Graph Database and Visualization

Prof. O-Joun Lee

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

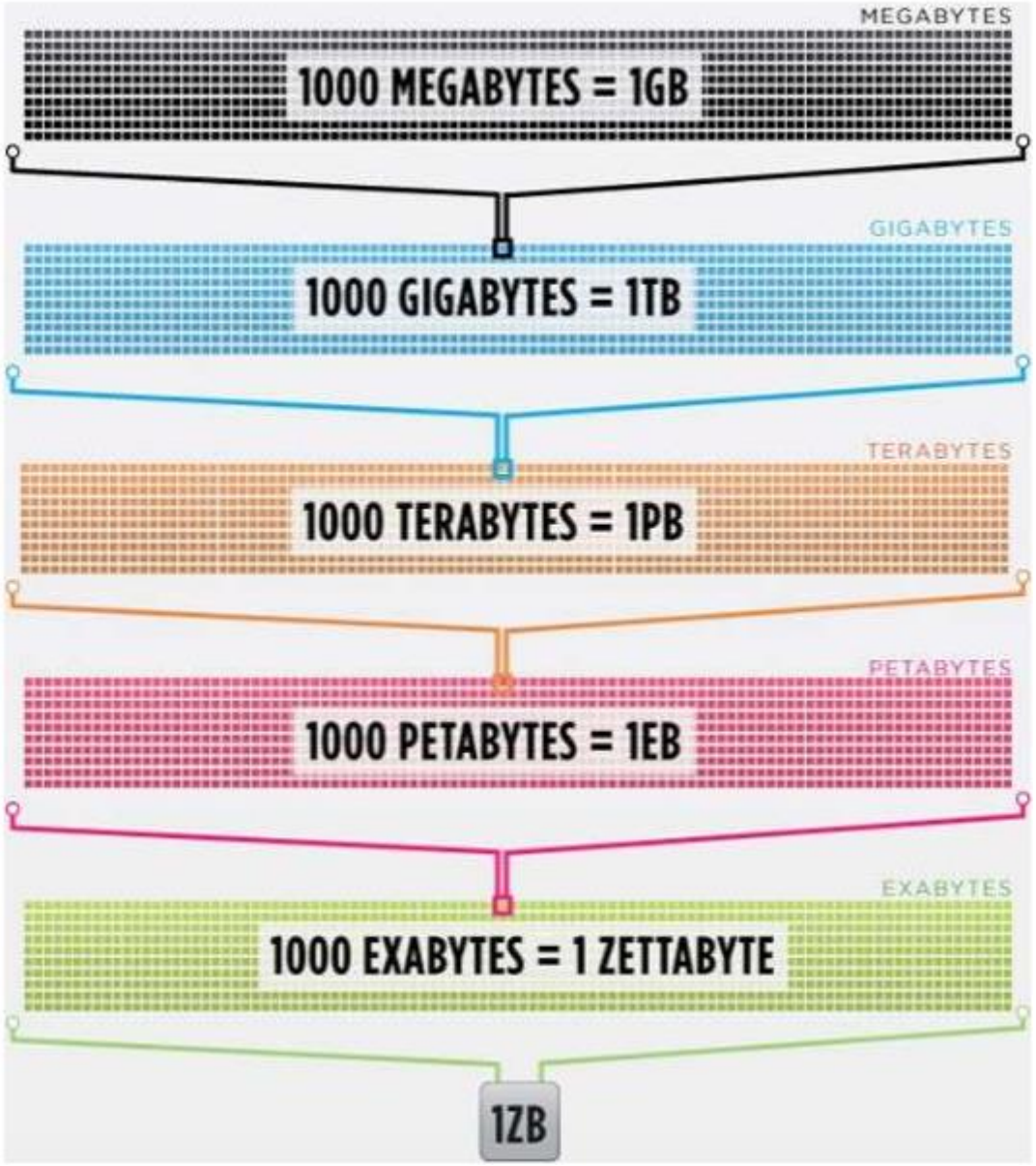
네트워크과학연구실 NETWORK SCIENCE LAB

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.

Trends in data



- **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.

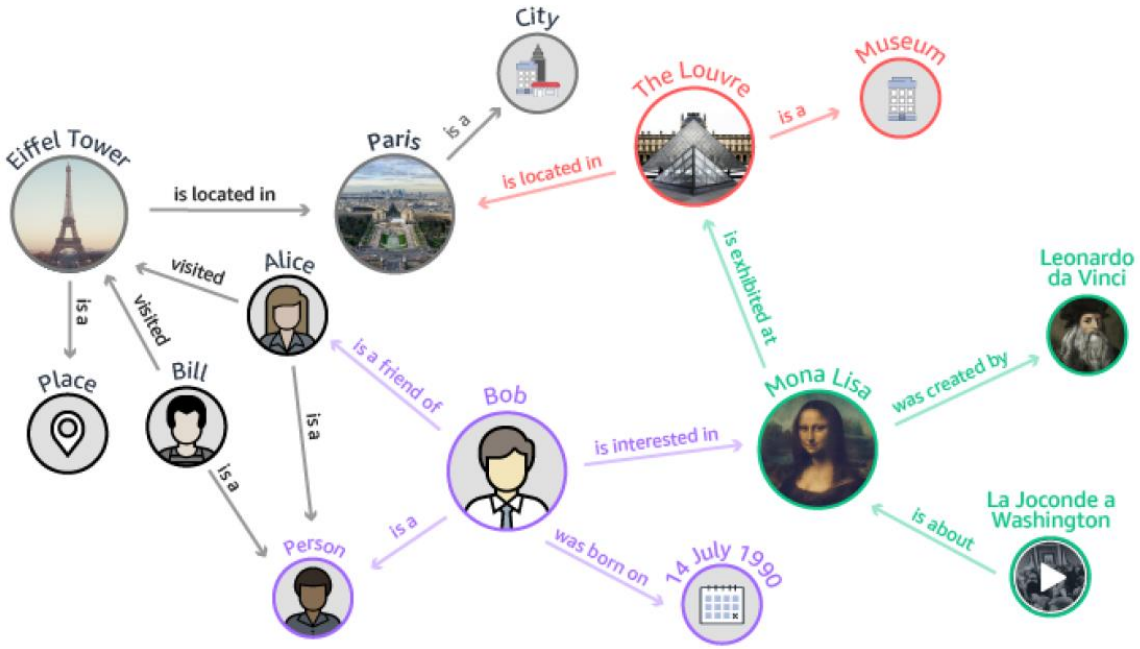
IMDb

⇒ Graphs are important to model and represent those relational features.

Source: IMDB and Netflix Korea.

What is a Graph Database?

- 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.



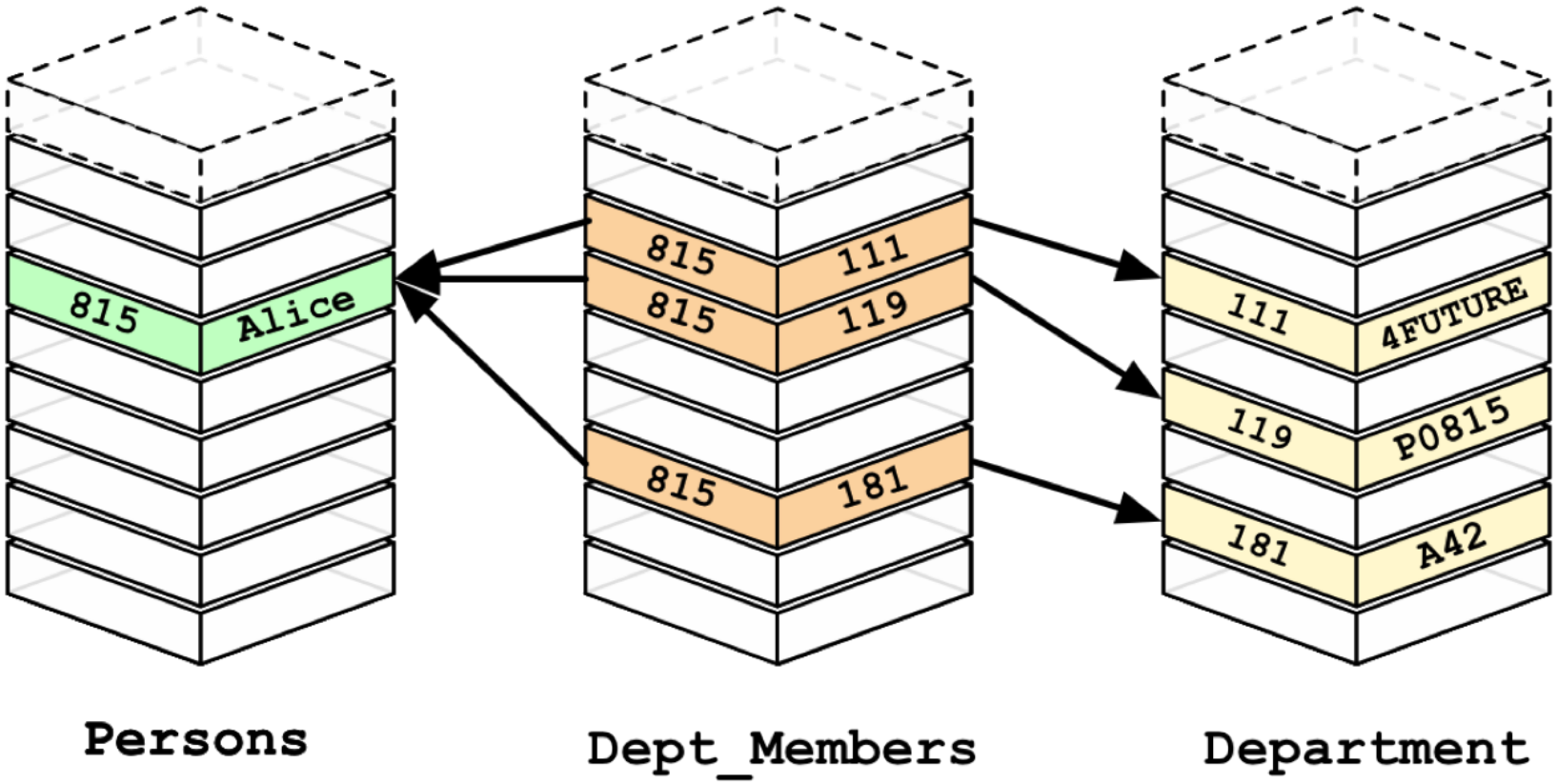
https://olab.ca/heterogeneous-graphs-and-graph-neural-networks/

Graph Database Characteristic

- 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.

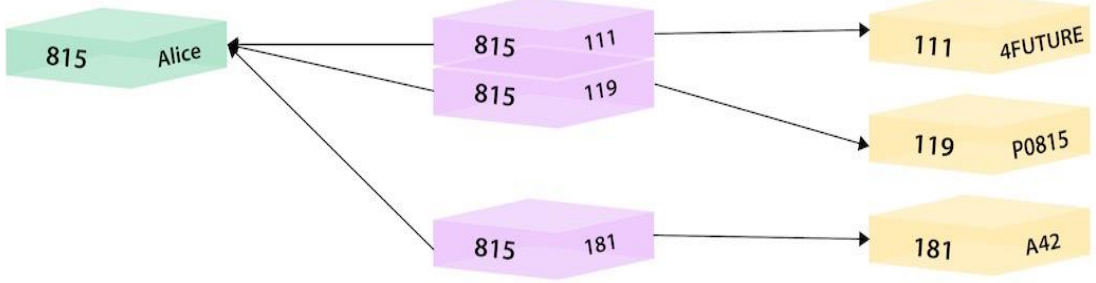
Graph Databases: An example

- 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?

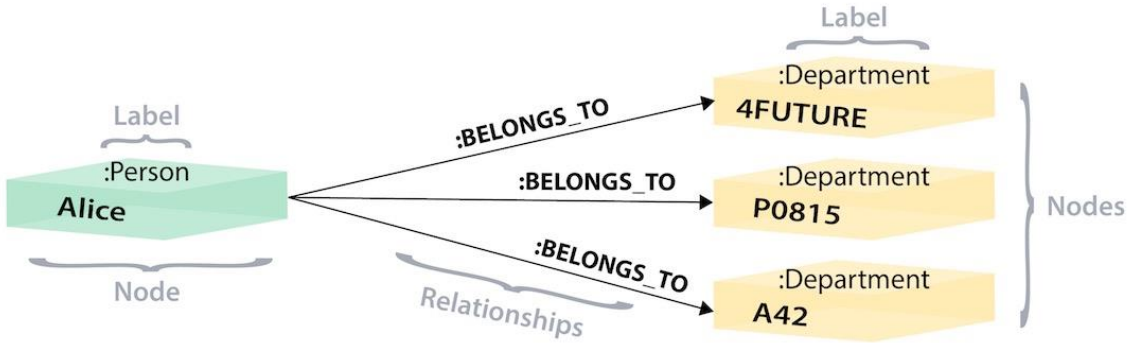


Graph Databases: An example

- 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.

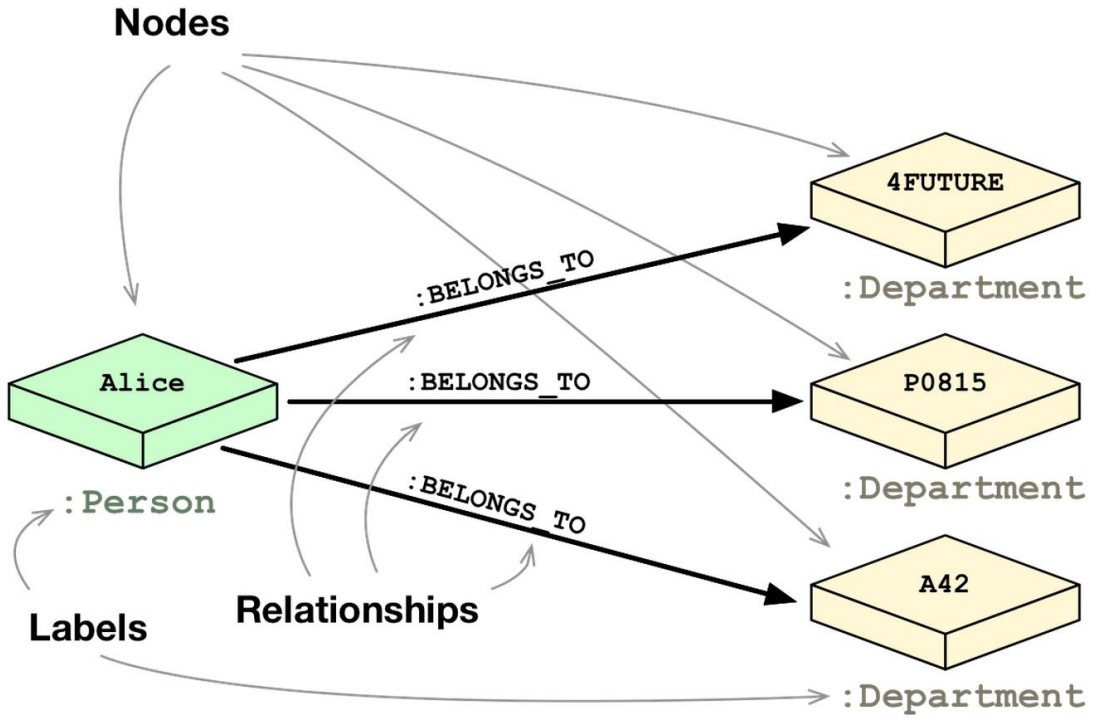


Relational database



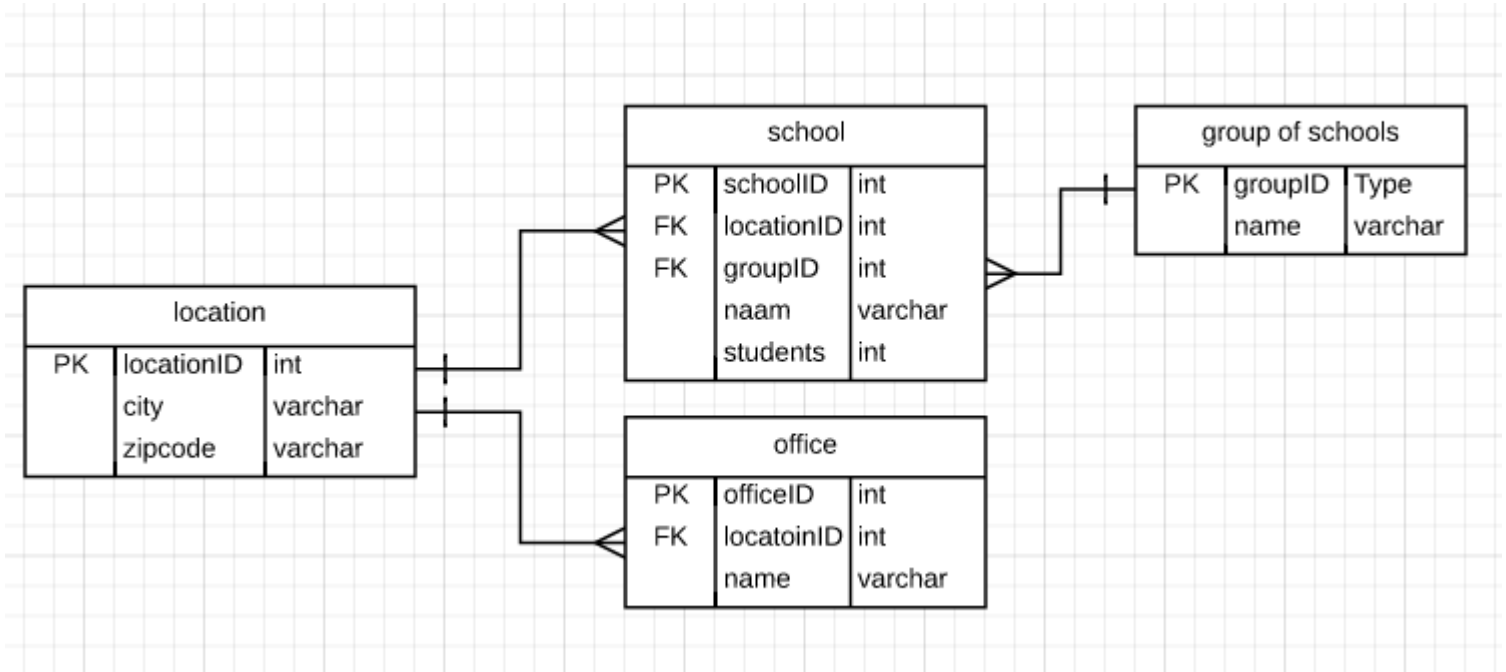
Graph database

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



Relational Databases

- 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

What are difference between Graph DB and Relational DB

- 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.

Popular graphdb Engines

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 & cons

Azure

- **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

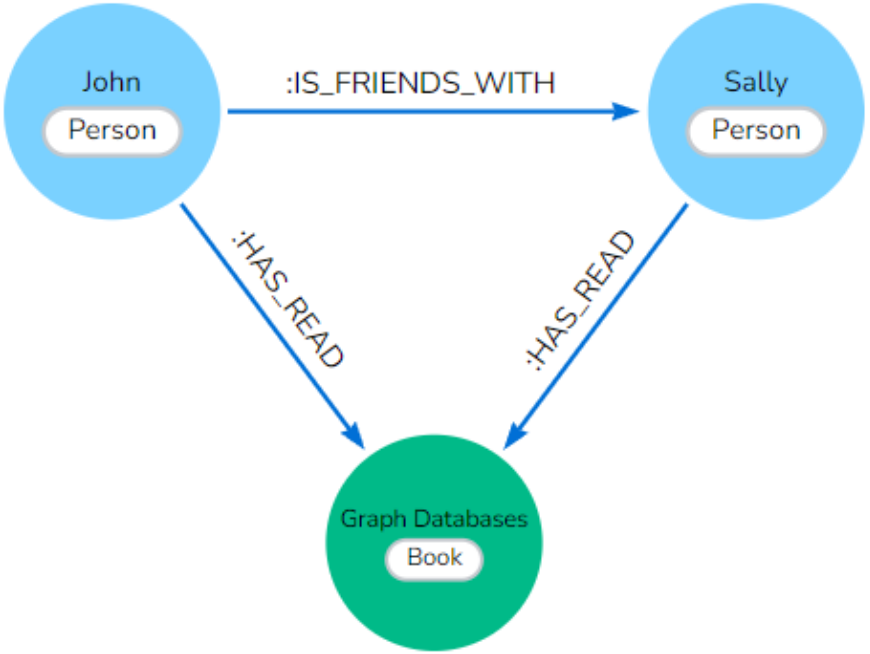
- **Cons:**
  - Only one Project can be running on one port at a time

What does Neo4j provide?

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

You can’t model it in SQL

- 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.

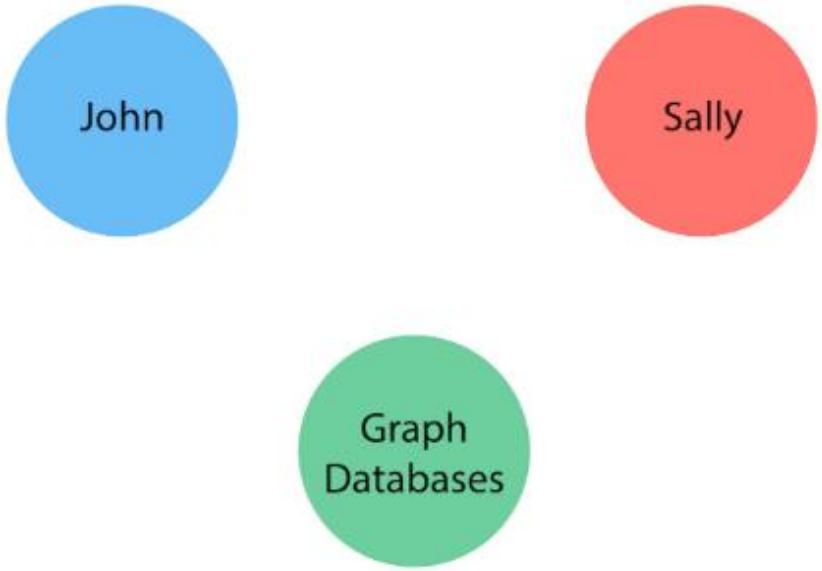


- Two people, Sally and John, are friends.
- Both John and Sally have read the book, Graph Databases.

https://neo4j.com/developer/guide-data-modeling/

Neo4J Model: Example

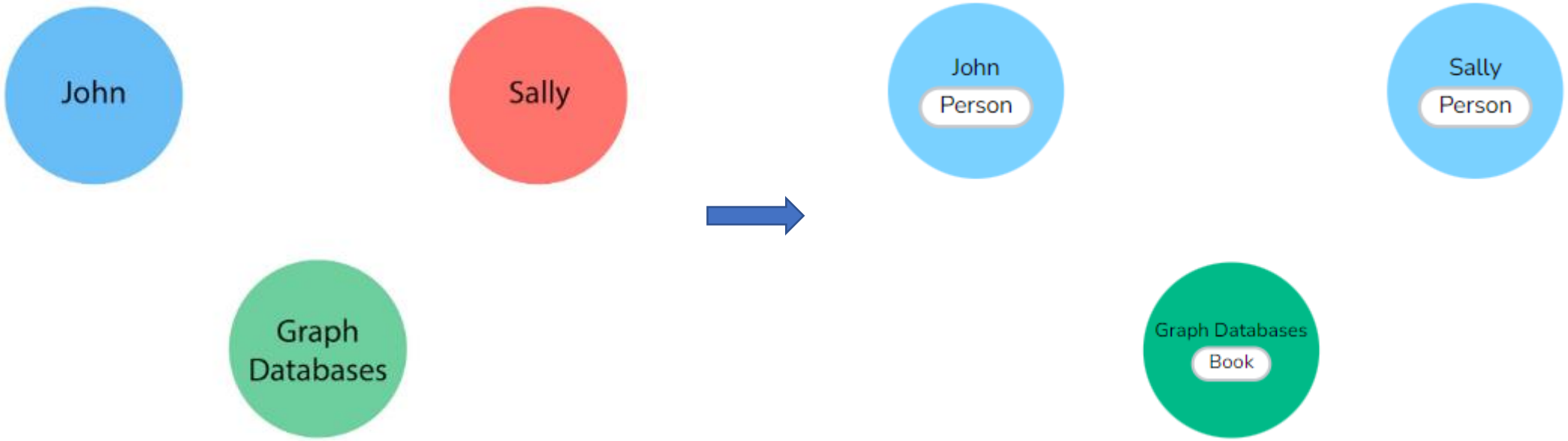
- Two *people*, **Sally** and **John**, are friends. Both **John** and **Sally** have read the *book*, **Graph Databases**.
  - Extracting the nodes: John, Sally, Graph Databases



https://neo4j.com/developer/guide-data-modeling/

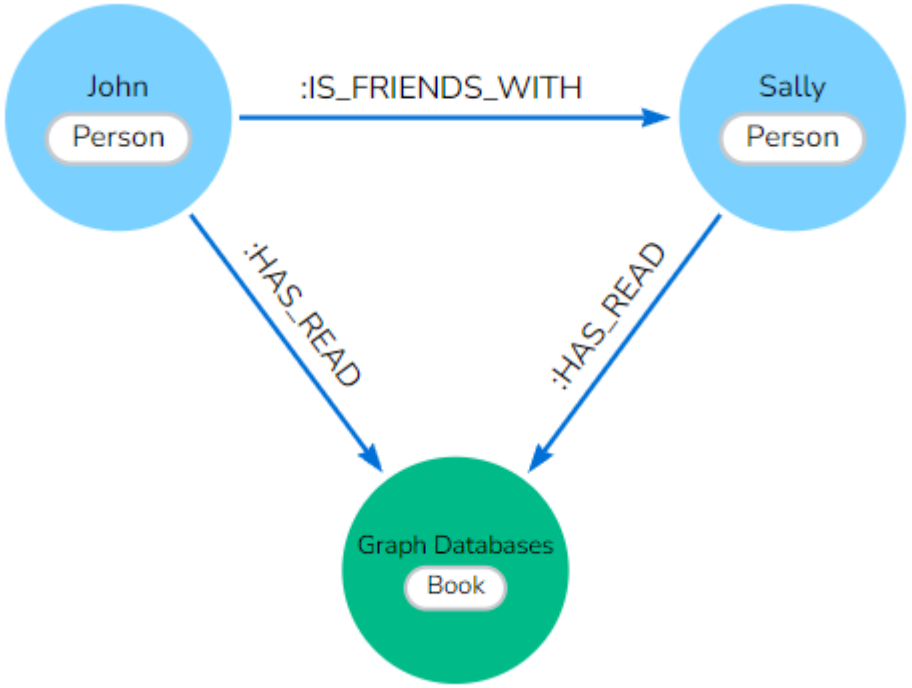
Neo4J Model: Example

- 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



https://neo4j.com/developer/guide-data-modeling/

- 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



https://neo4j.com/developer/guide-data-modeling/

Example: Neo4j database

Neo4j Installation.

Example: Neo4j database

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

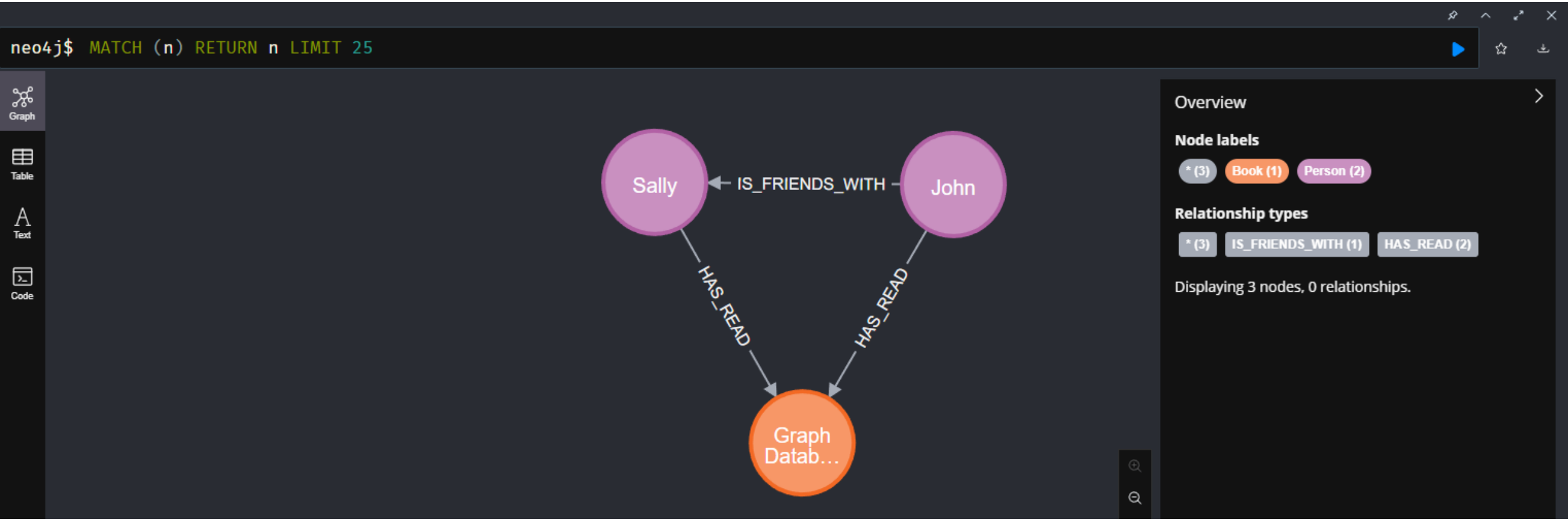
```
CREATE(:Person{name:'John',born:'Mar 8, 1998',linkedin:'@john'})
CREATE(:Person{name:'Sally',born:'Oct 16, 1997',linkedin:'@sali'})
CREATE(:Book{name:'Graph Databases',published_date:'Nov 16, 2015'})
```

  - 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);
```

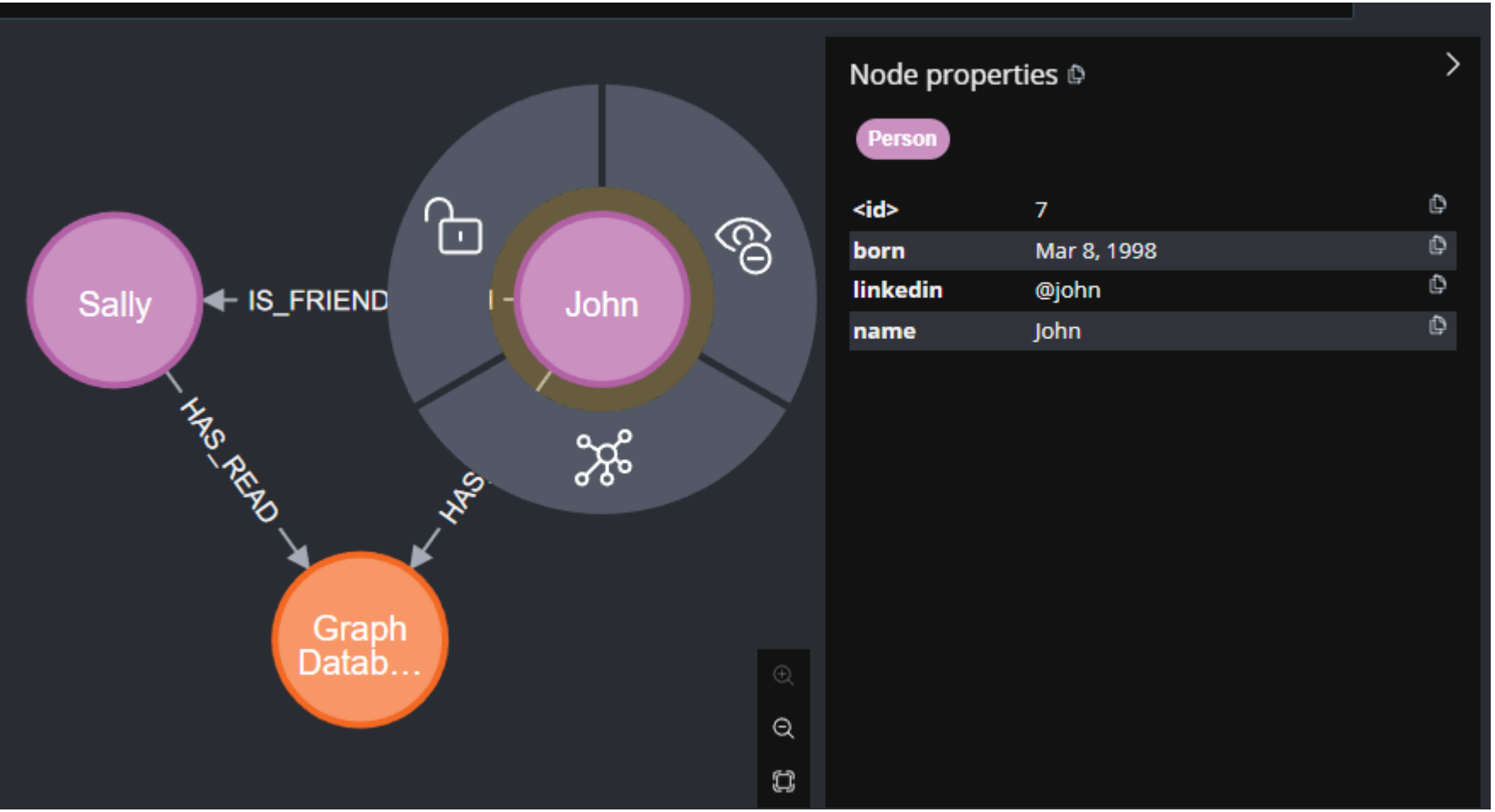
Example: Neo4j database

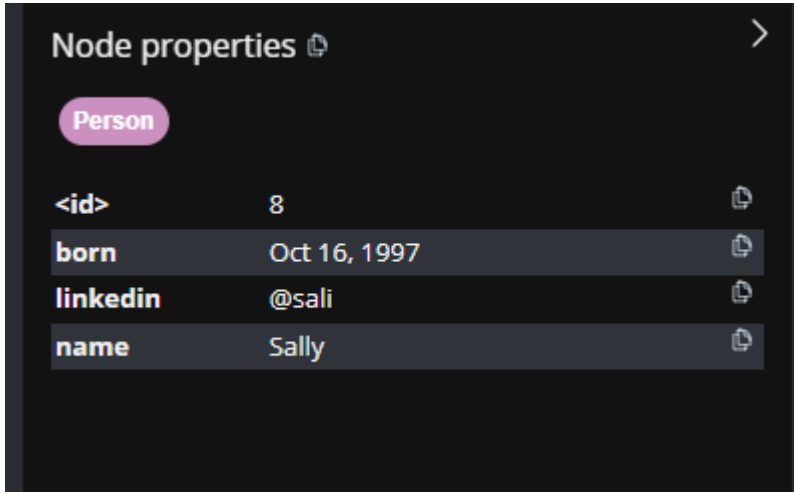
- Show database:

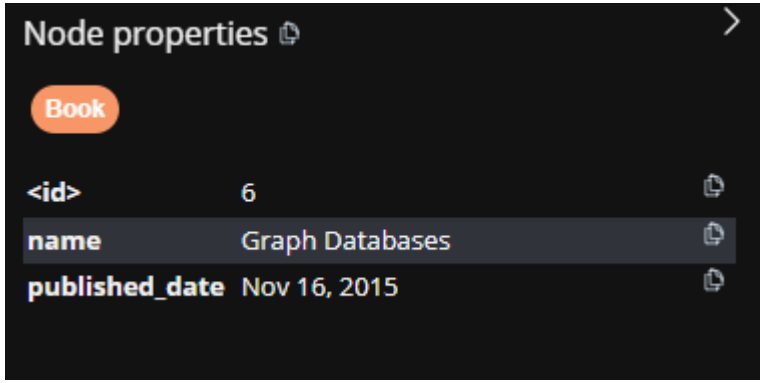


Example: Neo4j database

- Node properties:







Example: Neo4j database

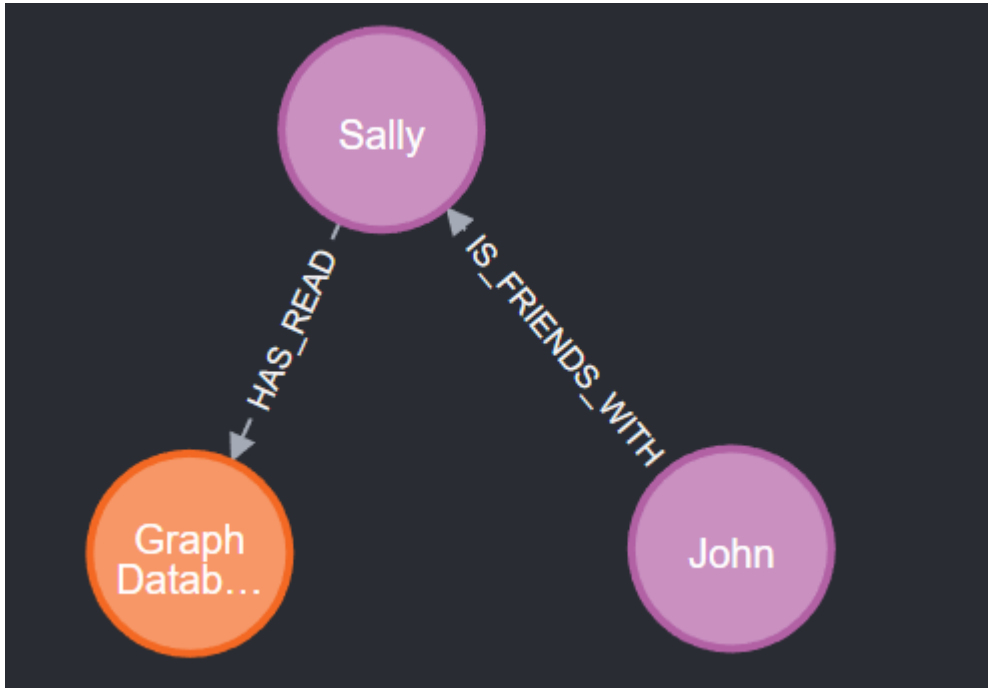
- Delete relationships:

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

Table

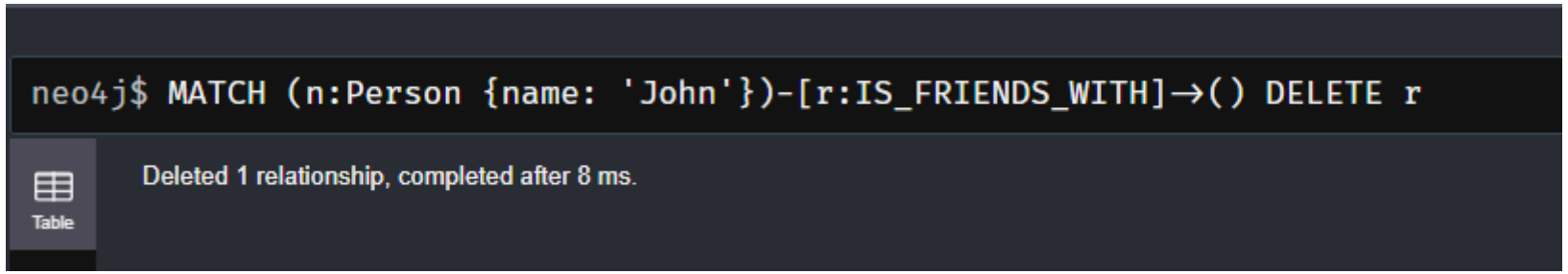
Deleted 1 relationship, completed after 17 ms.

- Show database:



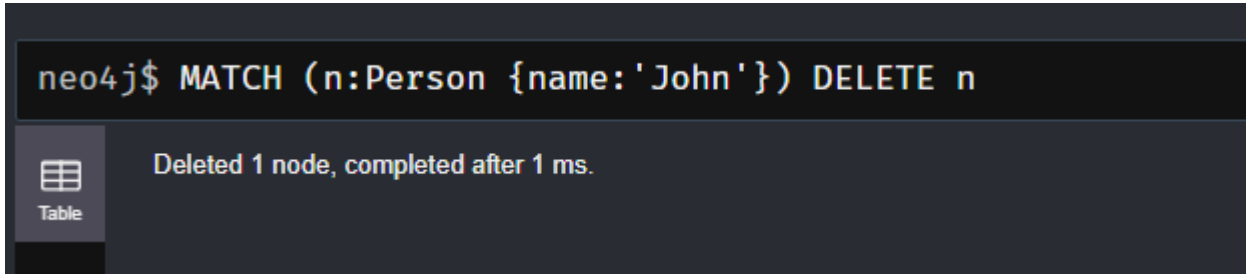
Example: Neo4j database

- 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.



Example: Neo4j database

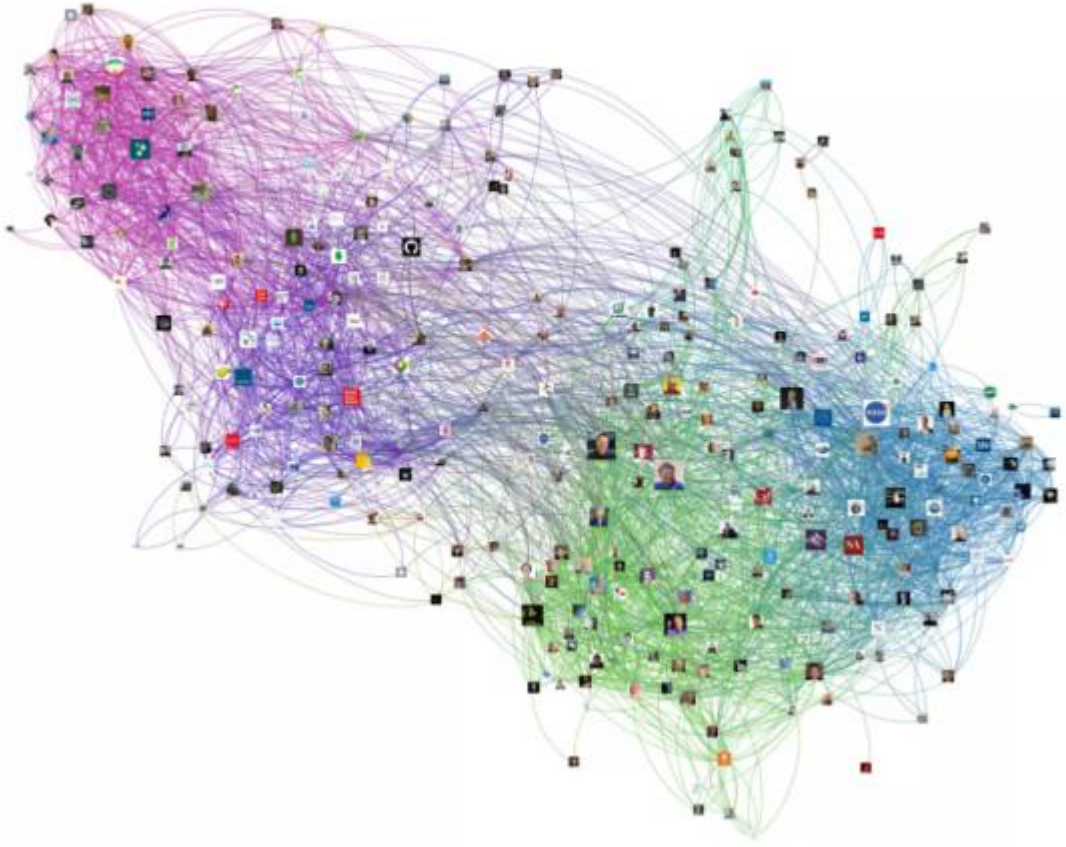
- Remove a node:

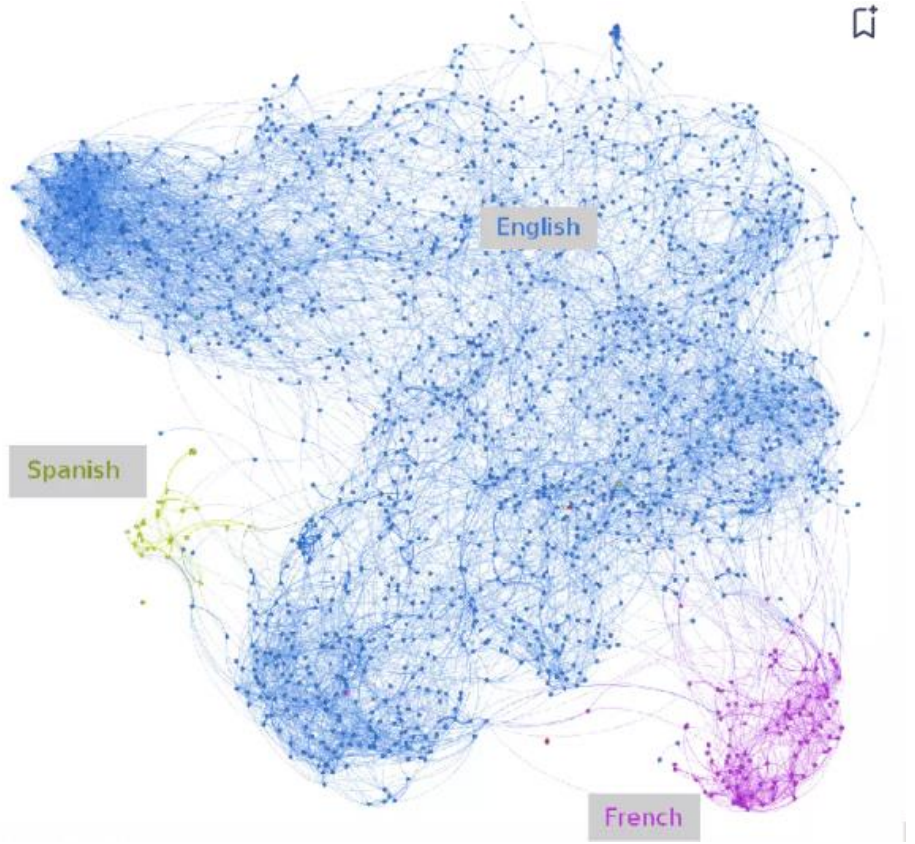


- Show database:

What is Visualization?

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

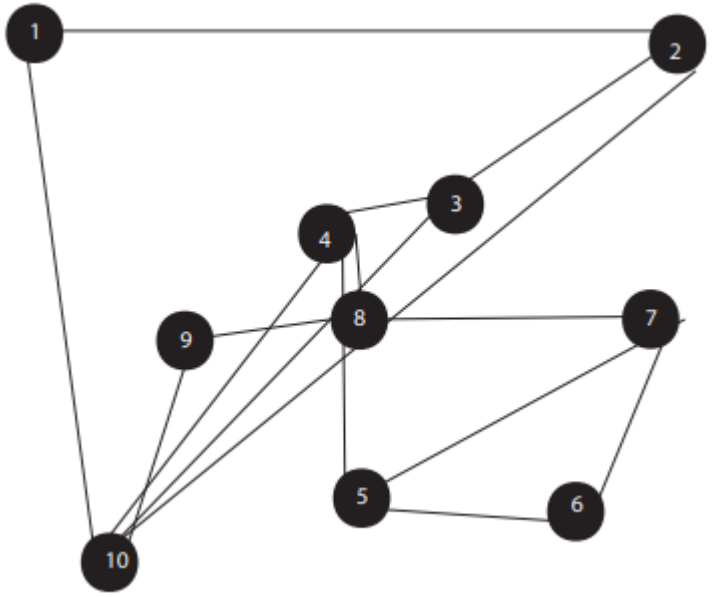




Source: Neo4j.

Why are Graph Visualization important?

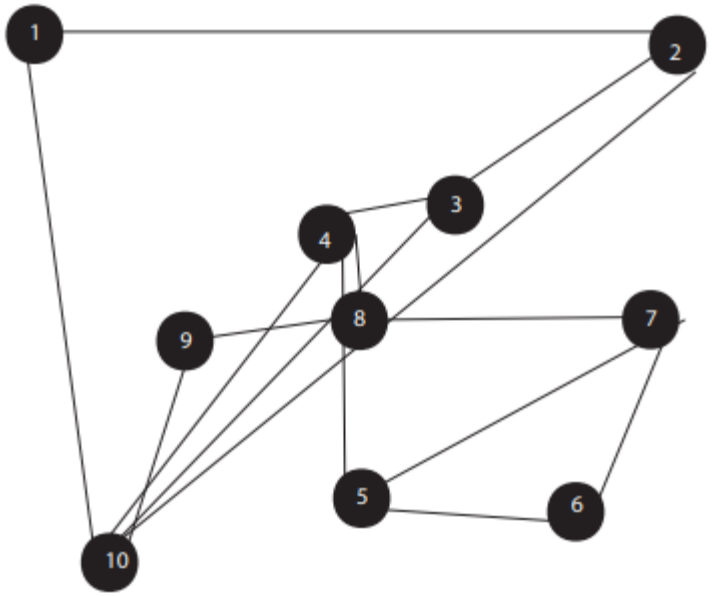
- Input: $Graph\ G\ =\ (V, E)$

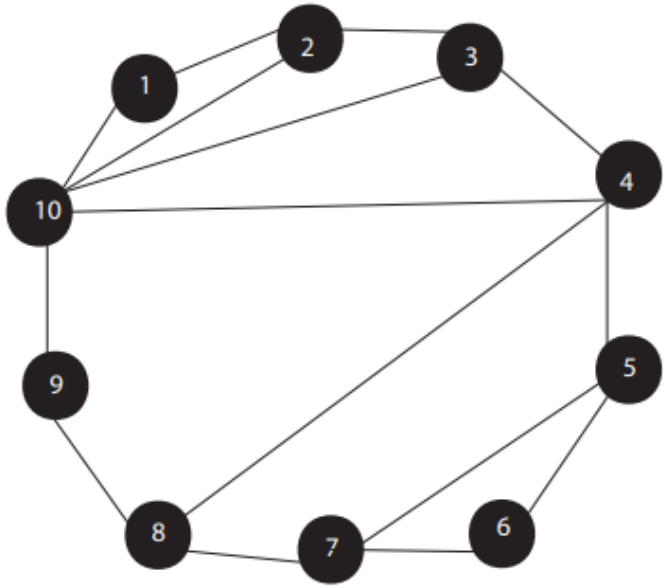


https://www.csd.uoc.gr/~hy583/reviewed_notes/circular.pdf

Why are Graph Visualization important?

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



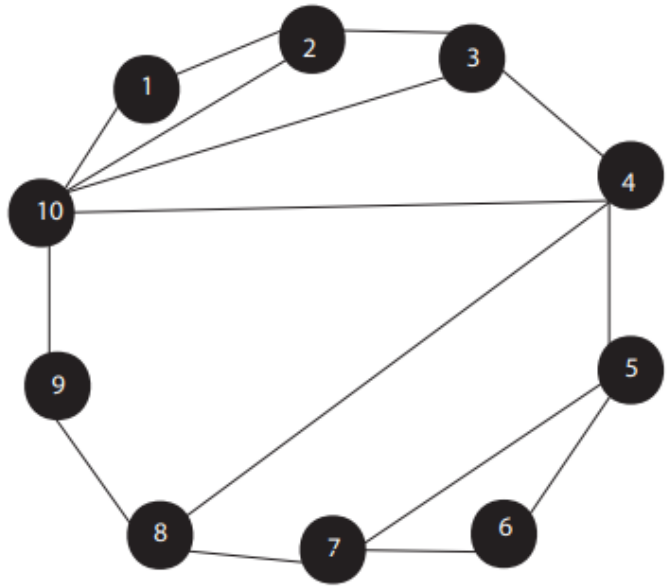


→ Which criteria would you optimize?

https://www.csd.uoc.gr/~hy583/reviewed_notes/circular.pdf

General Layout Problem

- 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.



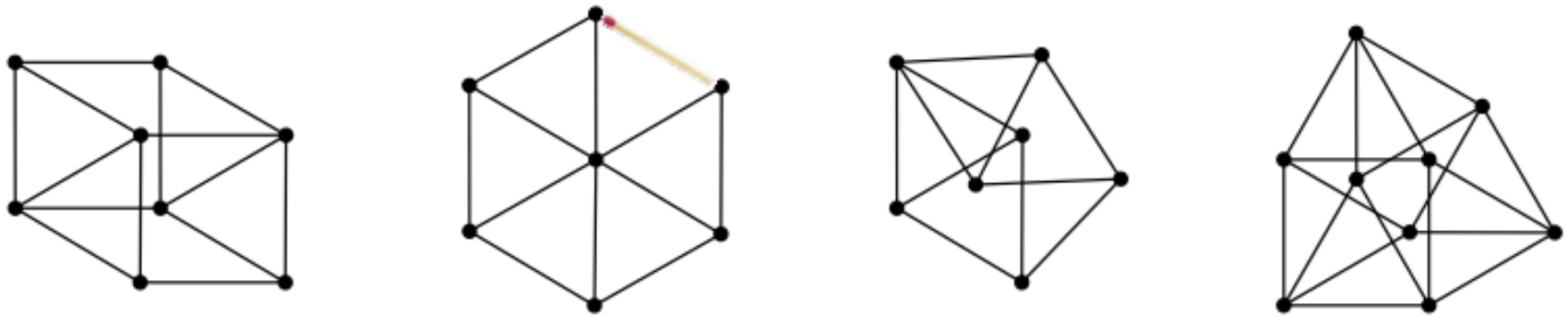
General Layout Problem

- 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

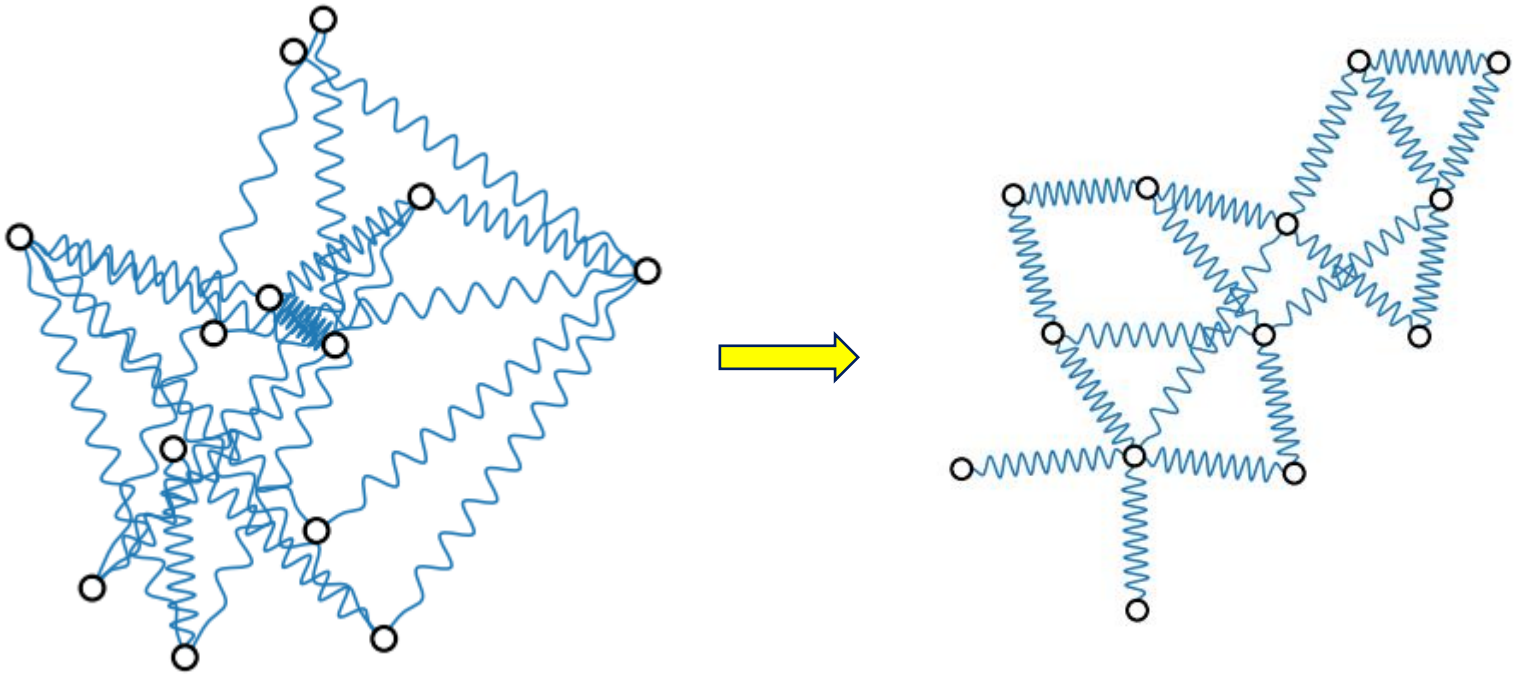
Fixed edge lengths?

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



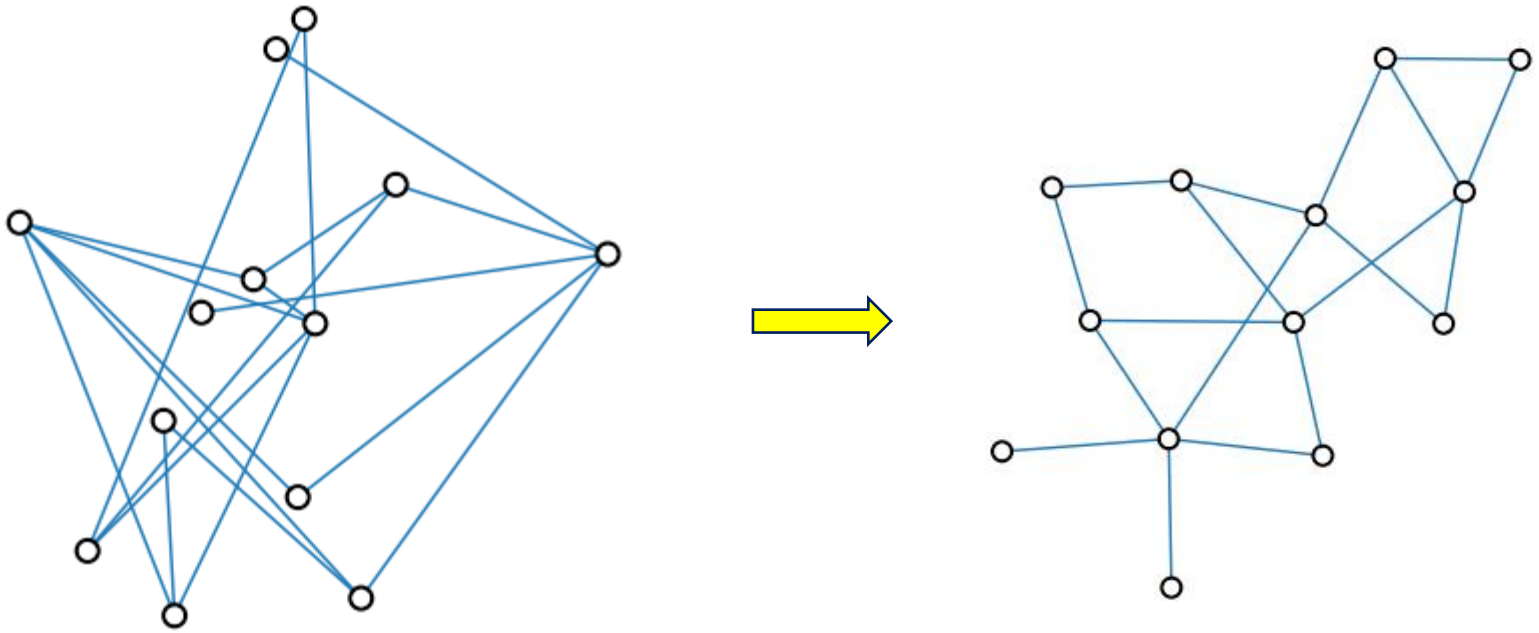
- NP-hard problem for:
  - Uniform edge lengths in any dimension.
  - Uniform edge lengths in planar drawing.

- 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.



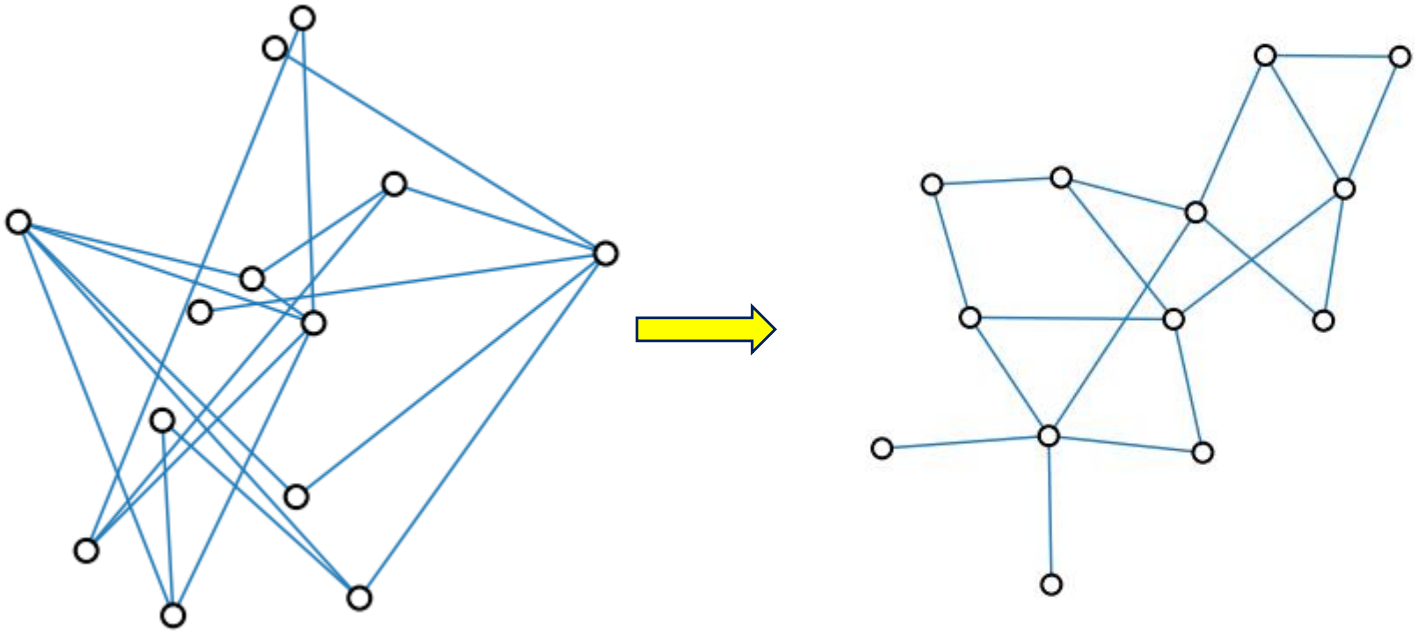
Spring-embedded: The main idea

- 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.



Spring-embedded: idea, constraints

- 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}

Spring-embedded by Eades – main functions

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

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

- Attractive force between two adjacent vertices v_i and v_j

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

- 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.
- $\vec{p_i p_j}$: 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

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
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}

[Eades 1984] A heuristic for graph drawing

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}

[Eades 1984] A heuristic for graph drawing

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* $\boldsymbol{MAX}_{v_i \in V} ||F_i(t)|| > \epsilon$ **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

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)

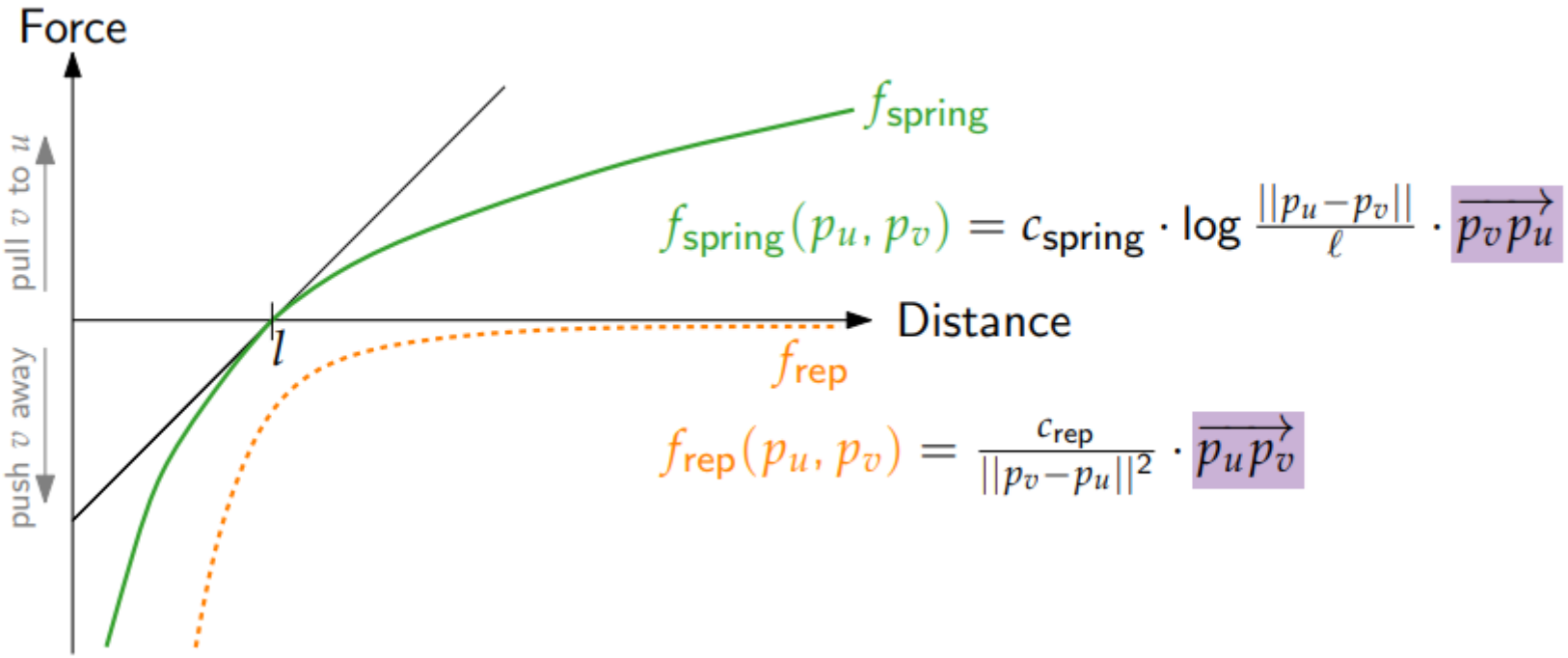
$$f_{rep}(p_i, p_j) = \frac{c_{rep}}{||p_i - p_j||^2} \cdot p_i\vec{p}_j$$

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

[Eades 1984] A heuristic for graph drawing

Spring-embedded by Eades – Force diagram

- 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)



Spring-embedded by Eades – Discussion

- 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

$$\mathrm{f}_{rep}(p_i, p_j) = \frac{l}{||p_i - p_j||^2} \cdot \vec{p_i p_j}$$

- Attractive force between two adjacent vertices v_i and v_j

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

- 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

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

Input: p: initial layout, ϵ: threshold
Output: p: is end layout

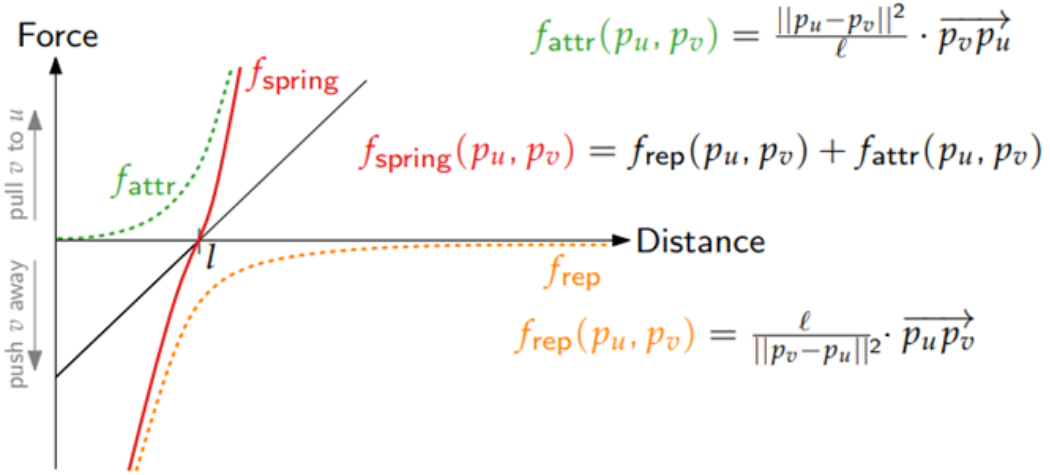
$t \leftarrow 1$
while $t < K$ *and* $\boldsymbol{MAX}_{v_i \in V} ||F_{i(t)}|| > \epsilon$ **do**
  for $v \in V$ **do**
    $F_i(t) \leftarrow \sum_{(v_i, v_j) \notin E} f_{rep}(p_j, p_i) + \sum_{(v_i, v_j) \in E} f_{spring}(p_j, p_i)$
  for $v \in V$ **do**
    $p_i \leftarrow p_i + \delta(t) \cdot F_{i(t)}$
  $t \leftarrow t + 1$
Return p

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

Variant by Fruchterman & Reingold

There are three forces:

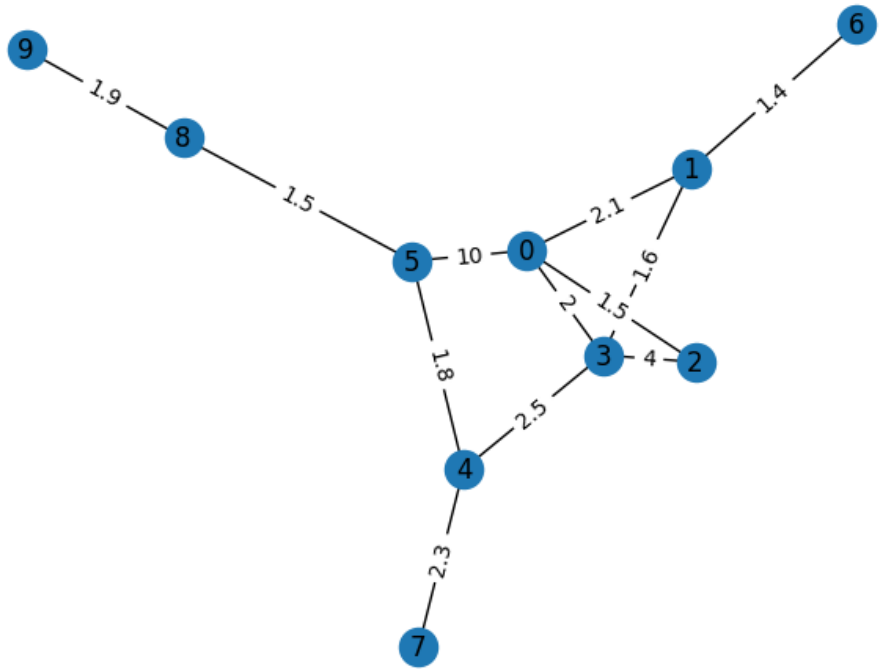
- Spring forces (f_{spring}): pull node v close to node u (u and v are adjacent).
- Attractive force between two adjacent nodes v_i and v_j (f_{attr}): pull node v close to node u (u and v are adjacent).
- Repulsive forces (f_{rep}): push node v faraway node u (u and v are non-adjacent).



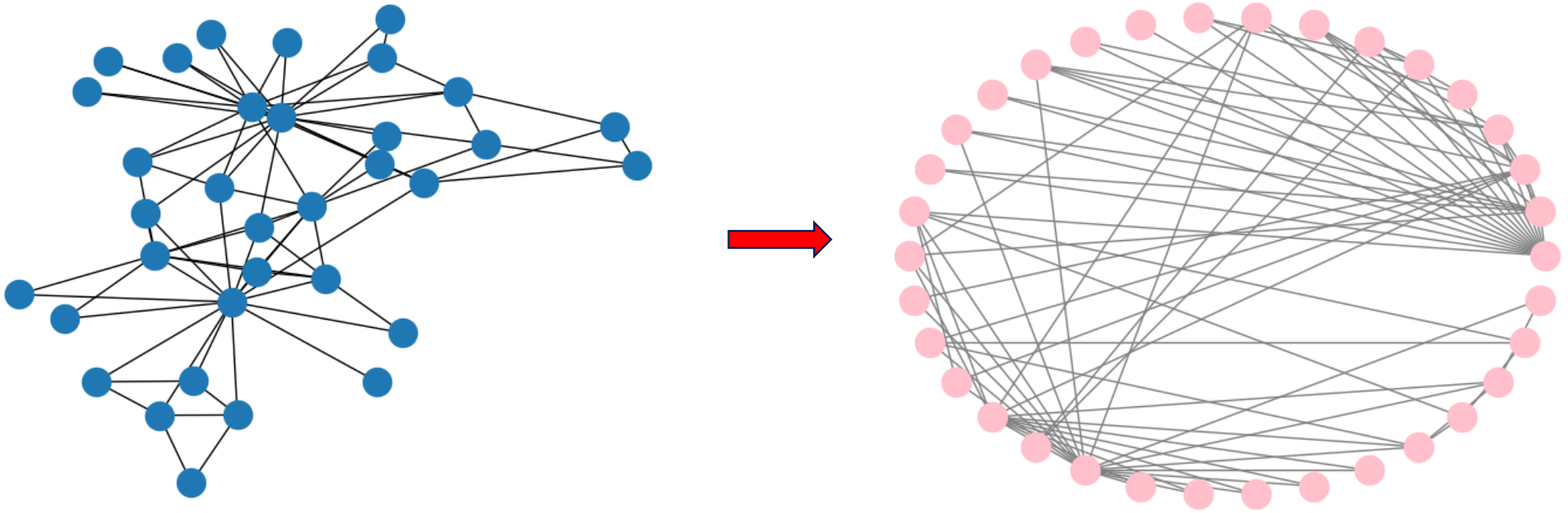
Sample code: Visualizing a graph using NetworkX

- Spring layout

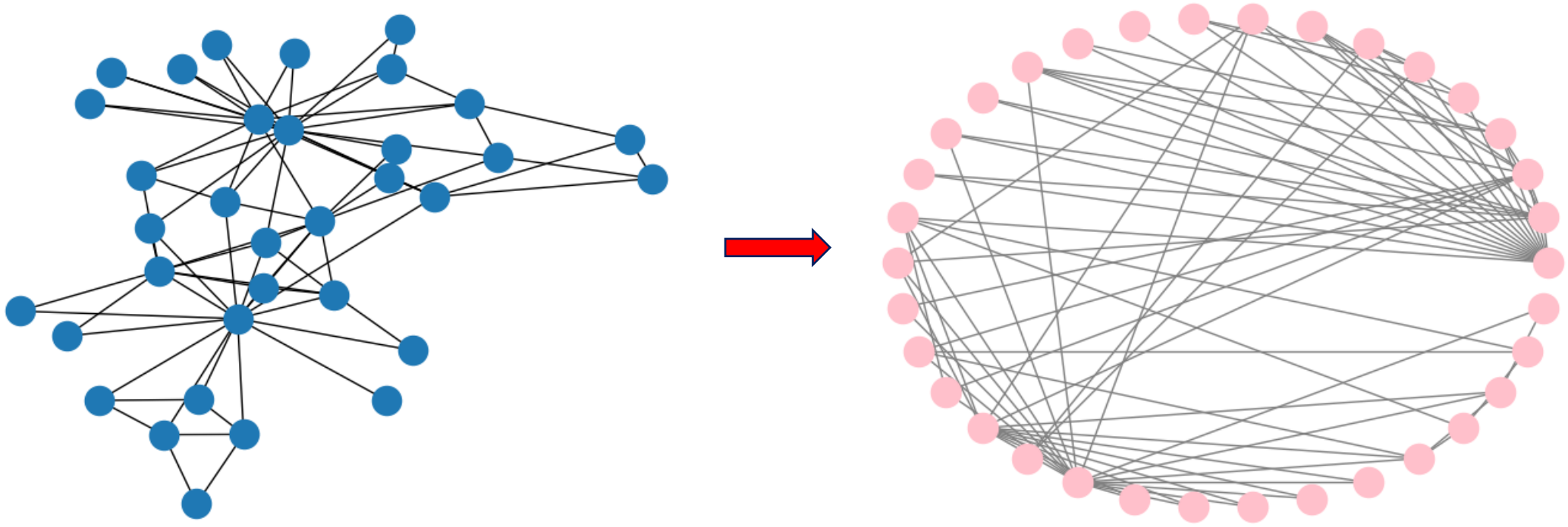
```
# Instantiate the graph
G = nx.Graph()

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



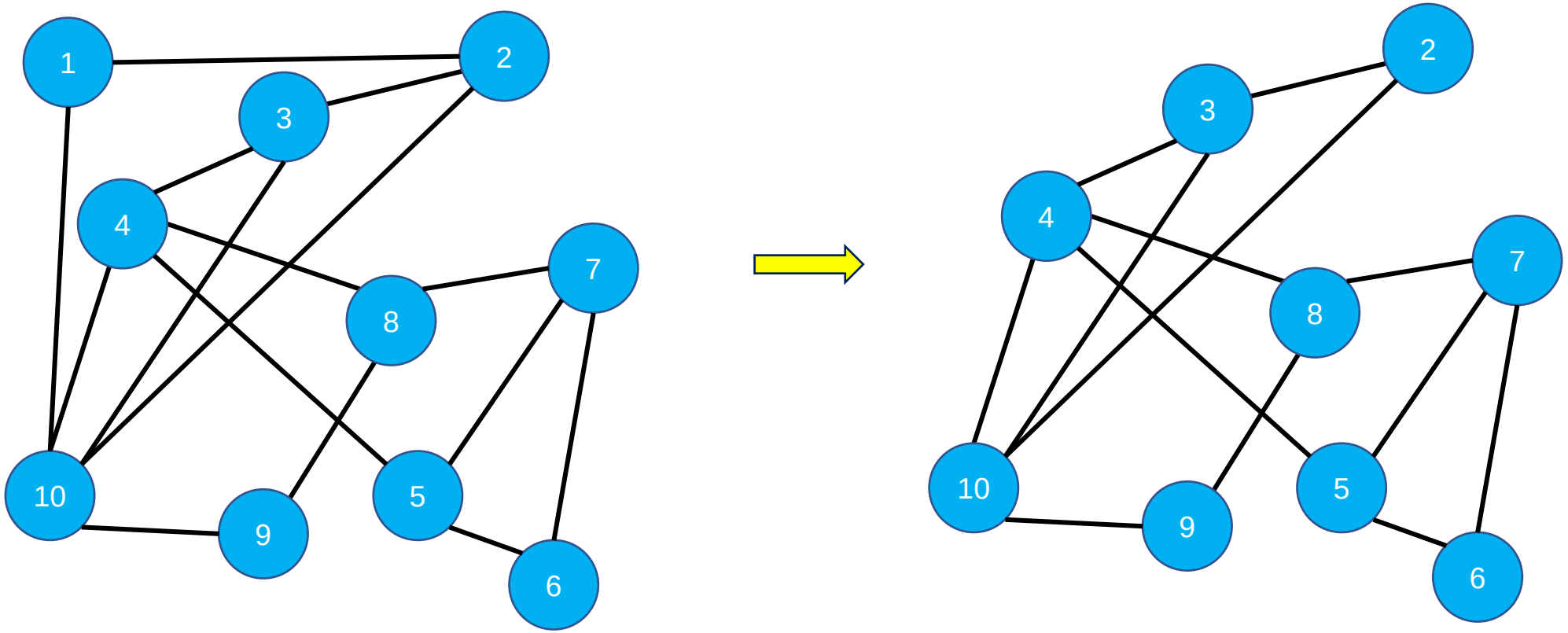
- In circular graphs, close nodes should not be connected:
  - Idea: Finding and store nodes that have two non-connected neighbours by using BFS algorithm.
  - Implement:
    - Starting at a random node, store nodes that do not have non-connected neighbours in a stack.
    - Restore the graph to generate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

https://www.csd.uoc.gr/~hy583/reviewed_notes/circular.pdf

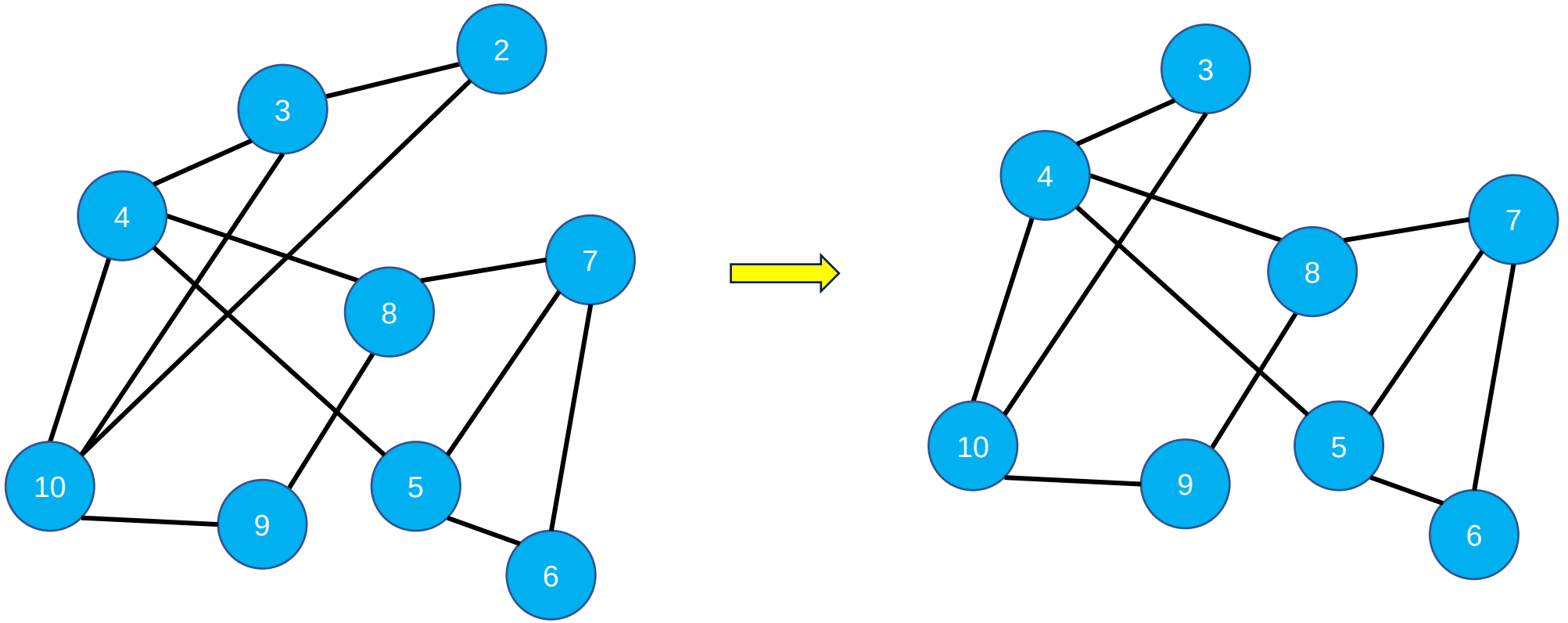
An Example of the Execution of the Algorithm.

- 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.



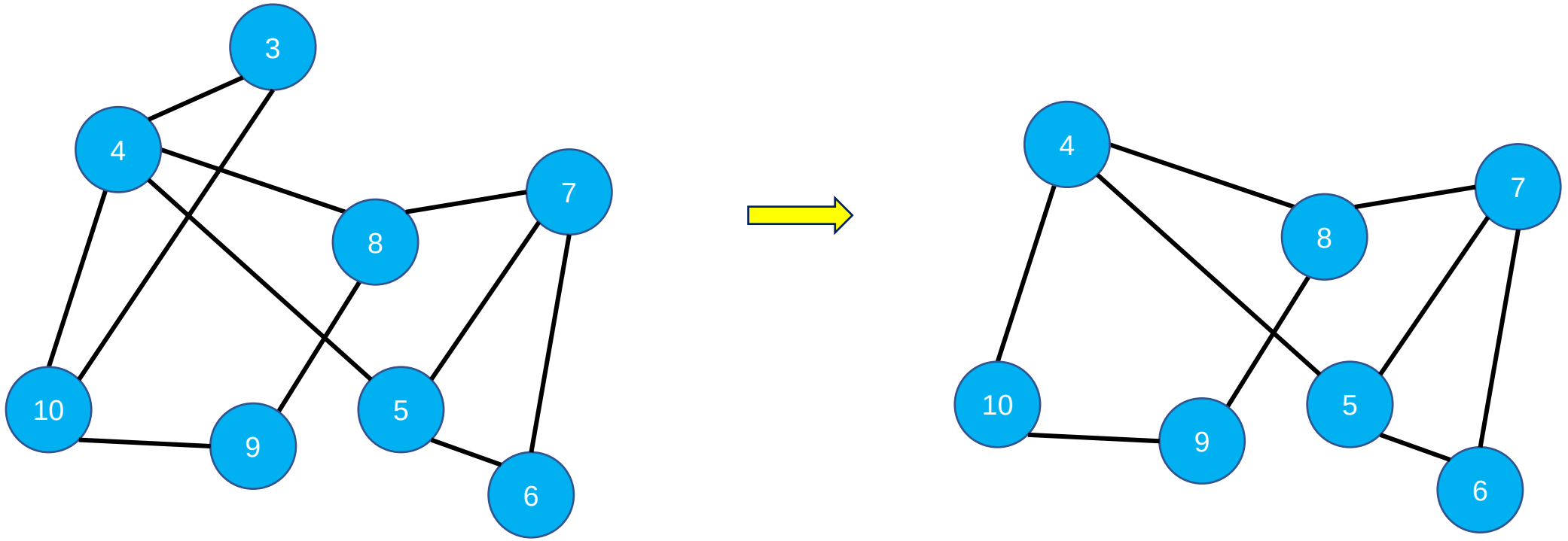
An Example of the Execution of the Algorithm.

- 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.



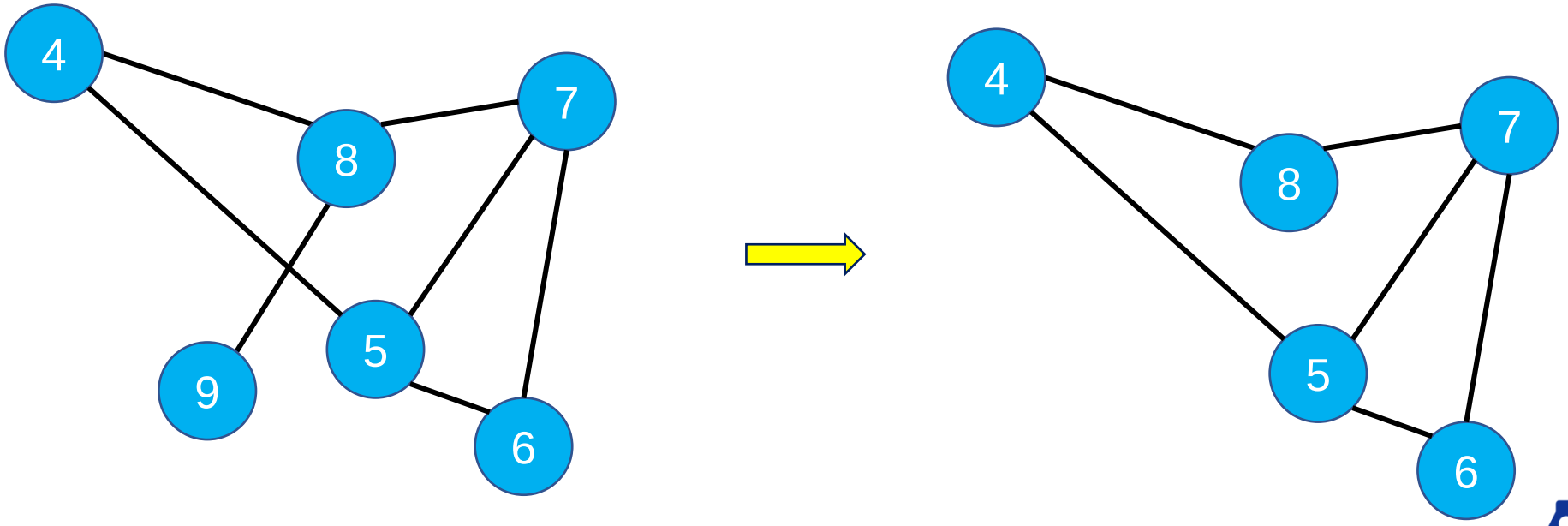
An Example of the Execution of the Algorithm.

- 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.



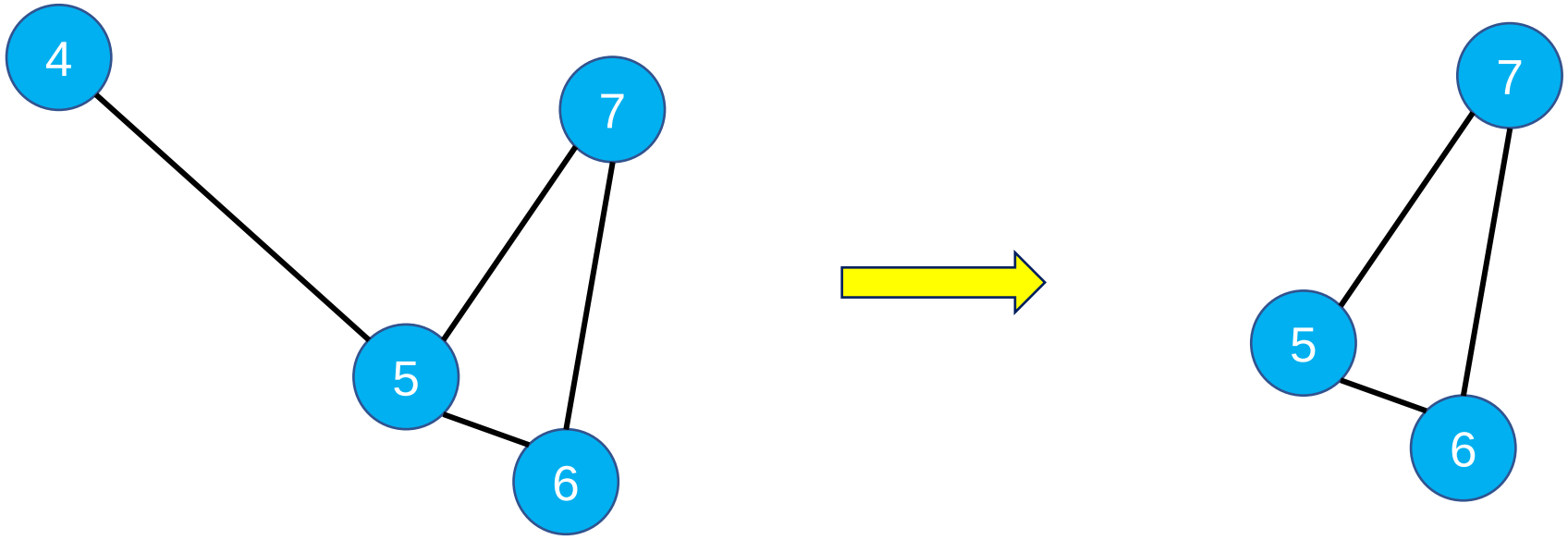
An Example of the Execution of the Algorithm.

- 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.



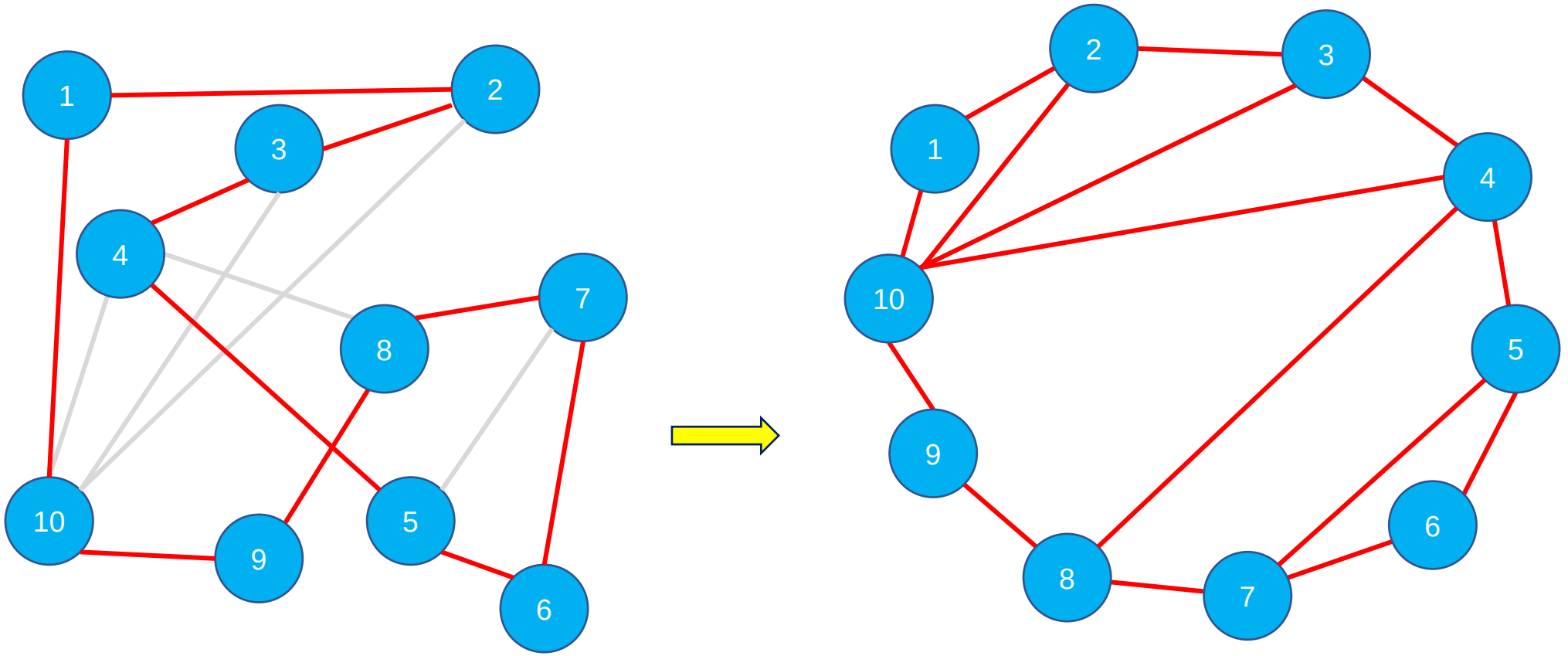
An Example of the Execution of the Algorithm.

- 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.



An Example of the Execution of the Algorithm.

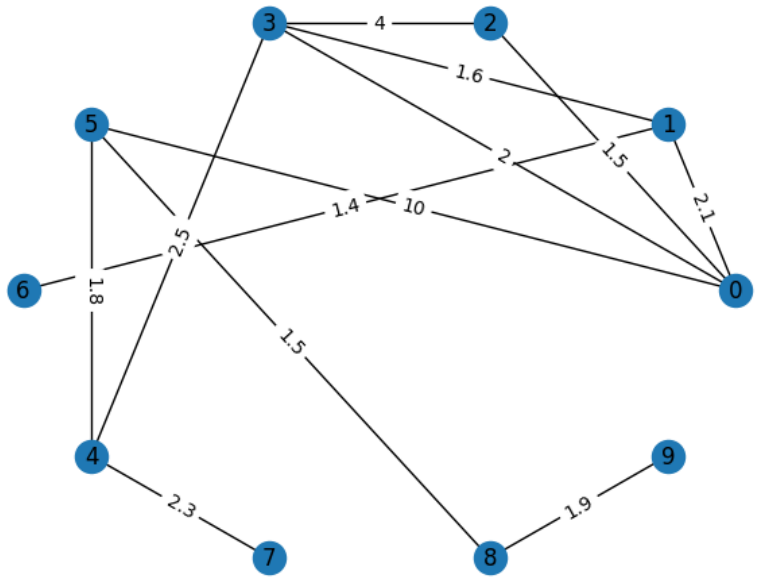
- Now we restore the graph and remove all stored edges.
- Since the graph is outerplanar, we have the Hamilton circle left.



Sample Code: Visualizing a graph using NetworkX

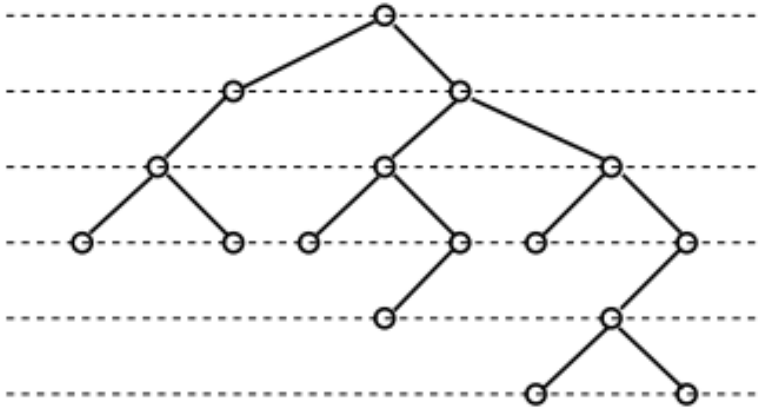
- Circular layout

```
# 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)
```

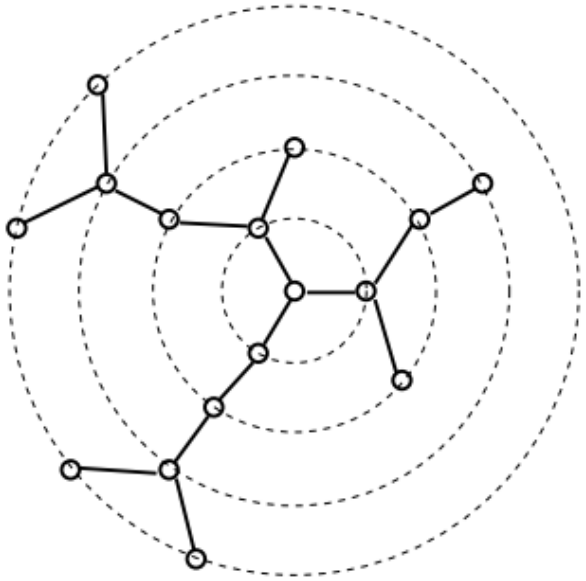


Another Graph Visualizations

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



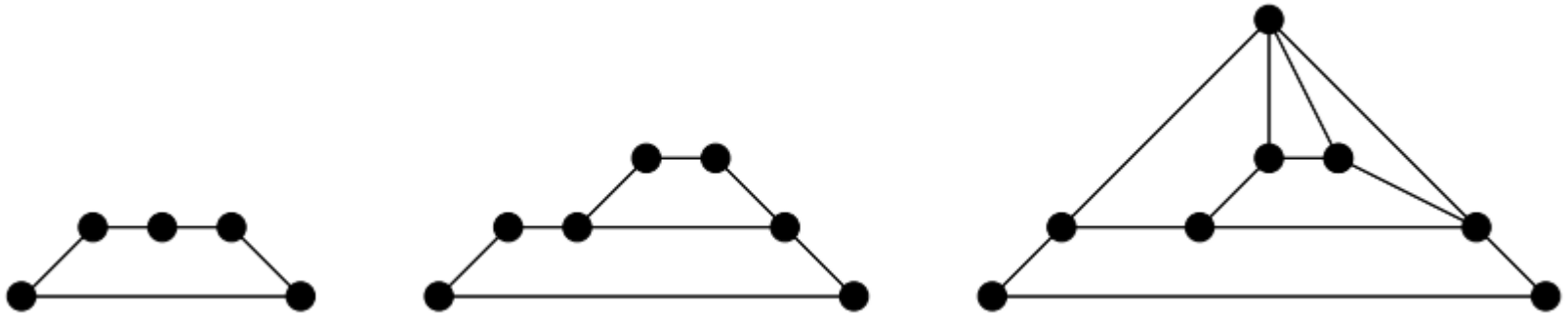
A layered tree drawing



A radial tree drawing

Another Graph Visualizations

- 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

Chrobak, M., and Kant, G. (1997). Convex grid drawings of 3-connected planar graphs. International Journal of Computational Geometry and Applications, 7(3):211–224

Tools For Graph Visualization

- 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…

Tools For Graph Visualization: Gephi

- 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.

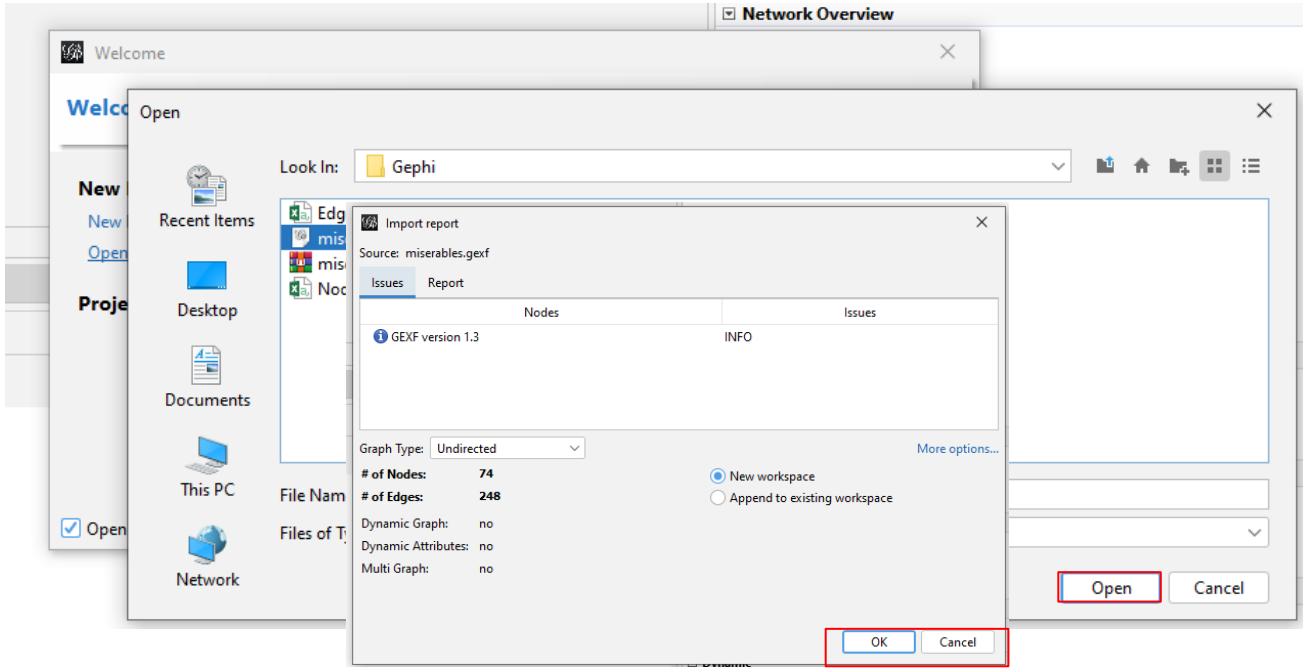
https://github.com/gephi/gephi

Tools For Graph Visualisation: Gephi

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

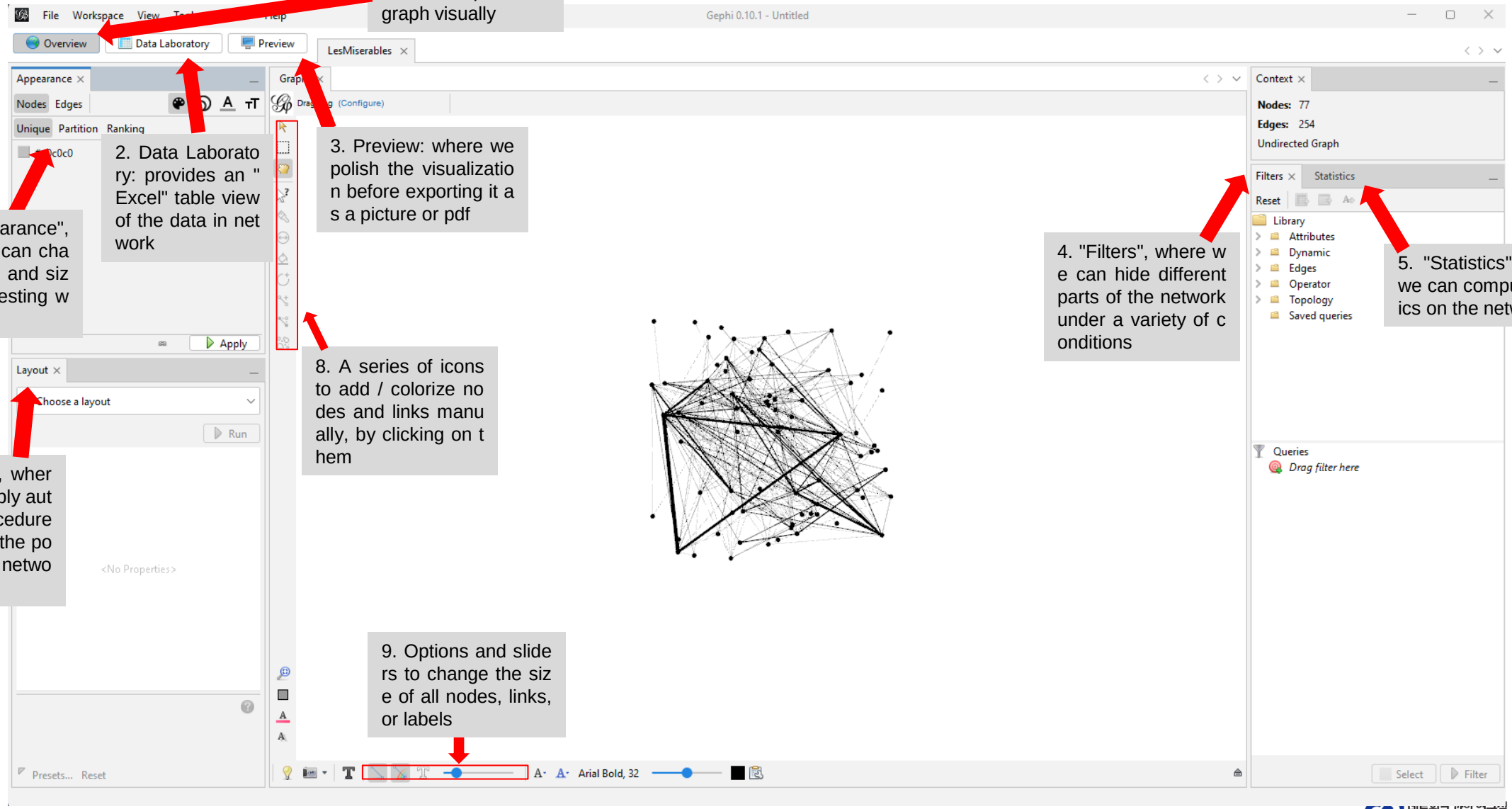
Visualizing a graph using Gephi

- 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



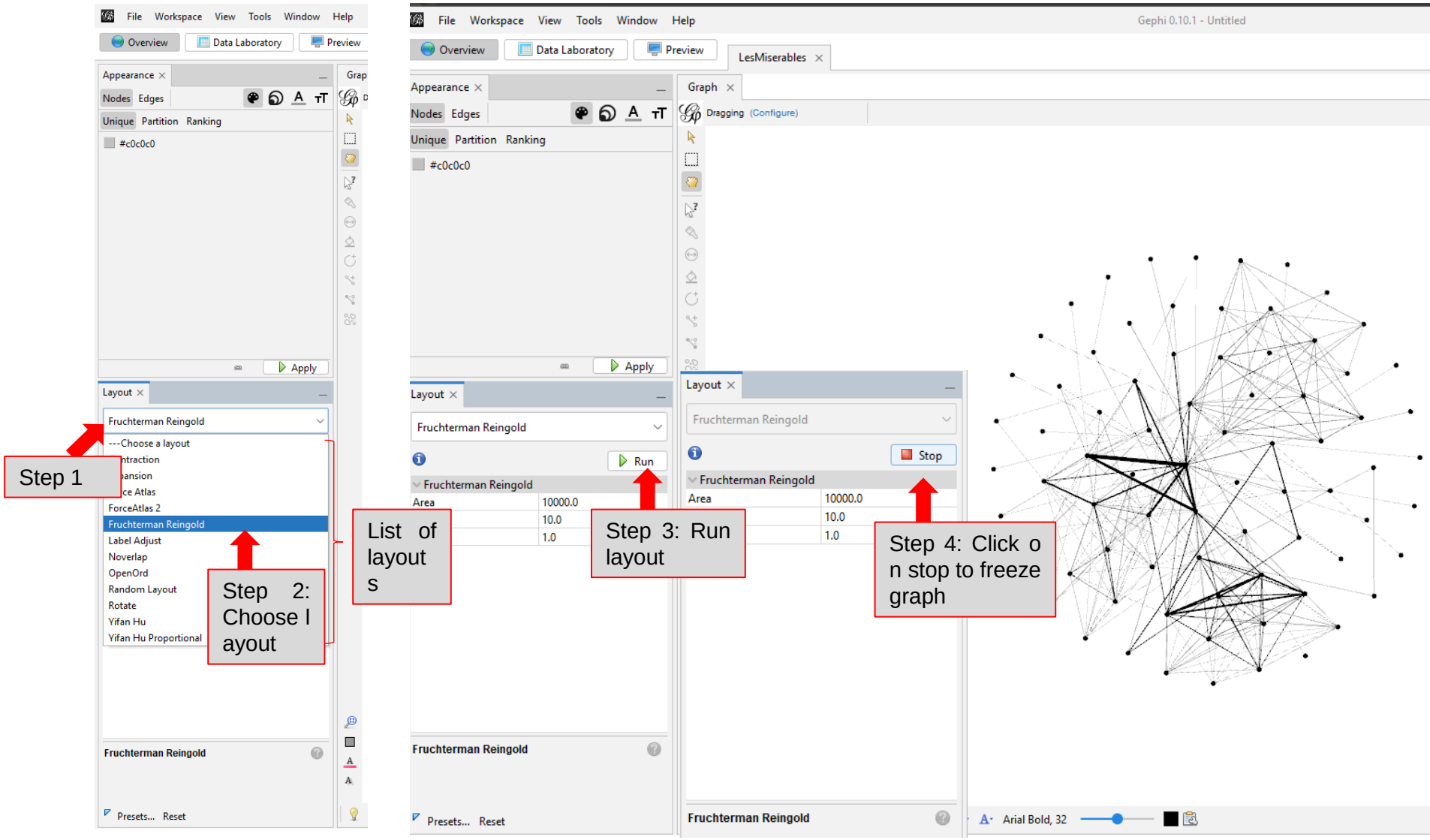
Visualizing a Graph using Gephi

- Gephi's interface:



Visualizing a Graph using Gephi

- Gephi's layout:



Visualizing a graph using Gephi

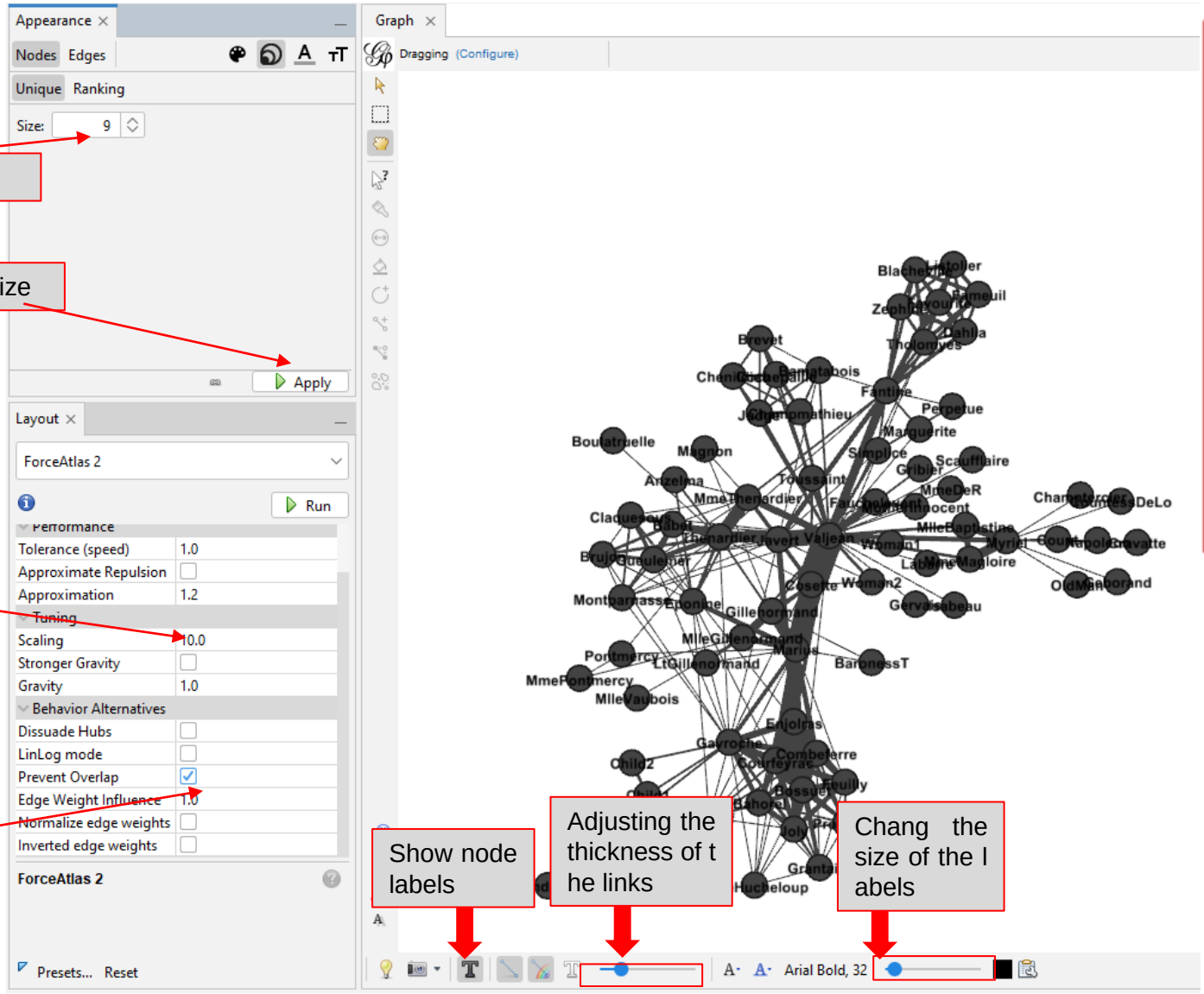
- Force atlas 2 layout:

1. Size: change node size.

2. Click Apply for changing the node size

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

3. Prevent overlap: a parameter to avoi d 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:

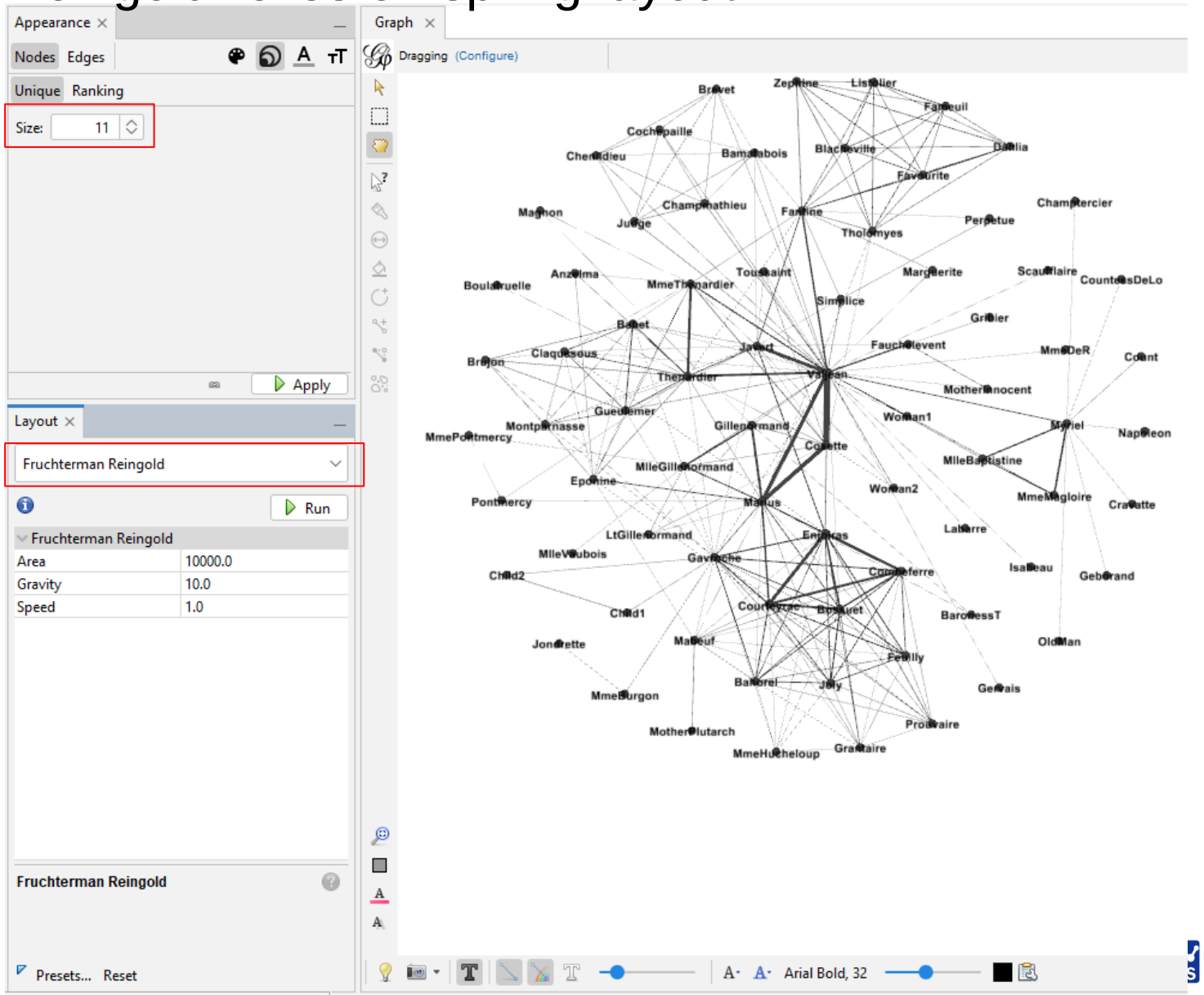
1. brings together nodes which are connected

2. spreads apart unconnected nodes

As an effect, we can easily detect communities of nodes

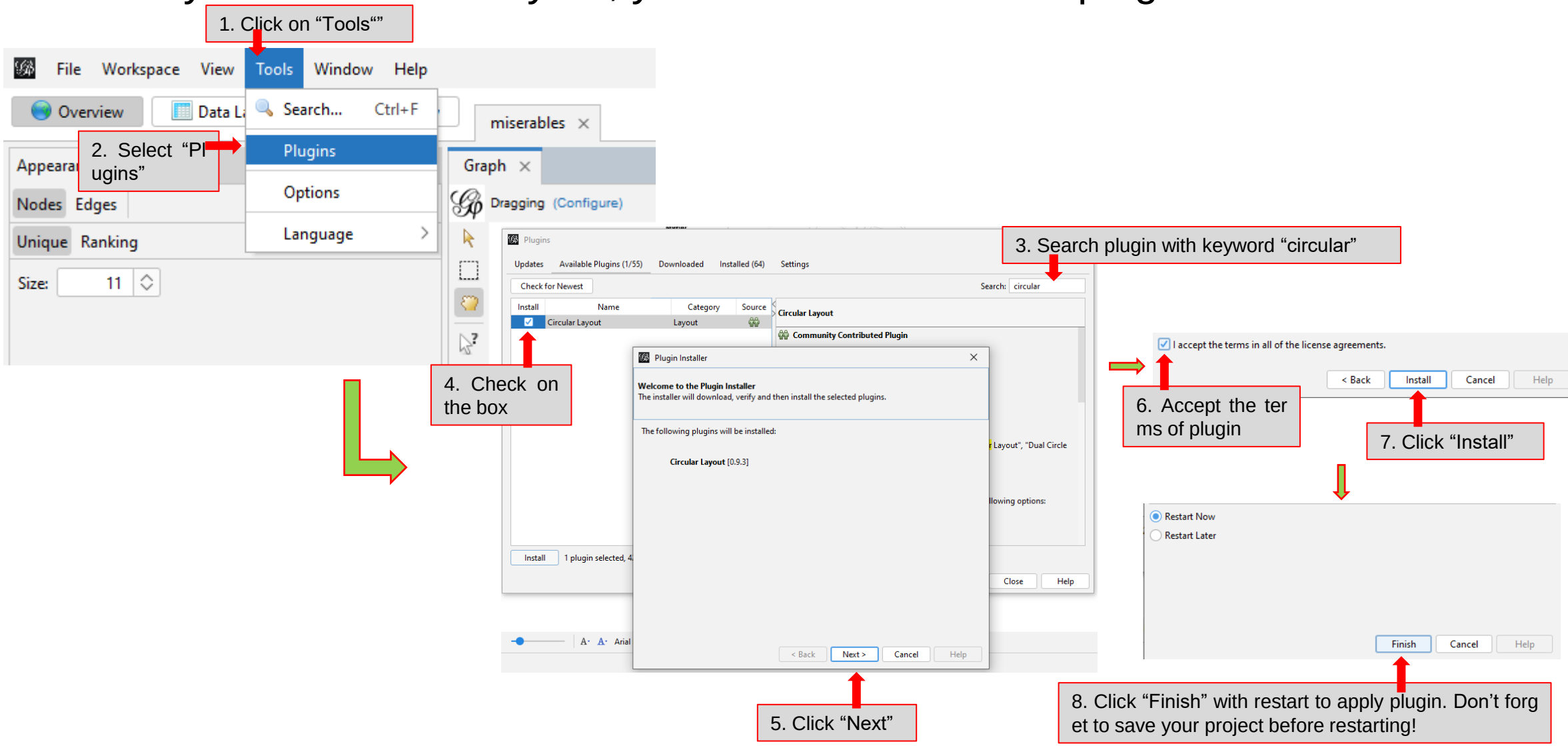
Visualizing a graph using Gephi

- Fruchterman and Reingold relies on spring layout



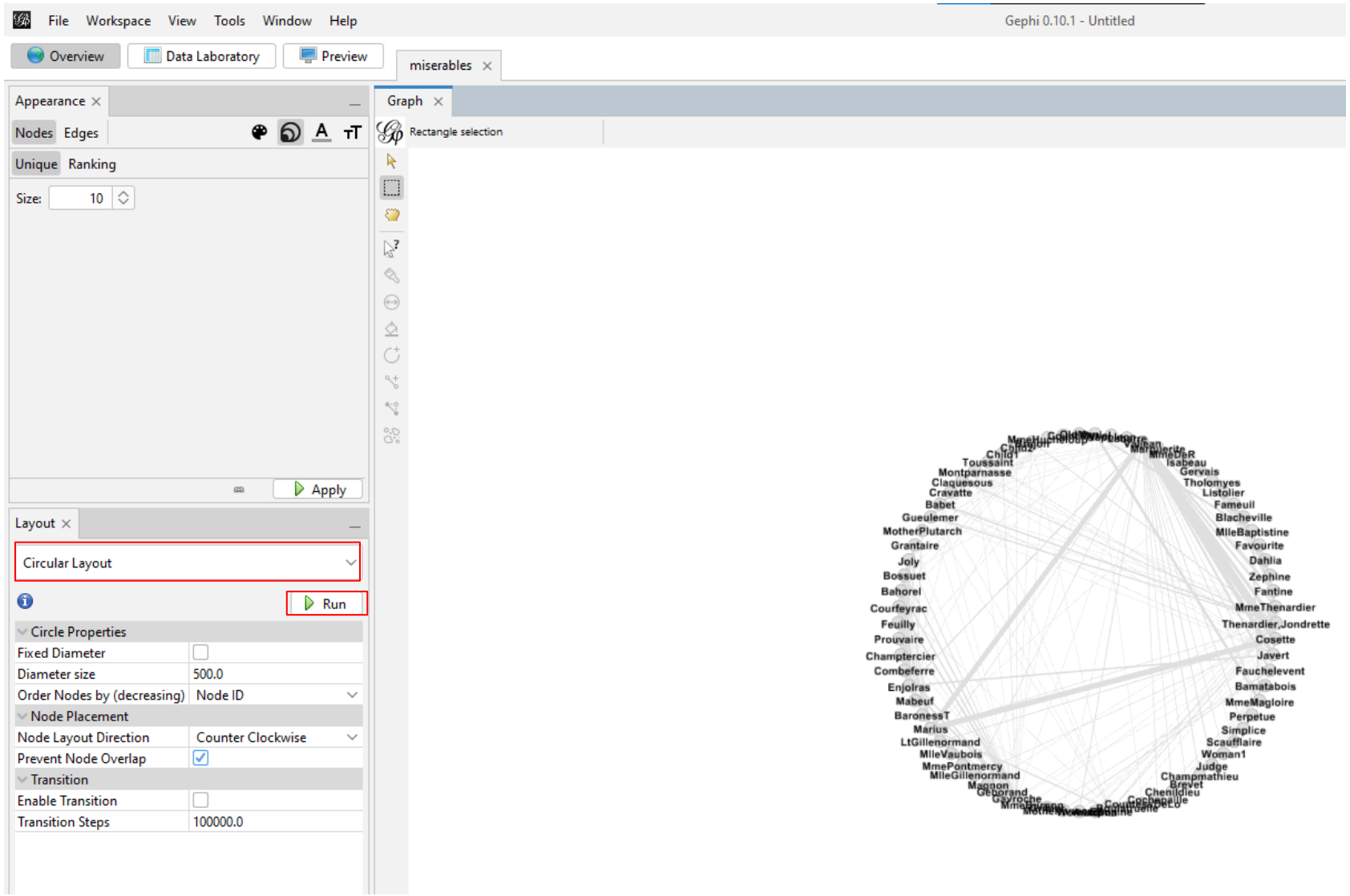
Visualizing a graph using Gephi

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



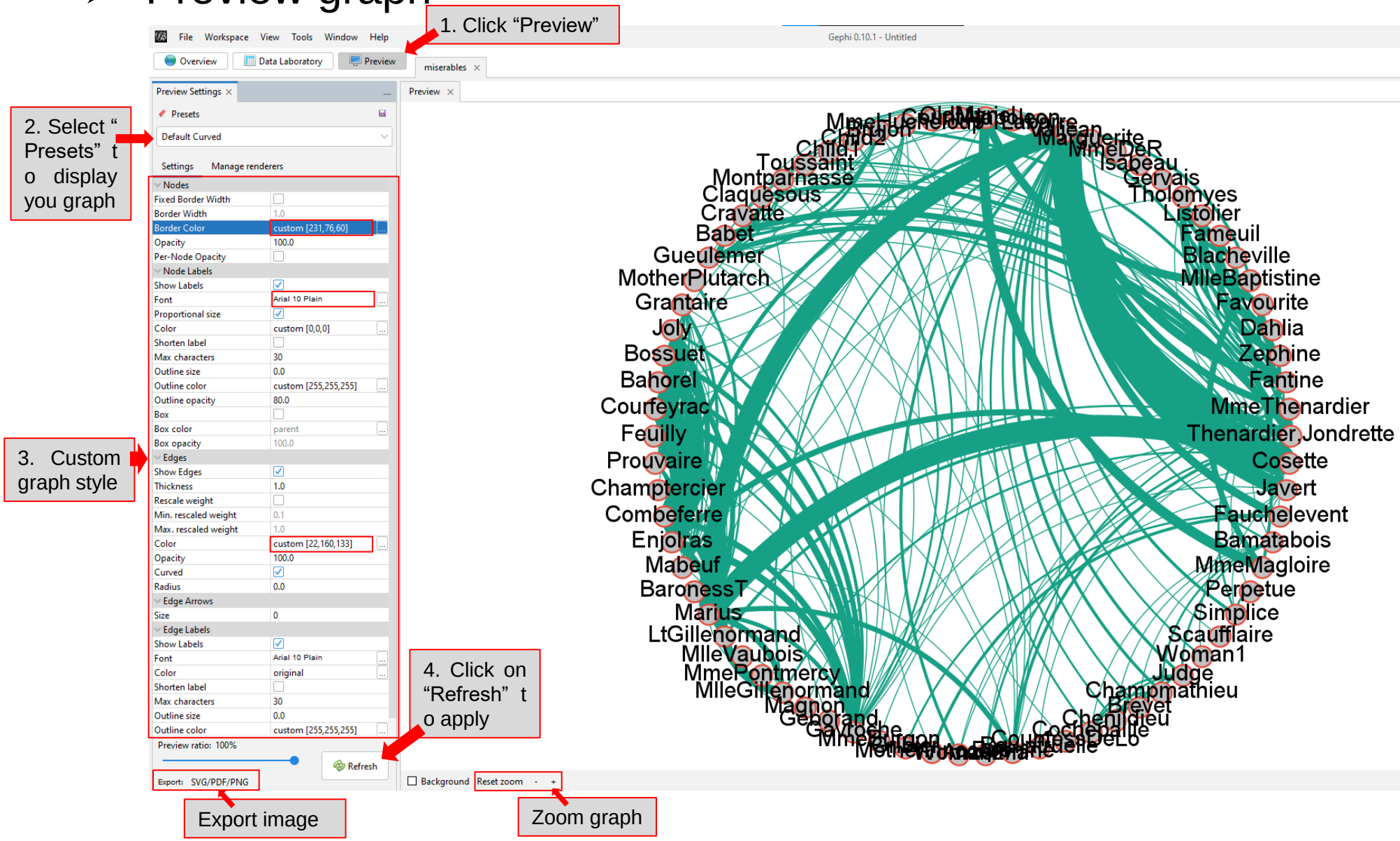
Visualizing a graph using Gephi

- Circular layout:



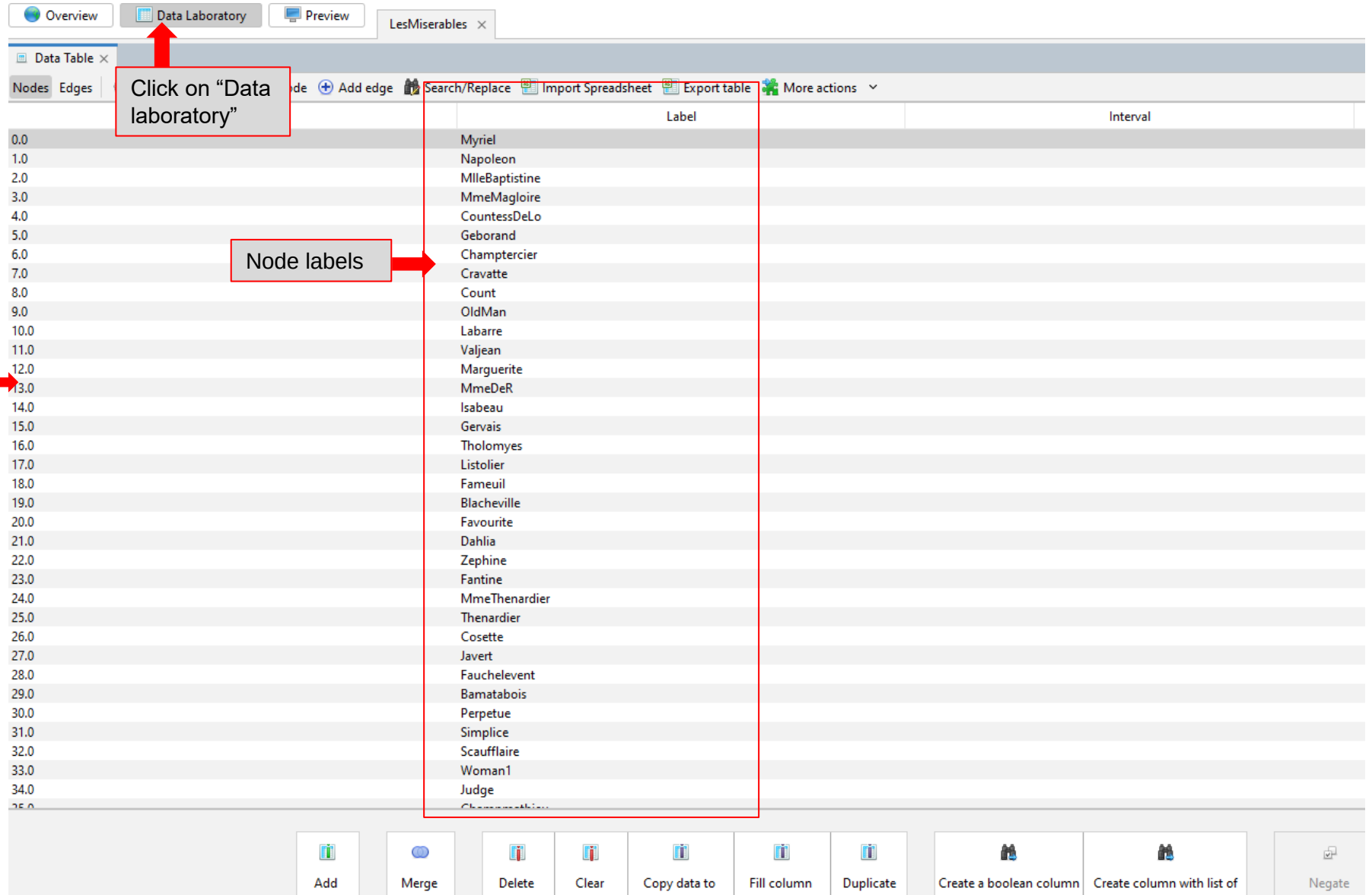
Visualizing a graph using Gephi

➢ Preview graph



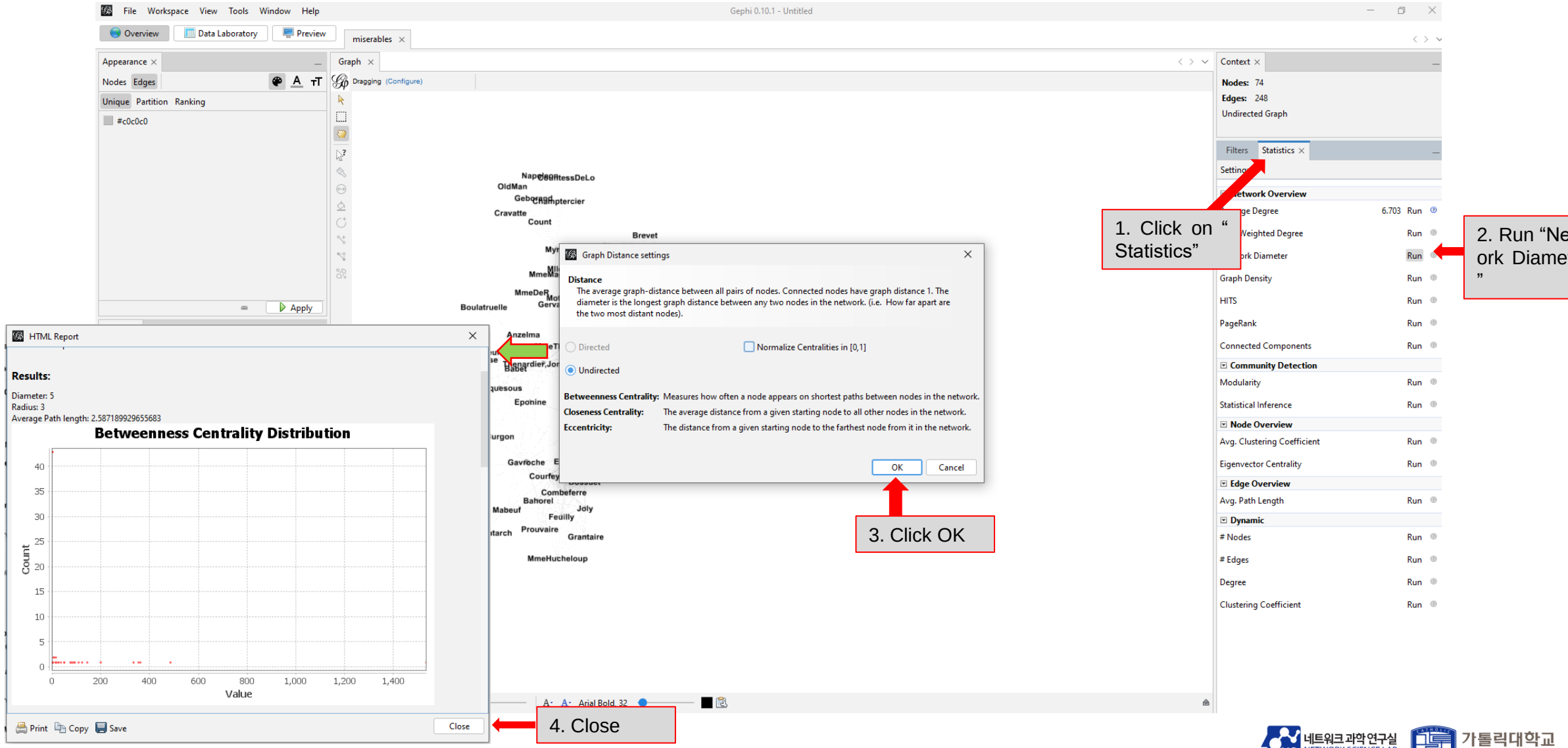
Visualizing a graph using Gephi

- Switching the view to the data laboratory:



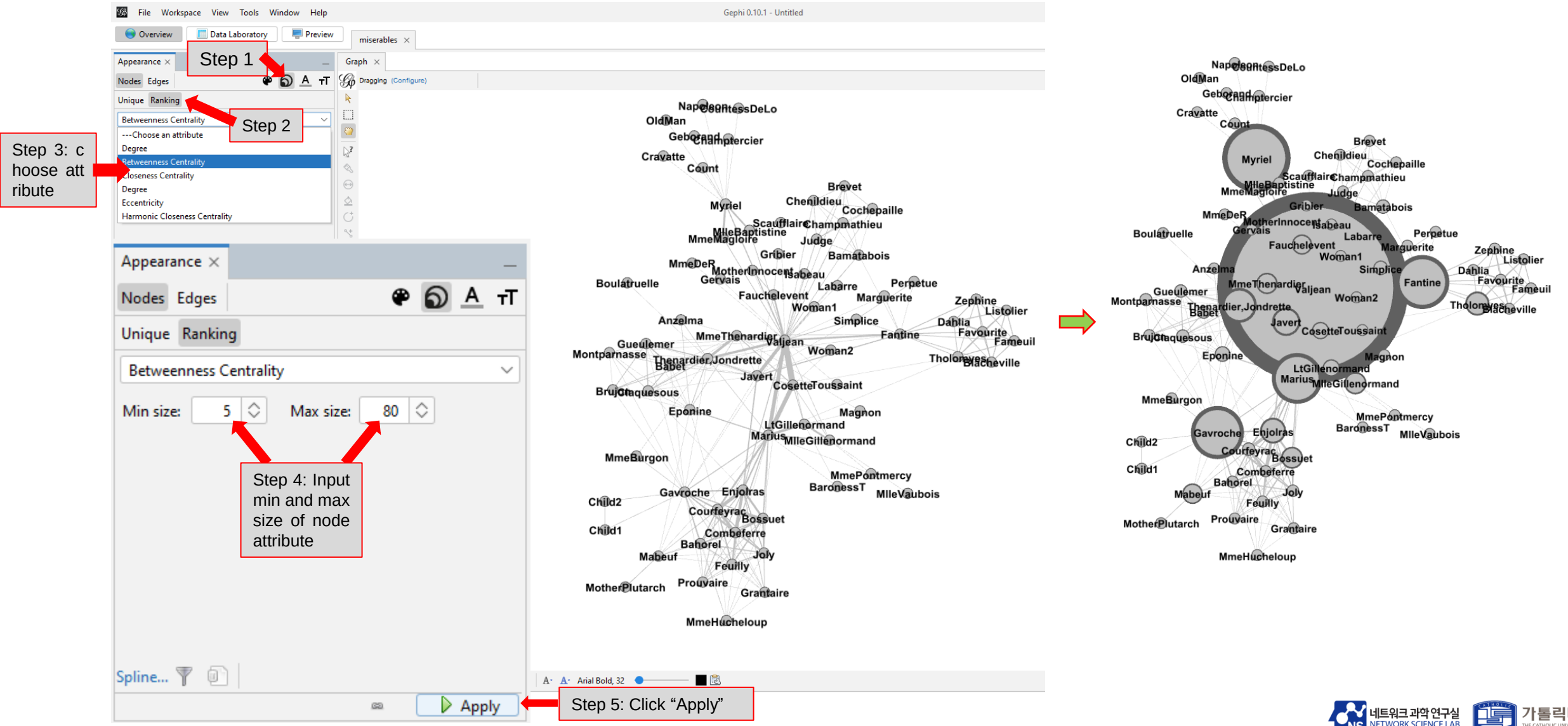
Visualizing a graph using Gephi

- Computing betweenness centrality with Gephi:



Visualizing a graph using Gephi

- View graph attribute: Betweenness Centrality



네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA