# Link prediction


Prof. O-Joun Lee

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

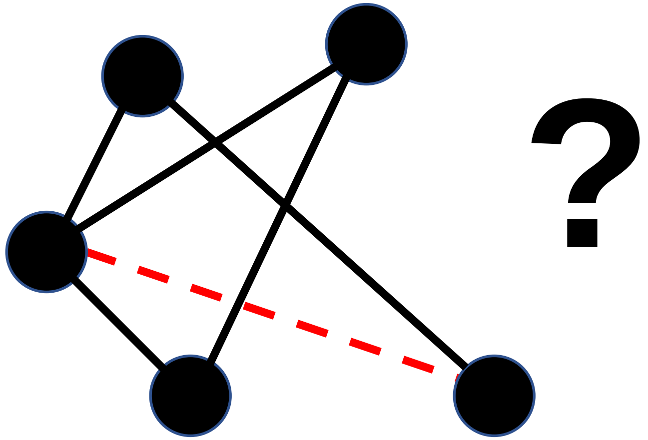

네트워크 과학 연구실 NETWORK SCIENCE LAB
가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA

# Contents

- Link Prediction: problems, intuition
- Commonly methods
  - Local methods: Common neighbors, Jaccard (JC),…
  - Global methods: Katz score, Hitting time, …
- Software Tools
- Evaluation metrics

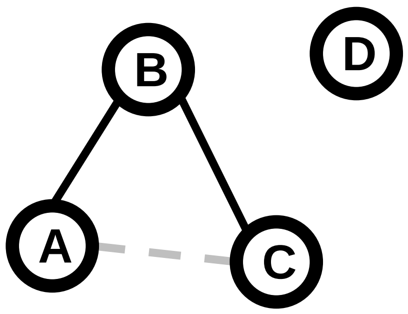
# Motivation

- Understanding how networks evolve
- The link prediction problem
  - Given a snapshot of a social network at time t, we seek to accurately predict the edges that will be added to the network during the interval (t, t')



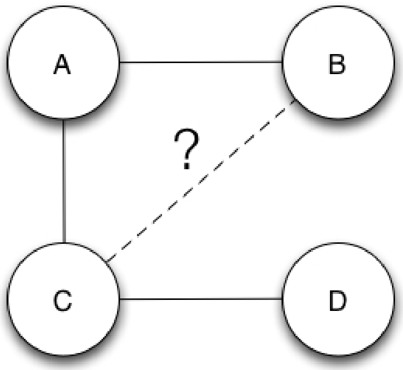
# Motivation

- In case of social network:
  - Model for Network evolution.
  - Predict likely interactions, not explicitly observed, based on observed links – useful for terrorist network monitoring.
  - Link prediction is one instance of Social Network Analysis.
  - Application for “Friend” suggestion in online SN. Notice, this takes on a flavor of link recommendation.



# Link Prediction

- Estimate the likelihood of the existence of a link between two nodes, based on observed links and the attributes of nodes
- Application
  - Biological networks: costly to identify links between nodes through field/laboratorial experiments
  - Online social networks: predicting friendship and recommending new friends (predicting future links in evolving networks)



# Goals



- Present measures of proximity
- Understand relative effectiveness of network proximity measures (adapted from graph theory, CS, social sciences)
- Prove that subtle measures outperform more direct measures

# Why?

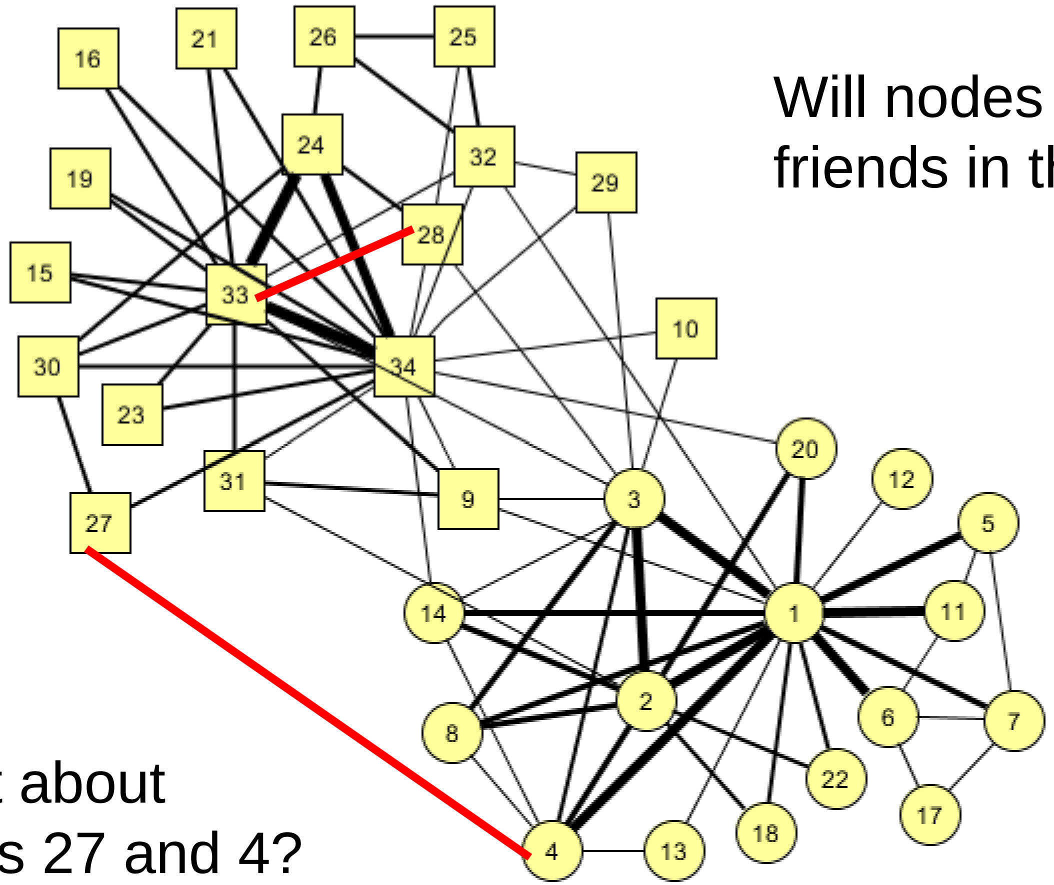
- To suggest interactions or collaborations that haven’t yet been utilized within an organization

- To monitor terrorist networks - to deduce possible interaction between terrorists (without direct evidence)

- Used in Facebook and Linked In to suggest friends

- Open Question: How does Facebook do it?

  (friends of friends, same school, manually…)



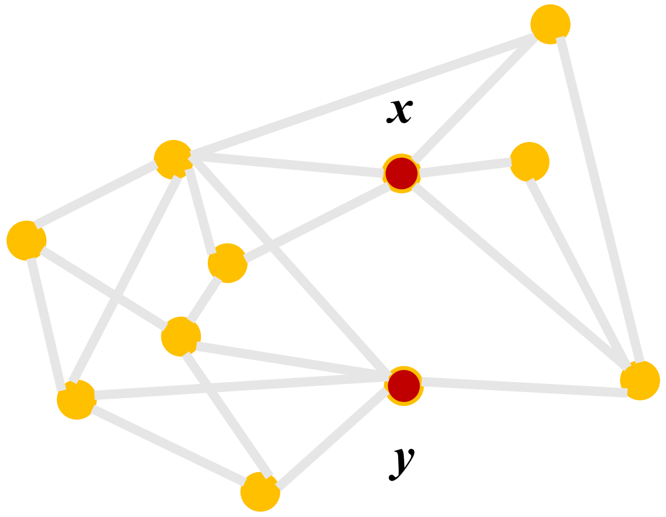

Will nodes 33 and 28 become friends in the future?

Does network structure contain enough information to predict what new links will form in the future?

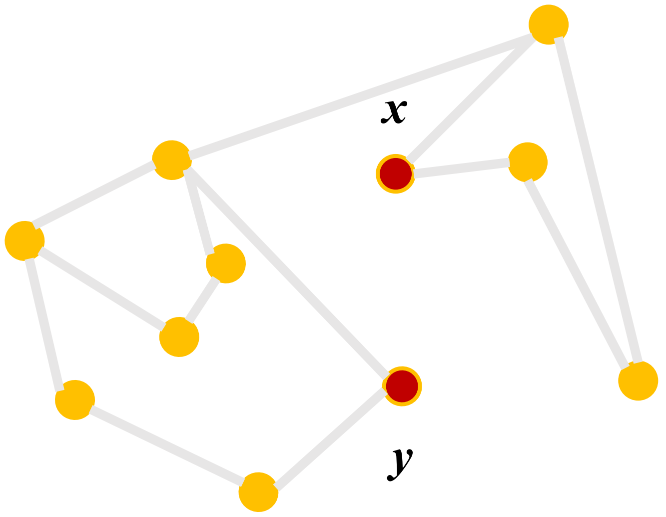
What about nodes 27 and 4?

# The intuition

- In many networks, people who are “close” belong to the same social circles and will inevitably encounter one another and become linked themselves.
- Link prediction heuristics measure how “close” people are

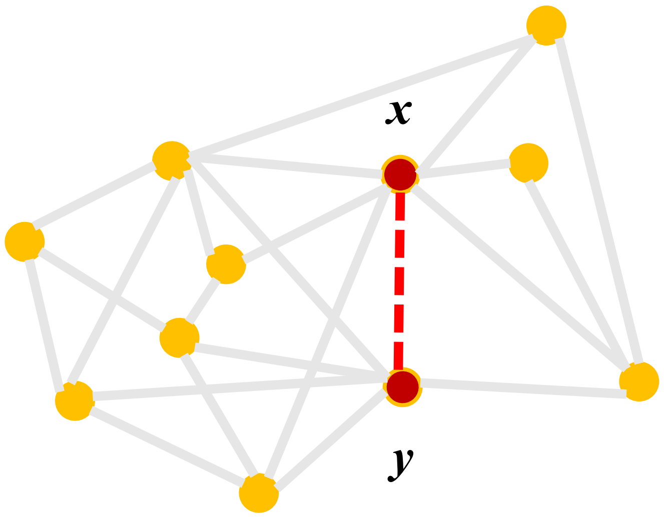



Red nodes are close to each other



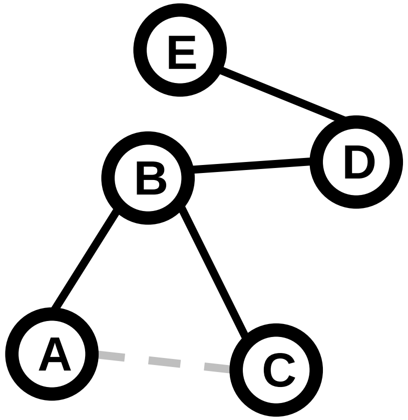

Red nodes are more distant

- Local
  - Common neighbors (CN)
  - Jaccard (JC)
  - Adamic-Adar (AA)
  - Preferential attachment (PA) …
- Global
  - Katz score
  - Hitting time…



# Graph distance

- Graph distance: (Negated) length of shortest path between two nodes in the graphs
- E.g., the distance between nodes in graphs:

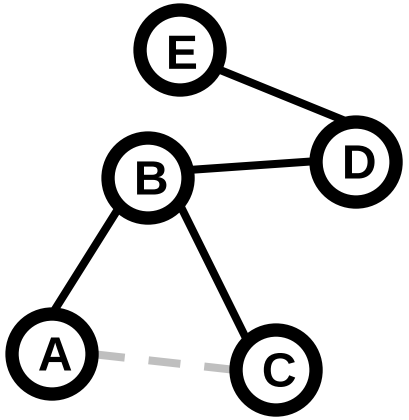



| | |
|---|---|
| (A, C) | -2 |
| (C, D) | -2 |
| (A, E) | -3 |

# Graph distance

```python
# Calculate graph distance
def calculate_graph_distance(G, source, target):
    return -nx.shortest_path_length(G, source=source, target=target)

# Instantiate the graph
G = nx.Graph()

edges = [("A", "B"), ("B", "C"), ("B", "D"), ("D", "E")]
# add node/edge pairs
G.add_edges_from(edges)

print(f"Graph distance:")
print(f"(A, C) | {calculate_graph_distance(G, 'A', 'C')}")
print(f"(C, D) | {calculate_graph_distance(G, 'C', 'D')}")
print(f"(A, E) | {calculate_graph_distance(G, 'A', 'E')}")
```

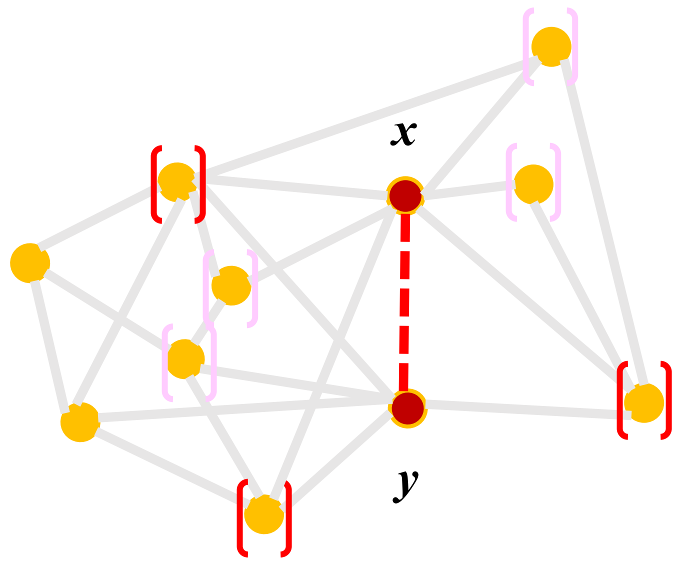
```
Graph distance:
(A, C) | -2
(C, D) | -2
(A, E) | -3
```




| | |
|---|---|
| (A, C) | -2 |
| (C, D) | -2 |
| (A, E) | -3 |

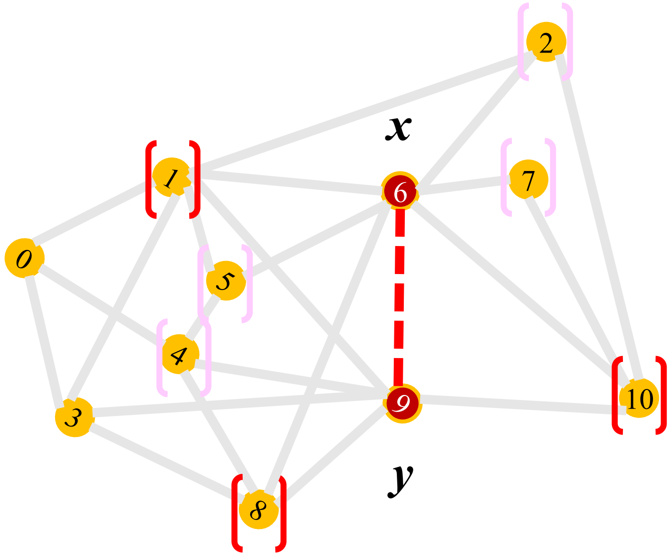
# Common Neighbors

- How many neighbors are in common between x and y
- X and Y have 3 common neighbors, more likely to collaborate
- Let N(x) denote the set of nodes adjancent to x, N(x)= {m| (x,m) ∈ E}




$$CN = |N(x)| \cap |N(y)| = 3$$

# Common Neighbors: Sample code

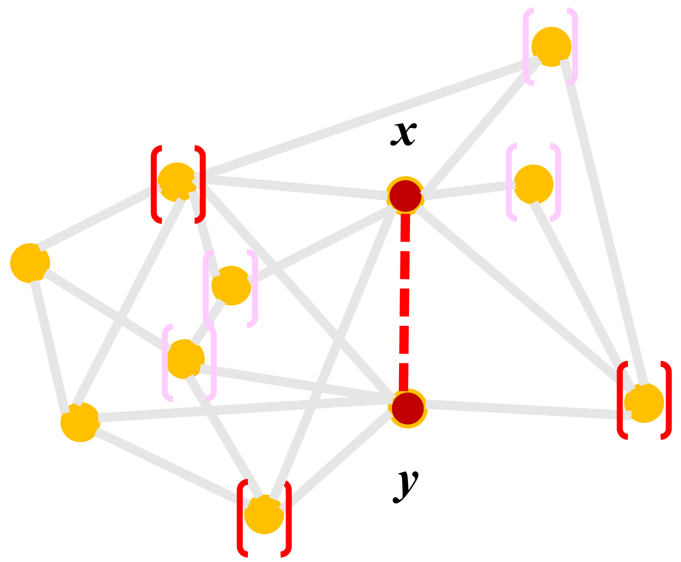



```
G = nx.Graph()
edges = [(0, 1), (1, 2), (0, 3), (1, 3), (0, 4), (1, 5), (4, 5), (1, 6),
         (2, 6), (5, 6), (6, 7), (3, 8), (4, 8), (6, 8), (3, 9), (4, 9),
         (1, 9), (8, 9), (2, 10), (6, 10), (7, 10), (9, 10)]
G.add_edges_from(edges)

cn_list = sorted(nx.common_neighbors(G, 6, 9))

print(cn_list)
```

```
[1, 8, 10]
```

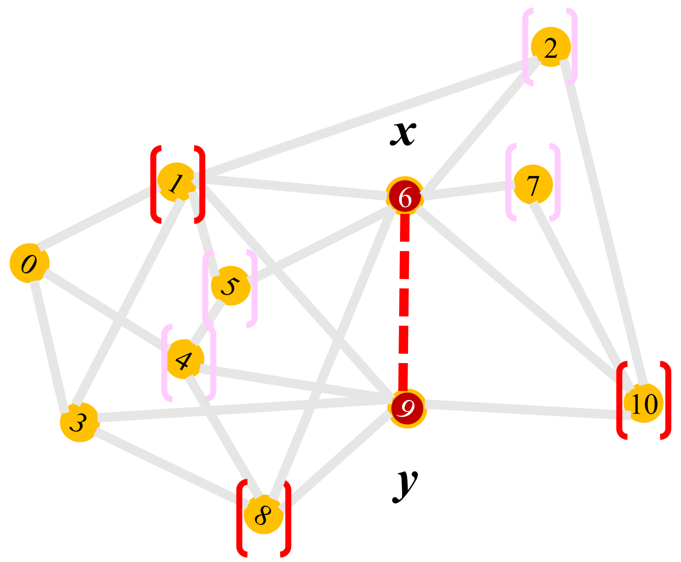
# Jaccard's coefficient

- How likely a neighbor of x is also a neighbor of y
- Same as common neighbors, adjusted for degree




$$JC = \frac{|N(x) \cap N(y)|}{|N(x) \cup N(y)|} = \frac{CN}{d_x + d_y - CN}$$

# Jaccard's coefficient: sample code



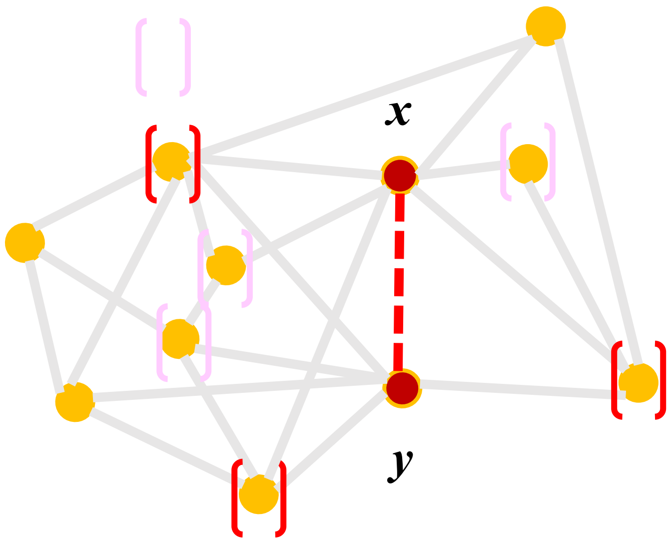

```
# Calculate Jaccard's coefficient of node pairs in a list of nodes
node_list = [(6, 9), (1, 3), (3, 9)]
preds = nx.jaccard_coefficient(G, node_list)
for u, v, p in preds:
    print(f"({u}, {v}) -> {p:.8f}")
```

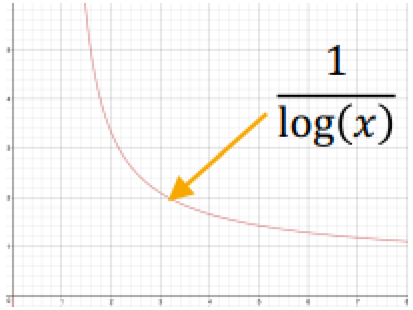
```
(6, 9) -> 0.37500000
(1, 3) -> 0.25000000
(3, 9) -> 0.28571429
```

# Adamic-Adar (AA)

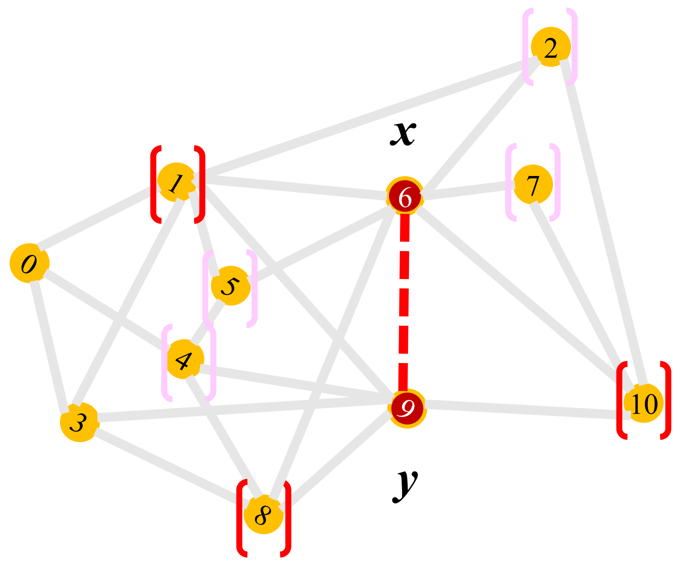
- Large weight to common neighbors with low degree (the lower the degree the higher the relevance)
- E.g., Neighbors who are linked with 2 nodes are assigned weight = 1/log(2) = 1.4
  - Neighbors who are linked with 5 nodes are assigned weight = 1/log(5) = 0.62

$$AA = \sum_{z \in CN} \frac{1}{\log d_z}$$



# Adamic-Adar (AA): sample code

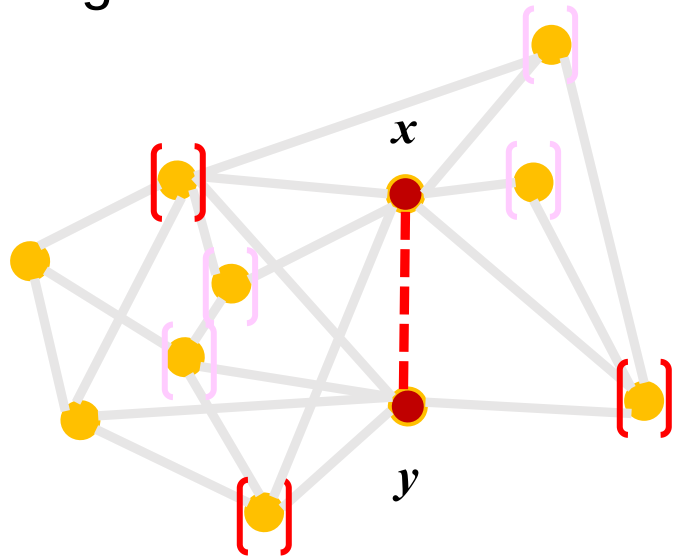



```
# Calculate Adamic-Adar of node pairs in a list of nodes
node_list = [(6, 9), (1, 3), (3, 9)]
preds = nx.adamic_adar_index(G, node_list)
for u, v, p in preds:
    print(f"({u}, {v}) -> {p:.8f}")
```

```
(6, 9) -> 2.00080567
(1, 3) -> 1.53157416
(3, 9) -> 1.27945815
```

# Preferential attachment (PA)

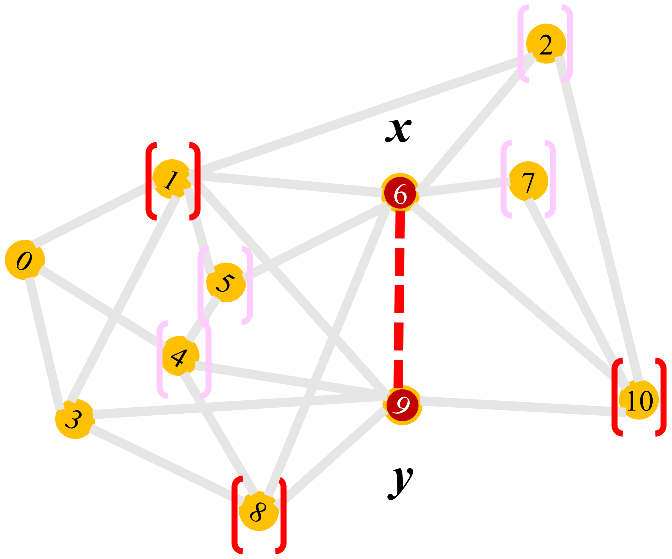
- Better connected nodes are more likely to form more links.
- The more popular a node is the more probable it will form a link with popular nodes.
- This depends on the degrees of the nodes not on their neighborhoods




$$PA = |N(x)|.|N(y)| = d_x.d_y$$

# Preferential attachment (PA): sample code






```
# Calculate Preferential attachment (PA) of node pairs in a list of nodes
node_list = [(6, 9), (1, 3), (3, 9)]
preds = nx.preferential_attachment(G, node_list)
for u, v, p in preds:
    print(f"({u}, {v}) -> {p:.8f}")
```

```
(6, 9) -> 30.00000000
(1, 3) -> 24.00000000
(3, 9) -> 20.00000000
```

# Other neighborhood based methods



- Salton index

$$score(x,y)=\frac{|N(x)\cap N(y)|}{\sqrt{|N(x)||N(y)|}}$$

- Sorensen index

$$score(x,y)=\frac{2|N(x)\cap N(y)|}{|N(x)|+|N(y)|}$$

- Hub Promoted index

$$score(x,y)=\frac{|N(x)\cap N(y)|}{\min\{|N(x)|,|N(y)|\}}$$

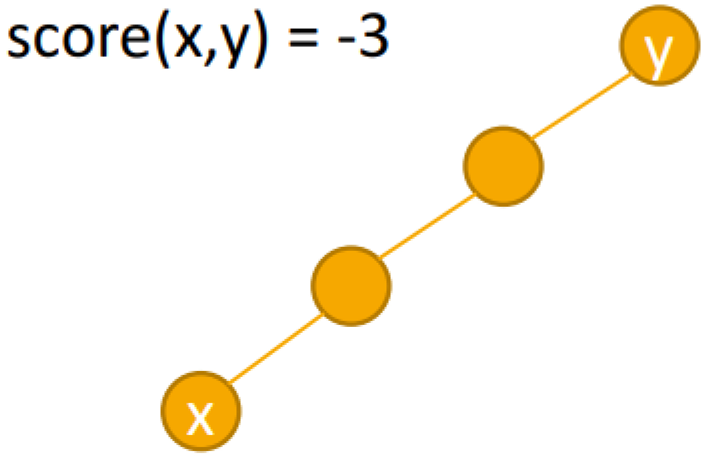
- Hub Depressed Index

$$score(x,y)=\frac{|N(x)\cap N(y)|}{\max\{|N(x)|,|N(y)|\}}$$

https://www.nature.com/articles/s41598-019-57304-y
https://www.nature.com/articles/s41598-020-76799-4

# Methods for Link Prediction: Path based

- Use the (shortest) distance between two nodes as a link prediction measure
- Score(x,y) = (negated) length of shortest path between x and y.
- Very basic approach, it does not consider connections among (x,y) but only the distance



# Methods for Link Prediction: Path based



- Katz index:

Element (x,y) in the adjacency matrix

$$score(x,y) = \sum_{l=1}^{\infty} \beta^l \left| paths_{xy}^{(l)} \right| = \beta A_{xy} + \beta^2 A_{xy}^2 + \ldots$$

- Sum over ALL paths of length $\ell$
- $0 < \beta < 1$ is a parameter of the predictor, exponentially damped to count short paths more heavily
- Small damped parameter = predictions much like common neighbors

# LP Methods: Path based Hitting time

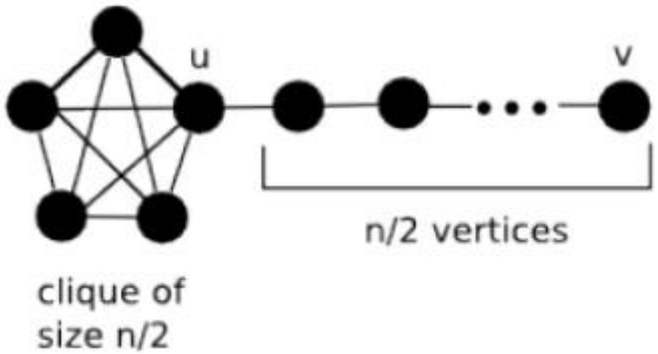
- Consider a random walk on graph G that starts at x and iteratively moves to a neighbor of x chosen uniformly random from N(x)
- **Hitting time (Hxy)** from x to y is the expected number of steps it takes for the random walk starting at x to reach y.

$$score(x,y) = -H_{x,y} = -\frac{1}{|N(\mathrm{x})|}\sum_{k}(1+H_{\mathrm{k},y})$$

$$H(i,j) = 1 + \sum_{k\sim i} p_{ik} H(k,j),\ j \neq i, \quad H(i,i) = 0.$$

- Is Hitting Time Symmetric?
  - NOT symmetric
  - E.g., path from u to v is different

  From v to u



# SimRank

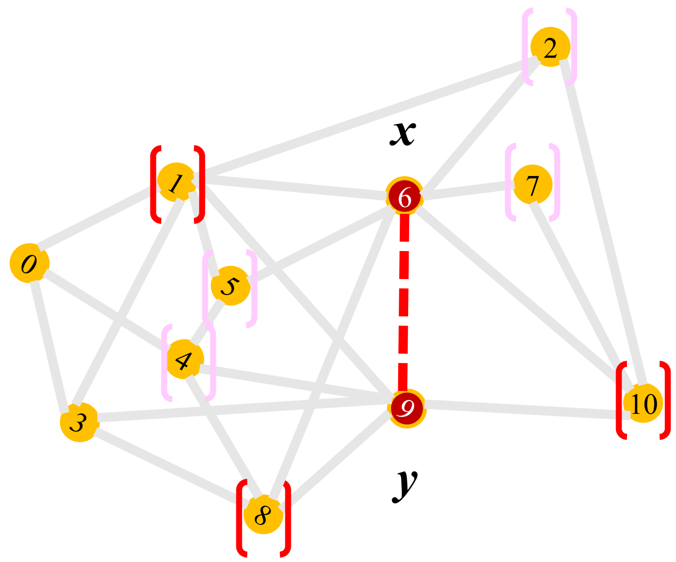
- Intuition: Two objects are similar, if they are related to similar objects
- Two objects x and y are similar, if they are related to objects a and b respectively and a and b are themselves similar

$$\text{similarity}(x, y) = \frac{\sum_{a \in N(\mathrm{x})} \sum_{b \in N(\mathrm{y})} similarity(a,b)}{|N(\mathrm{x})| \bullet |N(\mathrm{y})|}$$

- Expresses the average similarity between neighbors of x and neighbors of y:

score(x, y) = similarity(x, y)

# SimRank: sample code




```python
# Calculate SimRank similarity between 2 nodes
source = 6
target = 9
nx.simrank_similarity(G, source, target)
```

```
0.4283123305678891
```

# Evaluation Metrics

- Precision and Recall, F measure

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$$

**True Positive (TP)**: when both the actual and predicted values are 1.
**True Negative (TN)**: when both the actual and predicted values are 0.
**False Positive (FP):** when the actual value is 0 but the predicted value is 1.
**False Negative (FN)**: when the actual value is 1 but the predicted value is 0.

- True positive rate (TPR), False positive rate (FPR)

$$\text{TPR} = \frac{TP}{TP + FN}$$

$$\text{FPR} = \frac{FP}{FP + TN}$$

- Other metrics:
  - MAP, Precision at K,…

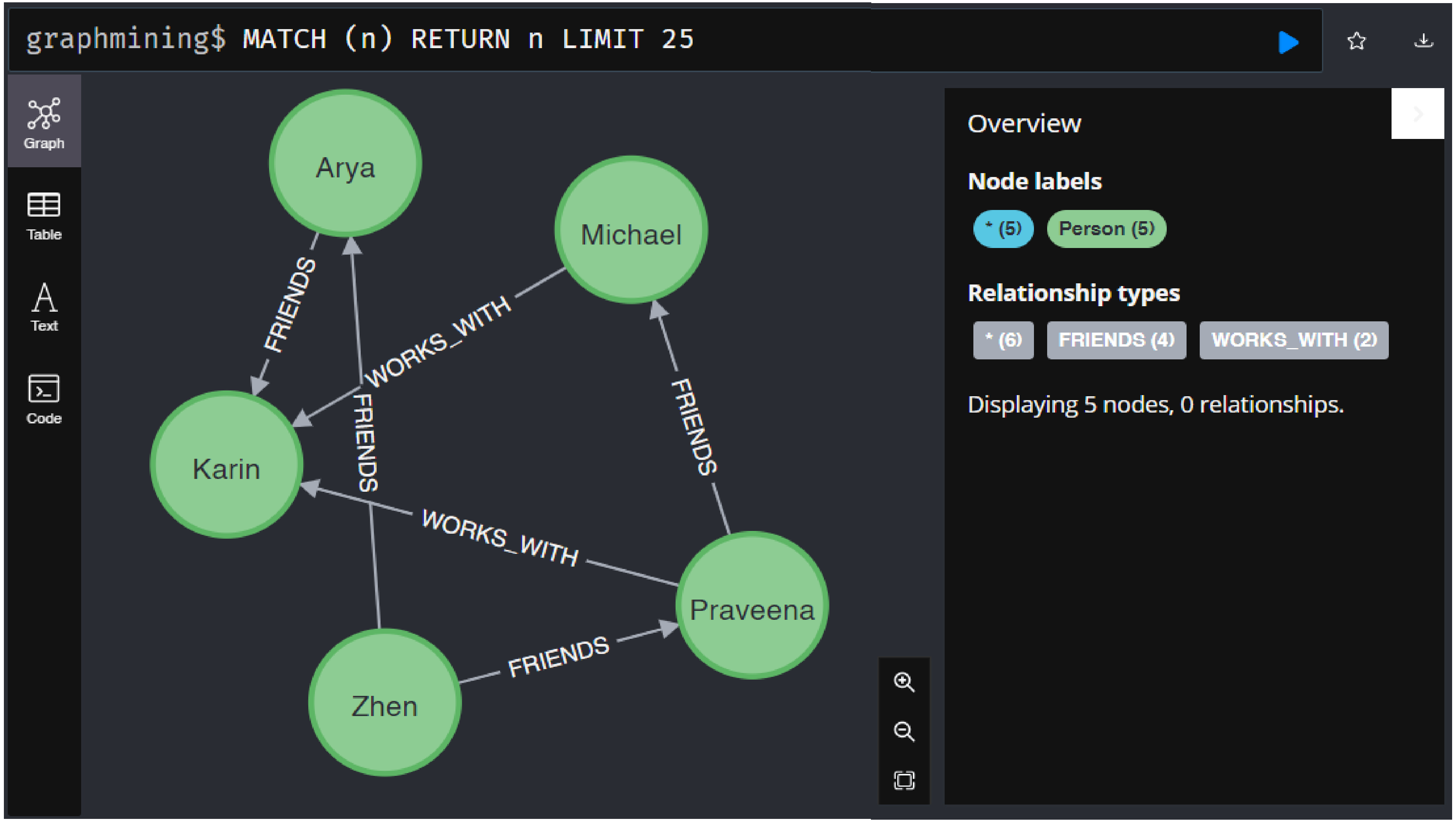
# Software Tools



- NodeXL
  - NodeXL is a graphical front-end that integrates network analysis and SNAP into Microsoft Office and Excel. Using NodeXL, users without programming skills can make use of key elements of the SNAP library.
- InfoVis Cyberinfrastructure
  - Software framework for information visualization (Linux, MacOSX, Windows).
- Analytic Technologies
  - Software for social network analysis (Windows).
- Neo4j
  - Graph visualization software
- NetworkX
  - Python package for the study of the structure of complex networks.

# Software Tools: sample code

- Show database



# Software Tools: sample code

```
CREATE
  (zhen:Person {name: 'Zhen'}),
  (praveena:Person {name: 'Praveena'}),
  (michael:Person {name: 'Michael'}),
  (arya:Person {name: 'Arya'}),
  (karin:Person {name: 'Karin'}),

  (zhen)-[:FRIENDS]->(arya),
  (zhen)-[:FRIENDS]->(praveena),
  (praveena)-[:WORKS_WITH]->(karin),
  (praveena)-[:FRIENDS]->(michael),
  (michael)-[:WORKS_WITH]->(karin),
  (arya)-[:FRIENDS]->(karin)
```

Table

Code

Added 5 labels, created 5 nodes, set 5 properties, created 6 relationships, completed after 21 ms.

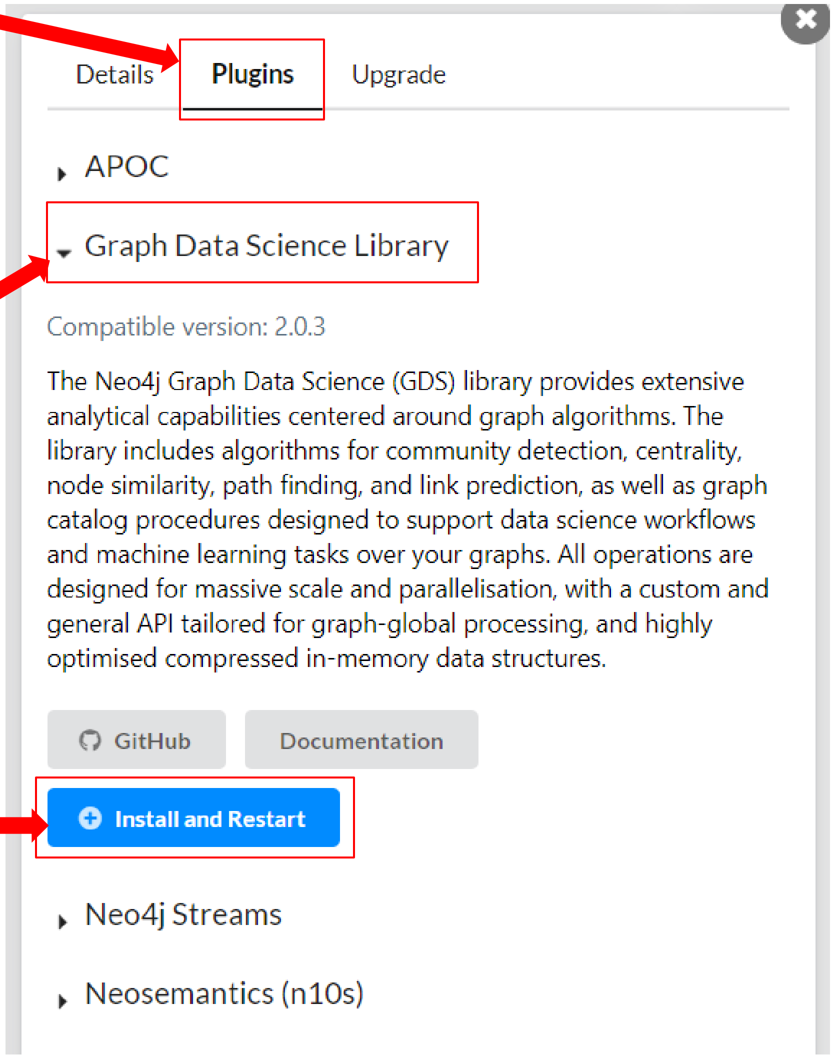
# Software Tools: sample code

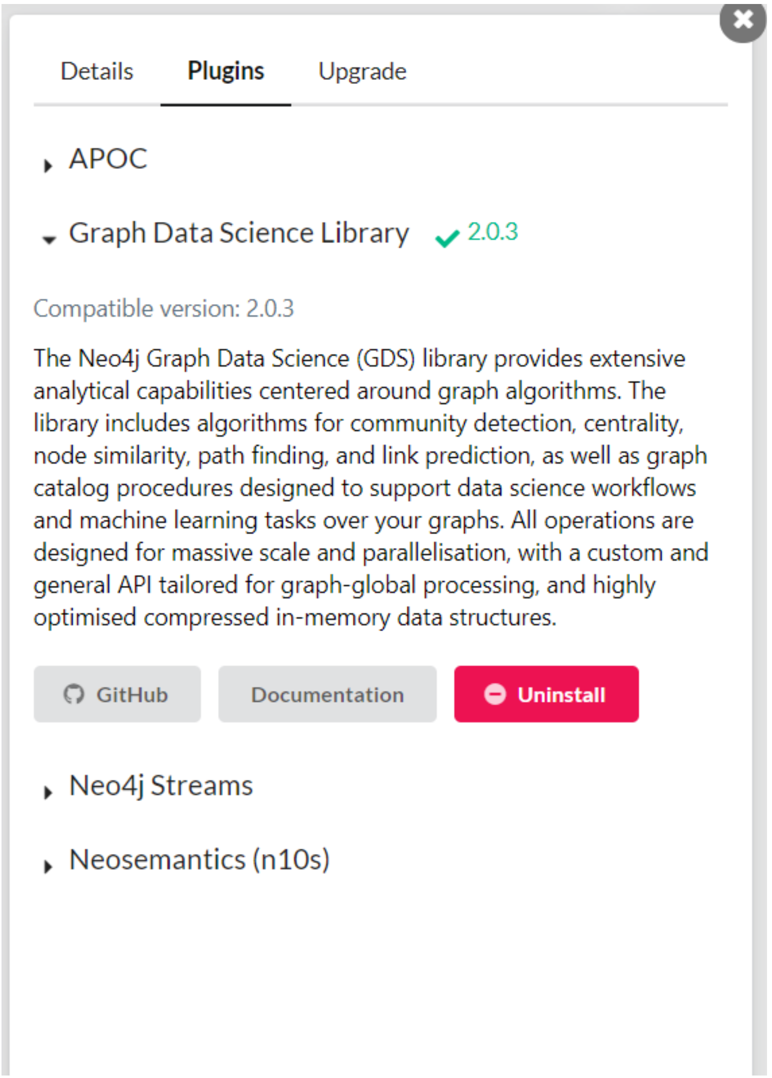
➢ Install the plugin to use graph algorithms

1. Click "Plugins" tab in the pane on the right.

2. Click on "Graph Data Science Library".

3. Install plugin.



- Calculate the number of common neighbors without considering relation type

```
MATCH (p1:Person {name: 'Michael'})
MATCH (p2:Person {name: 'Karin'})
RETURN gds.alpha.linkprediction.commonNeighbors(p1, p2) AS score
```

Table | Text | Code

| | score |
|---|---|
| 1 | 1.0 |

# Software Tools: sample code

- Calculate the number of common neighbors with considering relation type

```
MATCH (p1:Person {name: 'Michael'})
MATCH (p2:Person {name: 'Karin'})
RETURN gds.alpha.linkprediction.commonNeighbors(p1, p2, {relationshipQuery: "FRIENDS"})
AS score
```

| score |
| --- |
| 0.0 |

# Software Tools: sample code

- Calculate the Adamic Adar without considering relation type

```
MATCH (p1:Person {name: 'Michael'})
MATCH (p2:Person {name: 'Karin'})
RETURN gds.alpha.linkprediction.adamicAdar(p1, p2) AS score
```

Table

Text

Code

```
╒══════════════════╕
│"score"           │
╞══════════════════╡
│0.9102392266268373│
└──────────────────┘
```

# Software Tools: sample code



➢ Calculate the Adamic Adar with considering relation type

```
MATCH (p1:Person {name: 'Michael'})
MATCH (p2:Person {name: 'Karin'})
RETURN gds.alpha.linkprediction.adamicAdar(p1, p2, {relationshipQuery: 'FRIENDS'})
AS score
```

Table

Text

Code

| "score" |
|---|
| 0.0 |

# Software Tools: sample code



- Calculate the Preferential Attachment without considering the relation type

```
MATCH (p1:Person {name: 'Michael'})
MATCH (p2:Person {name: 'Karin'})
RETURN gds.alpha.linkprediction.preferentialAttachment(p1, p2) AS score
```

Table

Text

Code

```
╒═══════╕
│"score"│
╞═══════╡
│6.0    │
└───────┘
```

# Software Tools: sample code



- Calculate the Preferential Attachment with considering relation type

```
MATCH (p1:Person {name: 'Michael'})
MATCH (p2:Person {name: 'Karin'})
RETURN gds.alpha.linkprediction.preferentialAttachment(p1, p2,
{relationshipQuery: "FRIENDS"}) AS score
```

Table

Text

Code

| "score" |
| --- |
| 1.0 |


Q
A