Social Network Analysis

Lecture 2: Network Representation & Tools

Dr. Hung-Nghiep Tran
nghiepth@uit.edu.vn

University of Information Technology, VNU-HCM, Vietnam
2025

Mục tiêu buổi học

- Differentiate multiple graph types: bipartite, directed/undirected, weighted/unweighted, single/multigraph, etc.
- Choose between edge list, adjacency list, and adjacency matrix with clear memory/time trade-offs
- Perform basic I/O and analysis with NetworkX and do a quick visualization in Gephi; try another tool such as SNAP.py or Pytorch Geometric
- Produce a tiny, reproducible workflow: load → inspect → compute → export → visualize

Câu hỏi kiểm tra trước

Quiz

1. In a bipartite graph, edges connect:
A. Any nodes in the same set B. Nodes across two disjoint sets C. Only weighted nodes D. Only directed nodes

2. An edge list primarily stores:
A. A dense boolean table of adjacencies
B. Pairs (u, v) (plus optional attributes)
C. For each node, a dictionary of neighbors
D. Node features only

3. For a very sparse network that grows via edge insertions, the default-practical structure is:
A. Adjacency matrix B. Incidence matrix C. Adjacency list D. Edge weight matrix

4. A multigraph allows:
A. Self-loops only B. Parallel edges between the same pair of nodes C. Only weighted edges D. Only directed edges

5. Which tool is primarily for interactive visualization (layouts, styling)?
A. NetworkX B. SNAP.py C. Gephi D. pandas

Quiz

1. In a **bipartite** graph, edges connect:
A. Any nodes in the same set B. **Nodes across two disjoint sets** C. Only weighted nodes D. Only directed nodes

2. An **edge list** primarily stores:
A. A dense boolean table of adjacencies
B. **Pairs (u, v) (plus optional attributes)**
C. For each node, a dictionary of neighbors
D. Node features only

3. For a **very sparse** network that grows via edge insertions, the default-practical structure is:
A. Adjacency matrix B. Incidence matrix C. **Adjacency list** D. Edge weight matrix

4. A **multigraph** allows:
A. Self-loops only B. **Parallel edges between the same pair of nodes** C. Only weighted edges D. Only directed edges

5. Which tool is primarily for **interactive visualization** (layouts, styling)?
A. NetworkX B. SNAP.py C. **Gephi** D. pandas

Đọc và diễn giải

Tài liệu đọc

- Reading:
  - (Re-read) Choice of Network Representations
  - (Stanford Recitation 01) SNAP.PY recitation
  - (Stanford Recitation 02) Proofs recitation

Câu hỏi kiểm tra

Quiz

1. Best representation for fast adjacency checks (“is v a neighbor of u?”) on a sparse graph with low memory?
A. Adjacency list (hash-based neighbor sets) B. Edge list C. Dense adjacency matrix D. CSV table

2. You need repeated matrix-vector multiplications Ax for algorithms like PageRank on a large graph. Which fits best?
A. Edge list only B. Sparse adjacency matrix (CSR/CSC) C. Dense adjacency matrix D. Adjacency list

3. You must count triangles quickly. Which combo is generally most efficient?
A. Edge list + nested scans
B. Sorted adjacency lists (set intersections) or sparse matrix ops
C. Dense matrix + triple loops in Python
D. CSV join in pandas

4. You have a tiny, dense graph ($n \leq 2k$) and need constant-time adjacency checks. Best structure?
A. Edge list B. Dense adjacency matrix C. Adjacency list D. Incidence matrix

5. Which statement is true about file formats?
A. Edge lists encode whether the graph is directed
B. Edge lists don’t encode directedness; your loader (Graph vs DiGraph) decides
C. GraphML cannot store attributes
D. GEXF can’t store positions

Quiz

1. Best representation for **fast adjacency checks** (“is v a neighbor of u?”) on a sparse graph with low memory?
A. **Adjacency list (hash-based neighbor sets)** B. Edge list C. Dense adjacency matrix D. CSV table

2. You need repeated **matrix-vector multiplications** *Ax* for algorithms like PageRank on a **large graph**. Which fit best?
A. Edge list only B. **Sparse adjacency matrix (CSR/CSC)** C. Dense adjacency matrix D. Adjacency list

3. You must count **triangles** quickly. Which combo is generally most efficient?
A. Edge list + nested scans
B. **Sorted adjacency lists (set intersections) or sparse matrix ops**
C. Dense matrix + triple loops in Python
D. CSV join in pandas

4. You have a **tiny, dense** graph ($n \leq 2k$) and need constant-time adjacency checks. Best structure?
A. Edge list B. **Dense adjacency matrix** C. Adjacency list D. Incidence matrix

5. Which statement is **true** about file formats?
A. Edge lists encode whether the graph is directed
B. **Edge lists don’t encode directedness; your loader (Graph vs DiGraph) decides**
C. GraphML cannot store attributes
D. GEXF can’t store positions

Thảo luận

Chuẩn bị tuần trước

- Muddiest point: “Điểm nào mù mờ nhất sau khi đọc? (< 50 từ)”

Chuẩn bị tuần trước

- Câu hỏi định hướng (trả lời ngắn):
  - Compare edge list, adjacency list, adjacency matrix: memory & time trade-offs…
  - Compare NetworkX, SNAP.py, and Gephi. What's it great at? What's inconvenient?

Compare edge list, adjacency list, adjacency matrix

Aspect	Edge list	Adjacency list	Adjacency matrix (dense)
Memory	~O(m); very compact; attributes easy	~O(n + m); neighbors stored per node; great for sparse	O(n^2) booleans/weights; prohibitive for large sparse graphs
Iterate all edges	Excellent (linear scan)	Good (visit lists)	Poor for sparse (scan n^2 or compress)
Neighbor lookup (u → N(u))	Slow (scan/filter)	**Fast** (direct list/set)	Fast (row access) but heavy memory
Adjacency check (u,v?)	Slow (scan)	**Fast** with set/dict (amortized O(1))	**O(1)** direct index

Compare edge list, adjacency list, adjacency matrix

Add edge	Append (but may duplicate)	**Fast** (add to two lists/sets)	Need to set A[u,v] (OK) but memory already paid
Remove edge	Slow (find & delete)	Fast (remove from two lists/sets)	Fast index but still $O(n^2)$ memory
Triangle counting	Slow (triple scans)	**Fast** via neighbor-set intersections (sorted)	Fast via algebra (`trace(A³)/6`), but needs matrix ops & memory
Matrix algorithms (PageRank, spectral)	Not direct	Possible but awkward	**Natural** (`A`, `L`, eigensolvers; sparse preferred)

Compare NetworkX, SNAP.py, Gephi

- **NetworkX**
  - *Good at*: Pythonic, readable, rich algorithms, easy I/O, integrates with pandas/NumPy; rapid prototyping, teaching.
  - *Inconvenient*: Pure-Python; slow for very large graphs (≫1-2M edges) unless offloading heavy parts; not a visual tool.

- **SNAP.py**
  - *Good at*: High-performance kernels (C++ backend) accessible from Python; scalable basics (connected comps, triangles, k-core, PageRank) on bigger graphs.
  - *Inconvenient*: API less “pythonic”; fewer high-level conveniences than NetworkX; smaller ecosystem for attributes/data frames.

- **Gephi**
  - *Good at*: **Interactive visualization** (layouts, styling, partition/ranking), screenshots; quick exploration for class demos/stakeholders.
  - *Inconvenient*: Not a programming library; limited analytics; pipelines aren’t as reproducible as code unless carefully documented.

Hands-on lab

Setup

- Install python
  - with Anaconda
- Install IDE:
  - (Microsoft) vscode
  - Jupyter Notebook
- Install networkx
  - with pip
- Install gephi

Mini hands-on

Use the dataset in the assignment 1 on the course website.

1. Write code in *networkx* to load the network and print network properties, 2 cases: undirected and directed graphs
2. Visualize the graph with networkx and matplotlib
3. Load the dataset into *gephi* (export data to gexf format using networkx), visualize and and answer the questions about network properties

Mini hands-on

```
import networkx as nx
G = nx.read_edgelist('toy_edges.txt', create_using=nx.Graph)  # undirected…
print(G.number_of_nodes(), G.number_of_edges())
print(nx.average_clustering(G))
print(sorted(nx.degree(G), key=lambda x: x[1], reverse=True)[:5])


nx.write_gexf(G, 'toy_edges.gexf')  # write to gephi format
```

Chuẩn bị cho tuần tới

Chuẩn bị trước tuần sau

- Reading:
  - (Stanford CS224W L2) Measuring Networks & Random Graph Models

- Câu hỏi định hướng (trả lời ngắn):
  - Define small-world property. Which two metrics typically indicate it?
  - Sketch the degree distribution you expect for a social network.
    - Do you expect it to be heavy-tailed, and why?
  - What is a log-log plot of degree frequency?
    - What can it show better than a linear plot?
  - Define local vs. global clustering coefficient. What do high values mean?
  - If a graph shows short average path length but high clustering, what mechanism might explain it?
  - Name one pitfall when estimating degree distributions from sampled data.

- Muddiest point: “Điểm nào mù mờ nhất sau khi đọc? (< 50 từ)”

Thank you for listening

Q & A