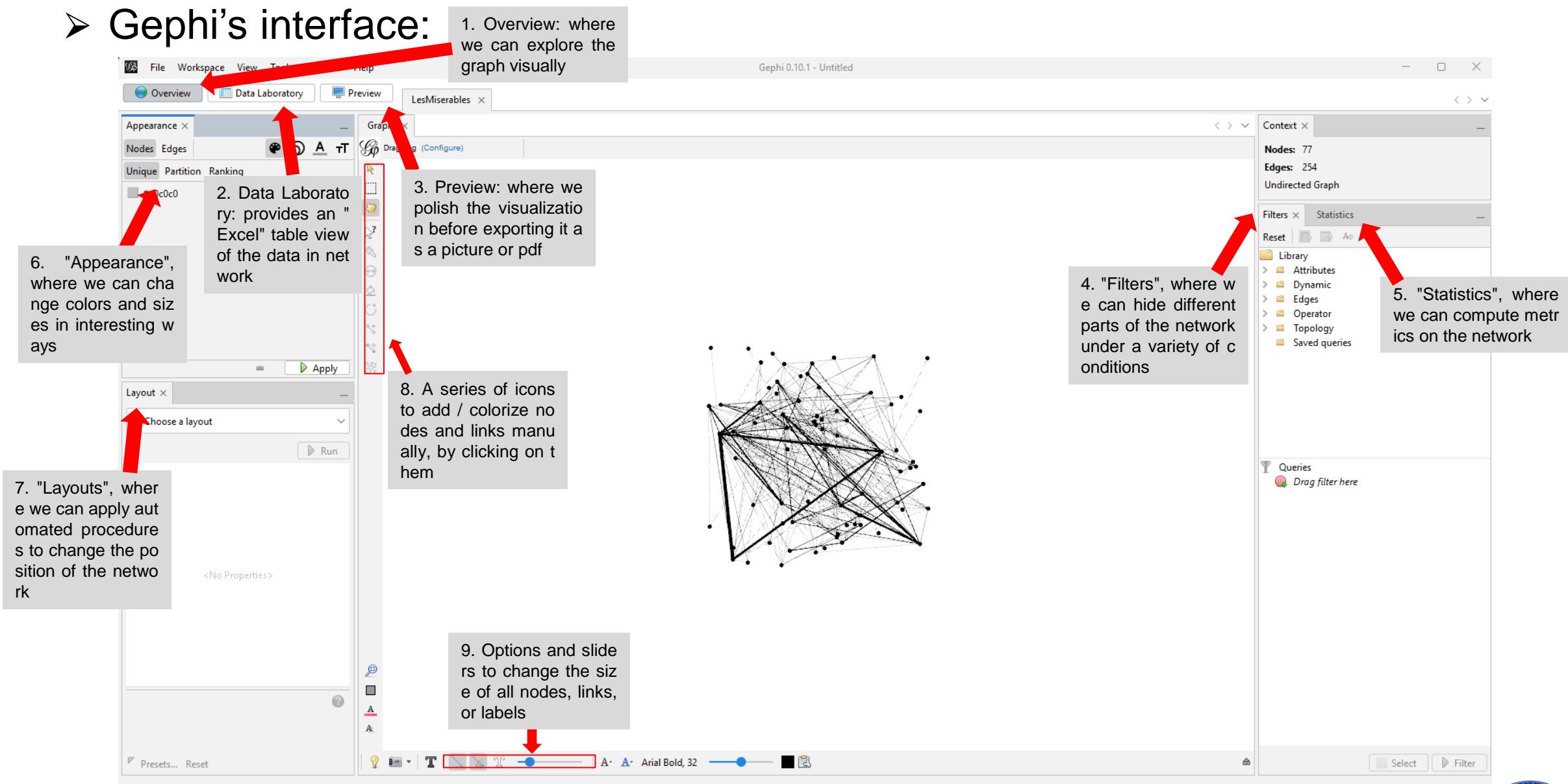
➤ Prepare:

- Sample graph: LesMiserables.gexf (download in [here](#) or in class's github)
- Open Gephi.
- On the Welcome screen that appears, click on Open Graph File.
- Open LesMiserables.gexf and click OK



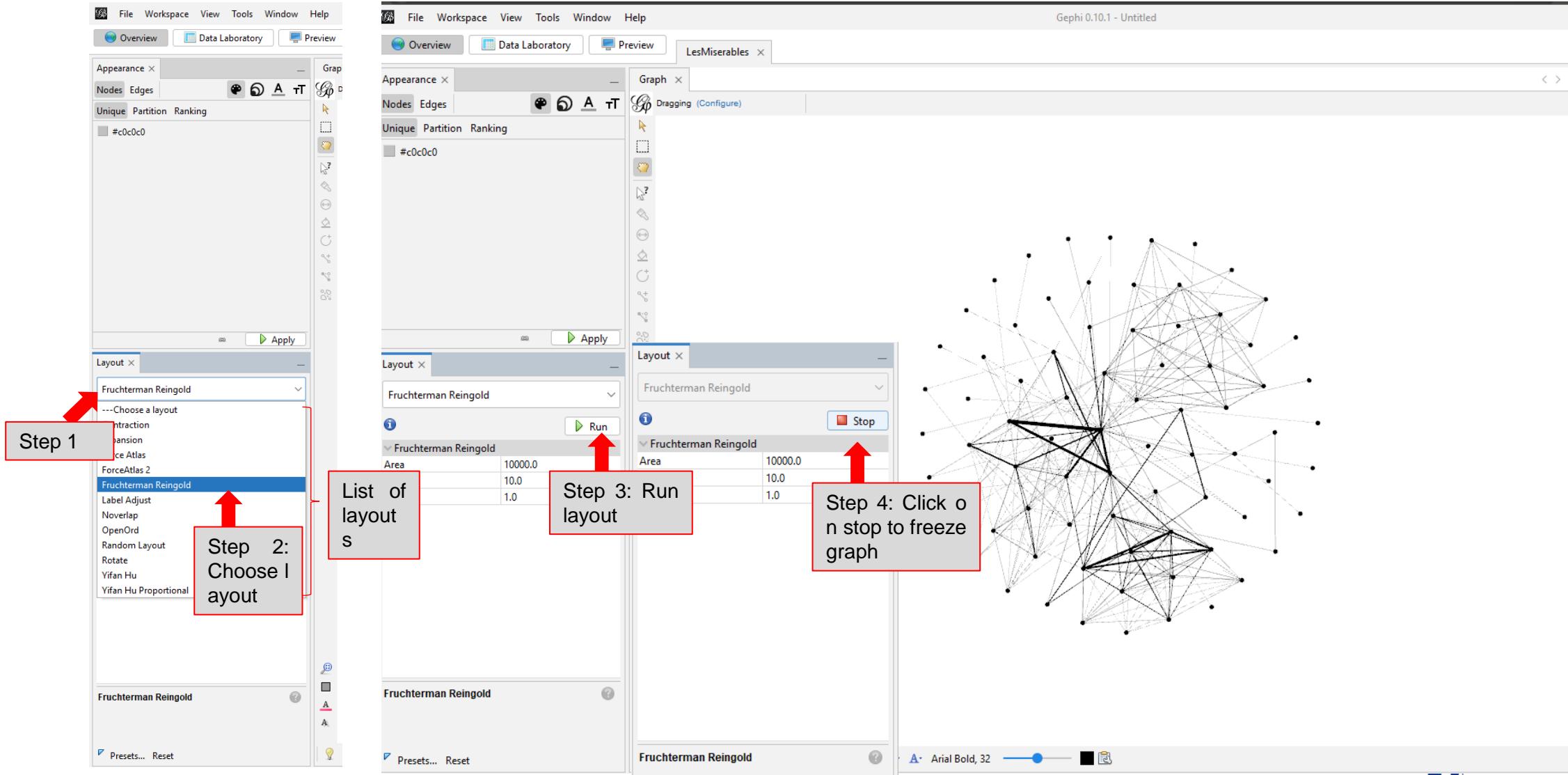
Visualizing a Graph using Gephi

➤ Gephi's interface:



Visualizing a Graph using Gephi

➤ Gephi's layout:



Visualizing a graph using Gephi

➤ Force atlas 2 layout:

The screenshot shows the Gephi interface with the Force Atlas 2 layout selected. On the left, the Layout panel shows various parameters for the layout, including Tolerance (speed), Approximate Repulsion, Approximation, Scaling (set to 10.0), Stronger Gravity, Gravity, and Behavior Alternatives (with Prevent Overlap checked). On the right, the Graph panel displays a network of nodes connected by edges, forming several clusters. Below the Graph panel, there are three red boxes with arrows pointing to specific features in the bottom toolbar:

- Show node labels**: Points to the 'T' icon in the toolbar.
- Adjusting the thickness of the links**: Points to the edge thickness slider in the toolbar.
- Change the size of the labels**: Points to the font size and style selector in the toolbar.

Red boxes on the left side of the interface also highlight specific configuration steps:

- 1. Size: change node size.** Points to the 'Size' dropdown in the Appearance panel.
- 2. Click Apply for changing the node size**. Points to the 'Apply' button in the Layout panel.
- 2. Scaling: a parameter to control how "wide" the graph will be.** Points to the 'Scaling' parameter in the Layout panel.
- 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.** Points to the 'Prevent Overlap' checkbox in the Layout panel.

Force Atlas 2 is a layout which:

1. brings together nodes which are connected

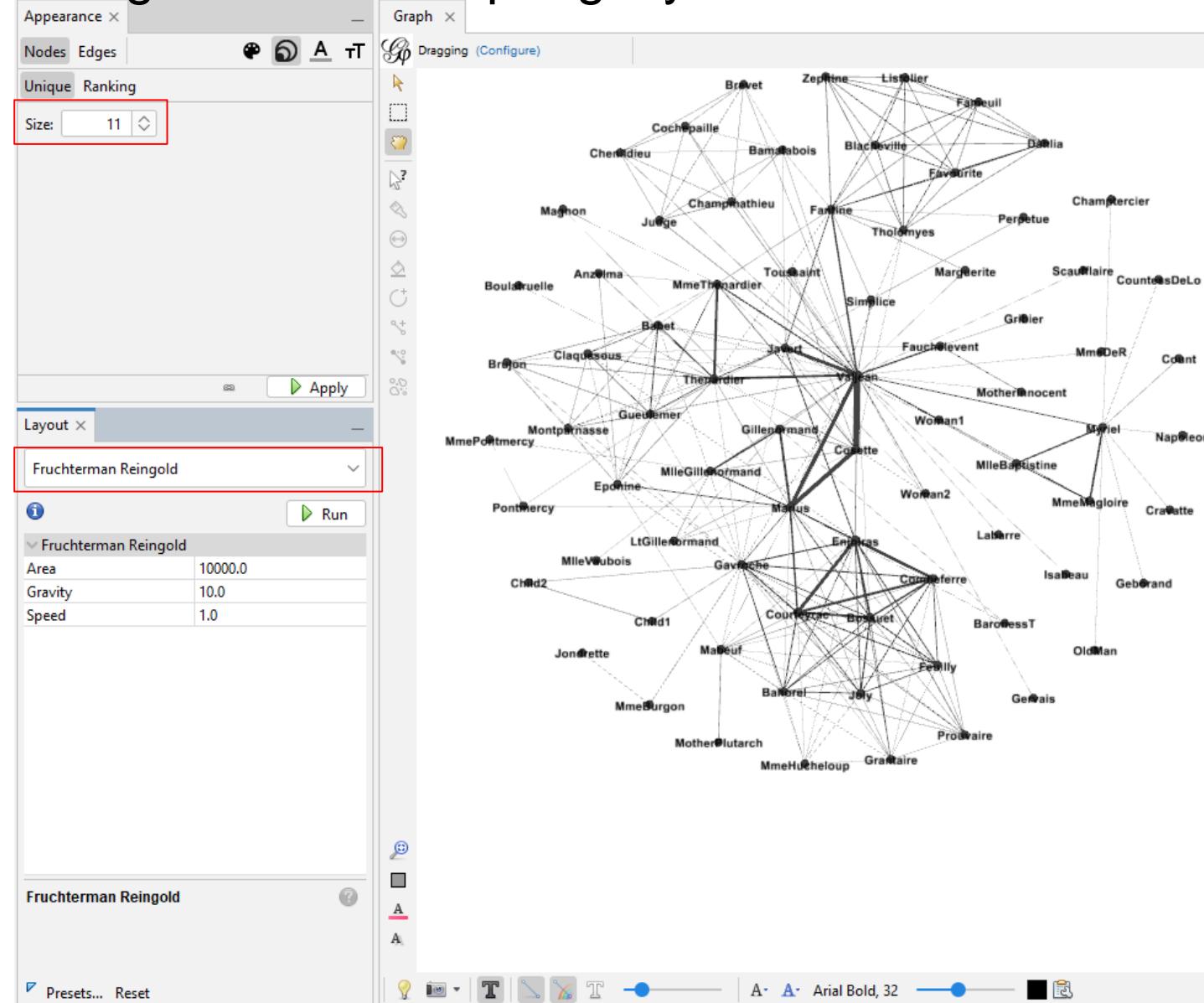
2. spreads apart unconnected nodes

As an effect, we can easily detect communities of nodes

Visualizing a graph using Gephi

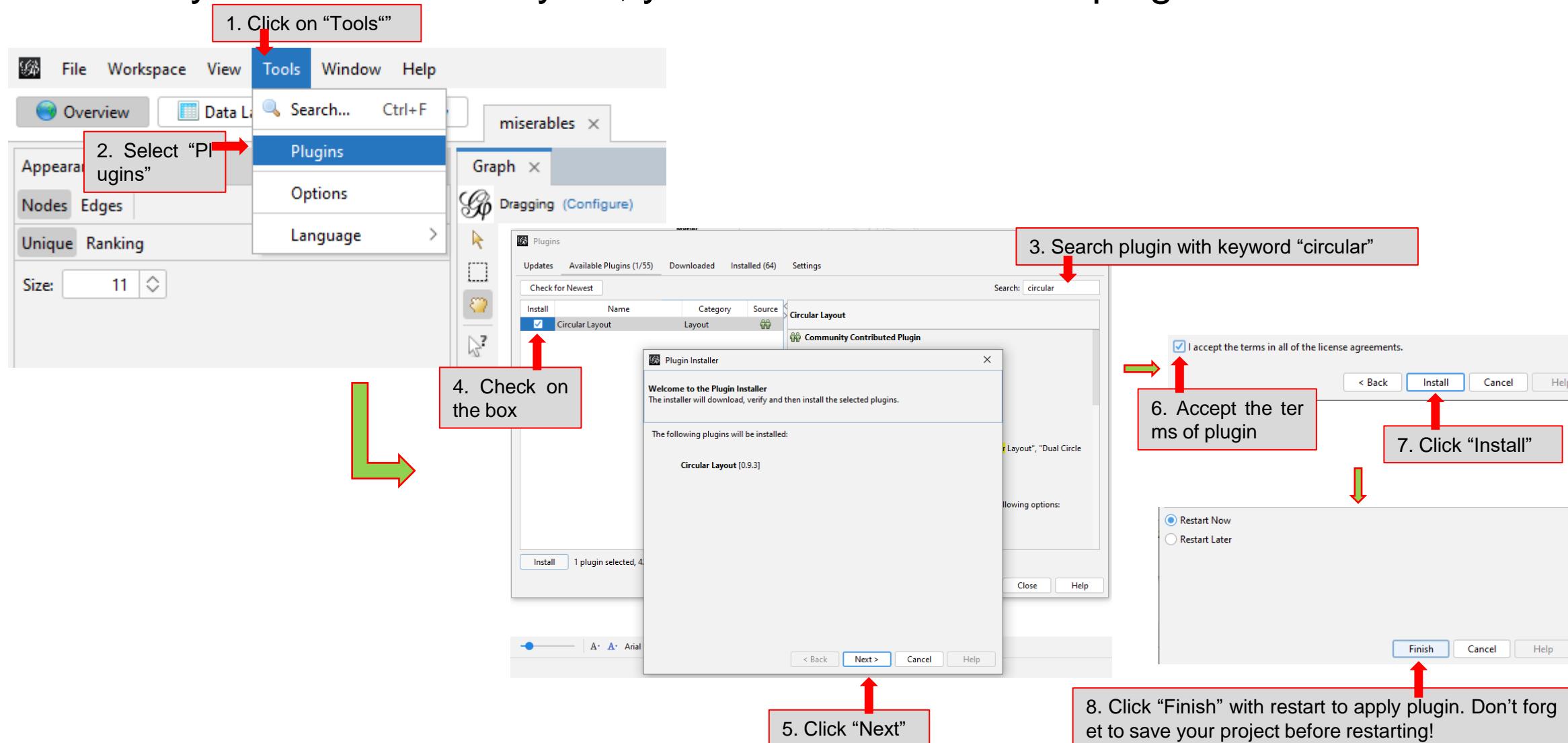
68

- Fruchterman and Reingold relies on spring layout



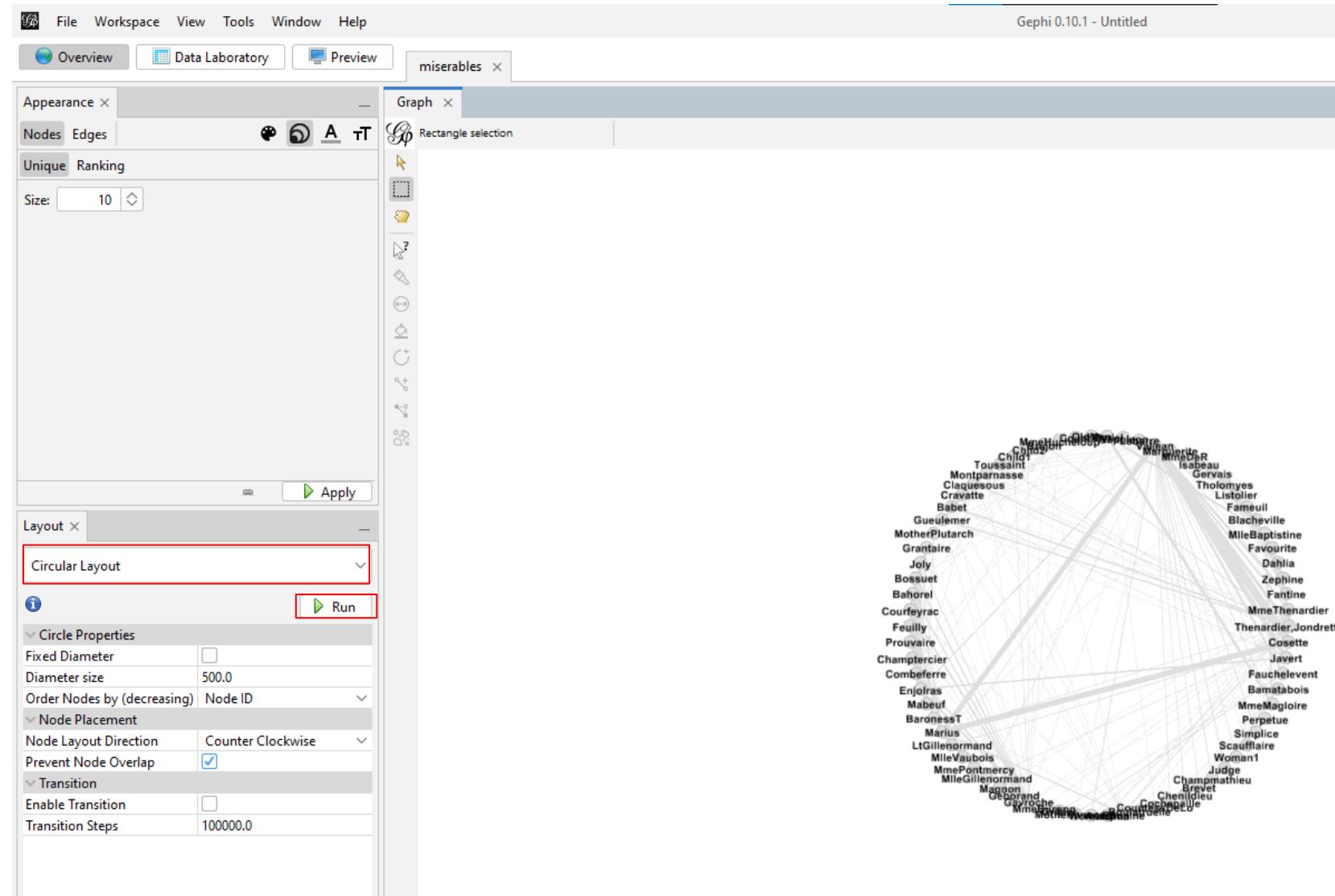
Visualizing a graph using Gephi

➤ Circular layout: to use this layout, you need to install new plugin



Visualizing a graph using Gephi

➤ Circular layout:



Visualizing a graph using Gephi

71

➤ Preview graph

1. Click "Preview"

2. Select "Presets" to display your graph

3. Custom graph style

4. Click on "Refresh" to apply

Export image

Zoom graph

Gephi 0.10.1 - Untitled

File Workspace View Tools Window Help

Overview Data Laboratory Preview miserables

Preview Settings Presets Default Curved

Nodes

- Fixed Border Width
- Border Width: 1.0
- Border Color: custom [231,76,60]
- Opacity: 100.0
- Per-Node Opacity
- Node Labels
- Show Labels: checked
- Font: Arial 10 Plain
- Proportional size: checked
- Color: custom [0,0,0]
- Shorten label
- Max characters: 30
- Outline size: 0.0
- Outline color: custom [255,255,255]
- Outline opacity: 80.0
- Box
- Box color: parent
- Box opacity: 100.0

Edges

- Show Edges: checked
- Thickness: 1.0
- Rescale weight
- Min. rescaled weight: 0.1
- Max. rescaled weight: 1.0
- Color: custom [22,160,133]
- Opacity: 100.0
- Curved: checked
- Radius: 0.0

Edge Arrows

- Size: 0

Edge Labels

- Show Labels: checked
- Font: Arial 10 Plain
- Color: original
- Shorten label
- Max characters: 30
- Outline size: 0.0
- Outline color: custom [255,255,255]

Preview ratio: 100%

Export: SVG/PDF/PNG Refresh

Background Reset zoom - +

miserables

Character network visualization showing connections between various characters from the novel Les Misérables.

Visualizing a graph using Gephi

➤ Switching the view to the data laboratory:

The screenshot shows the Gephi interface with the 'Data Laboratory' tab selected. A red box highlights the 'Data Laboratory' tab in the top navigation bar, with a red arrow pointing to it from the left. Another red box highlights the 'Label' column in the data table, with a red arrow pointing to it from below. A third red box highlights the vertical axis on the left, with a red arrow pointing to it from the left.

Click on "Data laboratory"

Node labels

Each row contains one node information

	Label	Interval
0.0	Myriel	
1.0	Napoleon	
2.0	MlleBaptistine	
3.0	MmeMagloire	
4.0	CountessDeLo	
5.0	Geborand	
6.0	Champtercier	
7.0	Cravatte	
8.0	Count	
9.0	OldMan	
10.0	Labarre	
11.0	Valjean	
12.0	Marguerite	
13.0	MmeDeR	
14.0	Isabeau	
15.0	Gervais	
16.0	Tholomyes	
17.0	Listolier	
18.0	Fameuil	
19.0	Blacheville	
20.0	Favourite	
21.0	Dahlia	
22.0	Zephine	
23.0	Fantine	
24.0	MmeThenardier	
25.0	Thenardier	
26.0	Cosette	
27.0	Javert	
28.0	Fauchelevent	
29.0	Bamatabois	
30.0	Perpetue	
31.0	Simplice	
32.0	Scauflaire	
33.0	Woman1	
34.0	Judge	
35.0	Chenushkin	

Below the table are several action buttons: Add, Merge, Delete, Clear, Copy data to, Fill column, Duplicate, Create a boolean column, Create column with list of, and Negate.

Visualizing a graph using Gephi

➤ Computing betweenness centrality with Gephi:

The screenshot shows the Gephi 0.10.1 interface with the following steps highlighted:

- 1. Click on "Statistics"**: A red box highlights the "Statistics" tab in the top right corner of the Context panel.
- 2. Run "Network Diameter"**: A red box highlights the "Run" button next to the "Network Diameter" item in the Statistics panel.
- 3. Click OK**: A red box highlights the "OK" button in the "Graph Distance settings" dialog.
- 4. Close**: A red box highlights the "Close" button in the bottom left corner of the "Betweenness Centrality Distribution" chart window.

Context Panel (Right Side):

- Nodes: 74
- Edges: 248
- Undirected Graph
- Filters
- Statistics
- Settings
- Network Overview
- Diameter: 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

Graph Distance Settings Dialog (Center):

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**

Betweenness Centrality Distribution Chart (Bottom Left):

Results:

- Diameter: 5
- Radius: 3
- Average Path length: 2.587189929655683

Betweenness Centrality Distribution

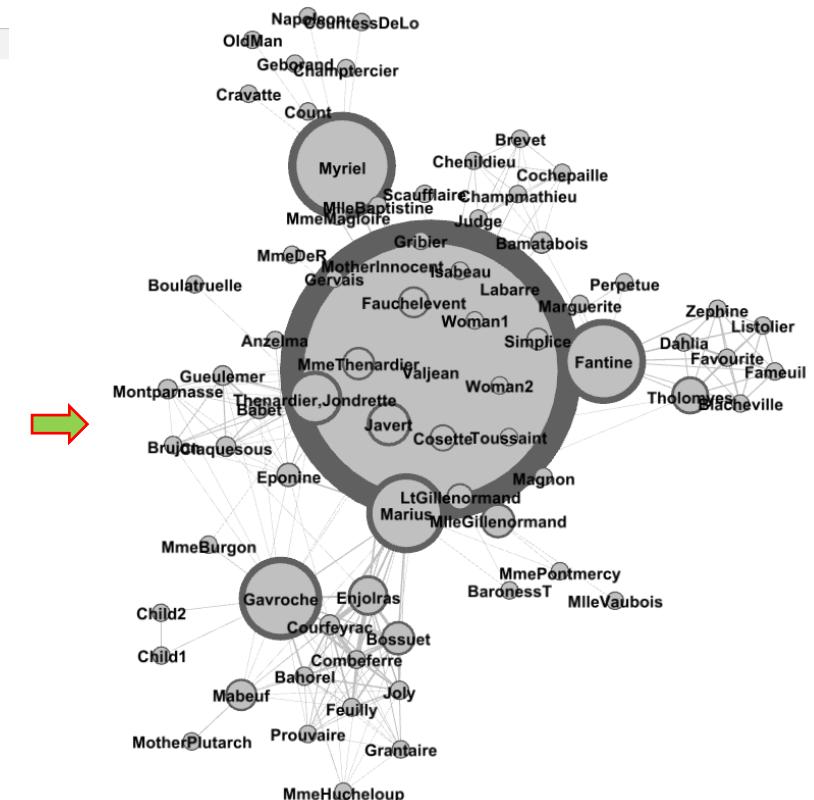
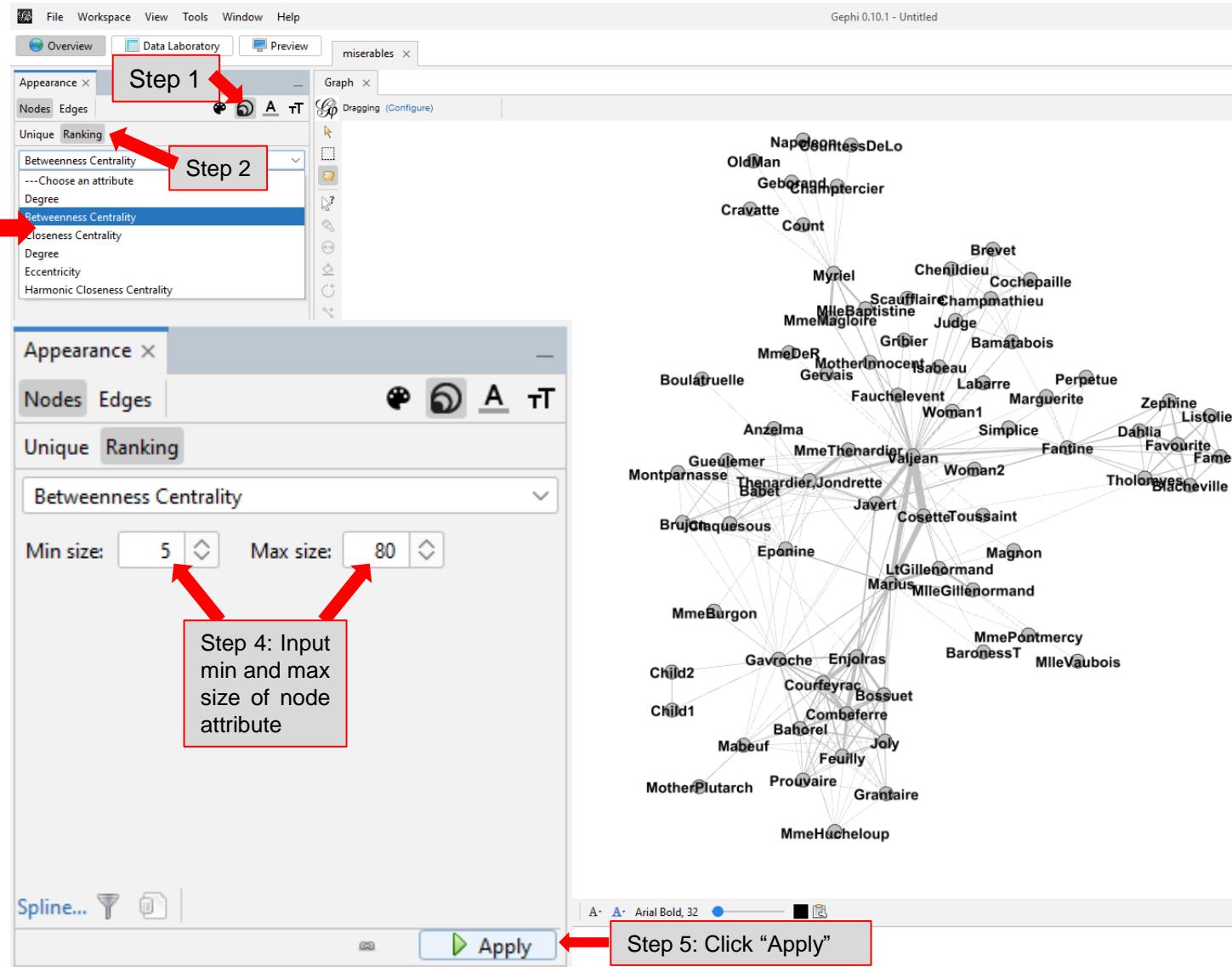
Count

Value

Print Copy Save

Visualizing a graph using Gephi

➤ View graph attribute: Betweenness Centrality





네트워크 과학 연구실
NETWORK SCIENCE LAB



가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA

