CSE2525 · DATA MINING · 2026–27

Graph Mining

Representing graphs as matrices, measuring structure, finding central nodes and communities — with spectral clustering, Girvan–Newman, and node embeddings.

A. Anand · lecture 13 study deck

PART I

Graphs are everywhere — and massive

What a graph is, what questions it answers, and how we store one on a computer.

MANY REAL-WORLD DATASETS ARE GRAPHS

Why graphs?

People & web

- Social networks
- Web graphs (pages & links)
- Citation networks

Science

- Protein–protein interactions
- Molecules
- XML / parse trees

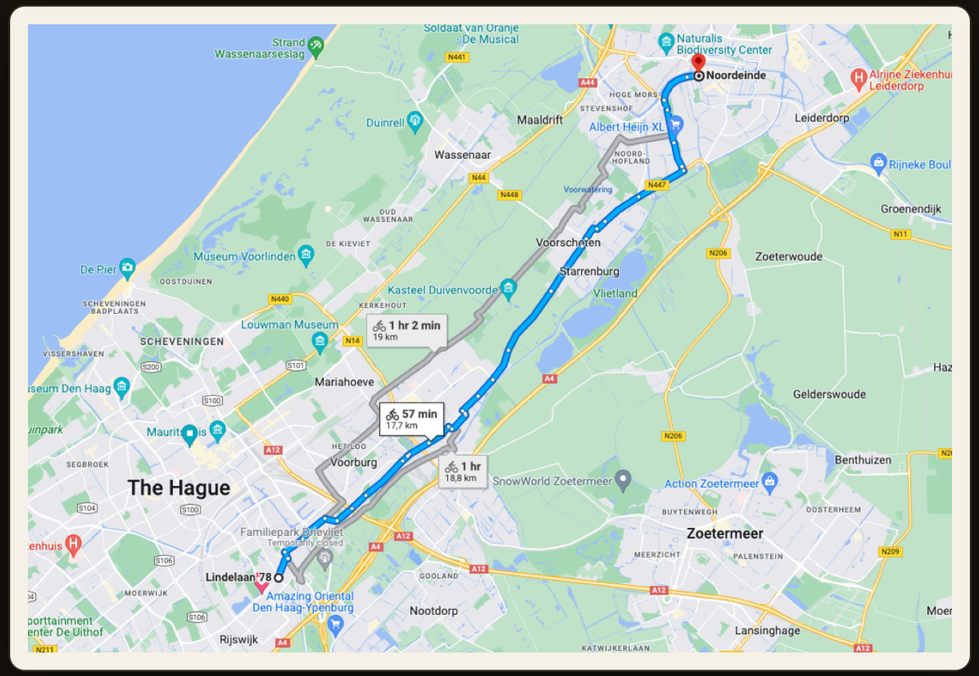
Infrastructure

- Road networks
- Internet routing
- Product / co-purchase graphs

These graphs are massive — we need *automatic* methods to understand them, not eyeballing.

WHAT'S POSSIBLE WITH GRAPHS · ROUTING

Finding shortest paths



A route is a shortest path on a weighted road graph. The same primitive powers internet routing and "degrees of separation" in social networks.

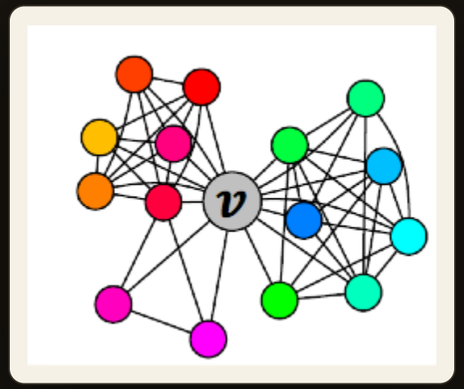
WHAT'S POSSIBLE WITH GRAPHS

Important nodes & communities

— Finding important nodes — social-media influencers & early adopters, authoritative web pages, key proteins. PageRank is the classic algorithm.

— Discovering communities — groups of developers / customers, clusters of related papers in a citation network.

— Both reduce to structure: who is connected to whom, and how densely.



THE VOCABULARY

What is a graph?

GRAPH G = (V, E)

A set of **vertices** V with $|V| = n$ and a set of **edges** E with $|E| = m$. Edges may be **directed** or **undirected**, and optionally **weighted**.

— Degree d(v) — number of edges at v. In a directed graph: in-degree and out-degree.

— Undirected: $\sum_v d(v) = 2m$ — every edge is counted at both ends.

— Directed: the sum of all out-degrees equals m (each edge leaves exactly one node).

HOW TO STORE A GRAPH

Three ways to store a graph

Adjacency matrix

- $n \times n$, entry $a_{ij}=1$ if edge i–j
- $O(n^2)$ space — dense, even if few edges
- Edge lookup in $O(1)$; enables linear algebra

Adjacency list

- Per node, a list of neighbours
- $O(n + m)$ space — great for sparse graphs
- Neighbour iteration is fast

Edge list

- Just the set of (u, v) pairs
- $O(m)$ space — simplest
- Slow for "are i, j adjacent?"

Real graphs are sparse ($m \ll n^2$), so lists save memory — but the matrix view unlocks spectral methods.

PART II

Graph properties

Distances, small-world and scale-free structure — the characteristics real networks share.

THE DATA-MINER'S WARNING

Why study graph properties?

KEY IDEA

Many real-world graphs share the same structural characteristics. Explaining why they appear is a core area of network research.

As data miners we must know the consequences: a "finding" that is fully explained by a generic property (e.g. a few huge-degree hubs) is not interesting — it is what any such graph would show.

DISTANCES IN A GRAPH

Eccentricity, radius & diameter

- Distance d(u,v) — the (weighted) length of the shortest path between u and v.
- Eccentricity e(v) — the maximum distance from v to any other vertex.
- Radius r(G) — the minimum eccentricity over all vertices.
- Diameter d(G) — the maximum eccentricity over all vertices.
- Effective diameter — the smallest value larger than the eccentricity of a *large fraction* of vertices (robust to a few outliers).

A PROPERTY OF REAL NETWORKS

The small-world property

SMALL-WORLD

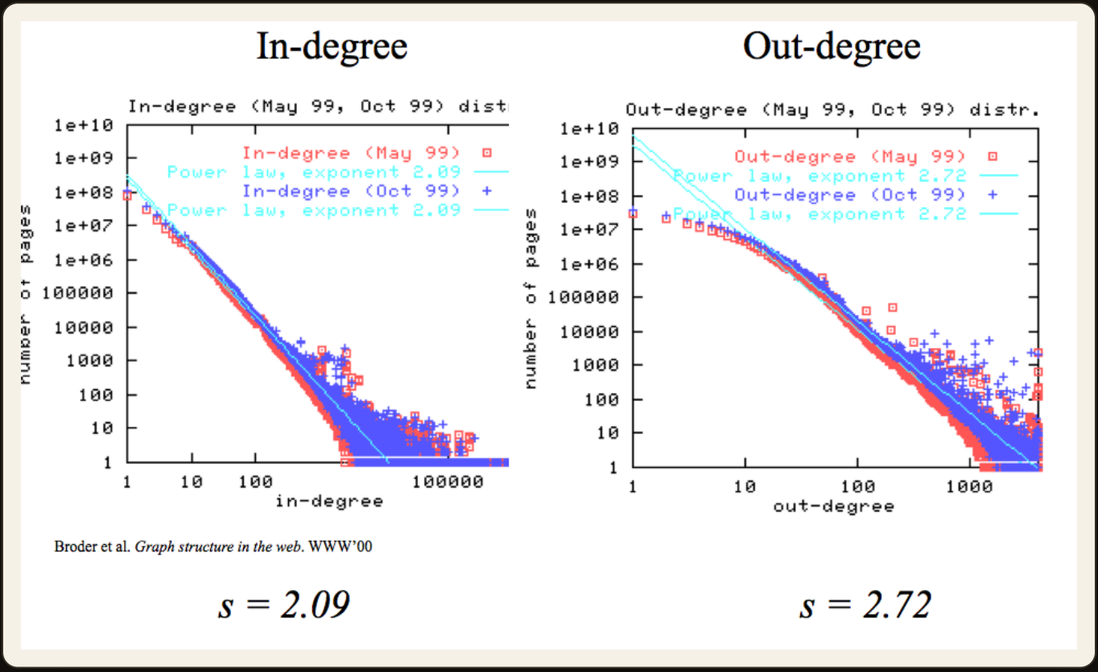
A graph is **small-world** if its average path length grows only **logarithmically** with the number of nodes — *avg path* $\propto$ *log n*. Any two nodes are reachable in surprisingly few hops.

— "Six degrees of separation" / the Kevin Bacon game; the Erdős number — how far a mathematician is from Paul Erdős.

— A large co-authorship network of 268K authors has radius 12 and diameter 23 — tiny for its size.

THE WORLD WIDE WEB · BRODER ET AL., WWW'00

Scale-free: power-law degrees



Web in-degree follows a power law with exponent $\gamma \approx 2.09$, out-degree $\gamma \approx 2.72$ — straight lines on a log–log plot. Most nodes have tiny degree; a few are huge hubs.

THE DEGREE DISTRIBUTION

The scale-free property

DEGREE DISTRIBUTION

f(k) = number of vertices with degree k. A graph is **scale-free** if $f(k) \propto k^{-\gamma}$ — a **power-law** distribution.

$$f(ck) = a(ck)^{-\gamma} = (a\, c^{-\gamma})\, k^{-\gamma} \propto k^{-\gamma}$$

"Scale-free" = rescaling the degree axis by c just rescales the curve — the shape is unchanged. The majority of vertices have small degree; a few hubs have very high degree.

QUESTION

In a network of millions, which **few** nodes act as **bridges** between communities — the ones whose removal would split the graph apart?

Not necessarily the most-connected nodes.

Think — then → for the answer.

PART III

Centrality — which nodes matter

Degree, eccentricity, closeness and betweenness — and Girvan–Newman for communities.

RANKING VERTICES BY IMPORTANCE

What is centrality?

DEFINITION

A centrality is a function $c : V \to \mathbb{R}$ that induces a **total order** on vertices. The higher c(v), the more **important / central** the vertex. The more central a node, the faster it reaches the rest of the graph.

- Degree & eccentricity centrality
- Closeness centrality
- Betweenness centrality

THE TWO SIMPLEST MEASURES

Degree & eccentricity centrality

DEGREE CENTRALITY

$c(v_i) = d(v_i)$ — simply the degree of the vertex. More neighbours = more central.

ECCENTRICITY CENTRALITY

$c(v_i) = 1 / e(v_i)$ — the **least eccentric** vertex is the most central; the most eccentric is **peripheral**.

MINIMISE THE AVERAGE DISTANCE

Closeness centrality

CLOSENESS

The vertex with the least **total distance** to all others is the centre:

$$c(x) = 1\overline{\sum_{y \neq x \in V} d(y, x)}$$

— Eccentricity centrality minimises the maximum distance.

— Closeness centrality minimises the average distance — this is the measure used to find the "centre of Hollywood".

LYING ON MANY SHORTEST PATHS

Betweenness centrality

BETWEENNESS

$$b(x) = \sum_{s \neq x \neq t} \sigma_{st}(x) \overline{\sigma_{st}}$$

σ_{st} = number of shortest paths s→t; $\sigma_{st}(x)$ = how many of them pass through x.

— Measures how many shortest paths route through a node.

— Also defined for edges (edge betweenness); high-betweenness edges are weak ties bridging communities.

COMPUTING BETWEENNESS EFFICIENTLY

The Girvan–Newman algorithm

Stage 1

Pre-processing & BFS

Root the graph at a node; run BFS top-down. Level = shortest-path length from root; each node's label = sum of its parents' labels (number of shortest paths from root).

Stage 2

Credit assignment

Work bottom-up, assigning credit to nodes and edges based on the BFS labels.

Stage 3

Aggregation

Repeat for every root node, then sum credits to get final betweenness.

GIRVAN–NEWMAN · THE BOTTOM-UP RULES

Credit assignment

— Each leaf gets credit 1.

— Each non-leaf node gets 1 + the credit flowing up from its children.

— A node with more than one parent splits its credit in proportion to the parents' BFS labels (the shares of shortest paths).

— Edge credit = the credit that flows from parent to child along that edge.

— In aggregation, divide every edge credit by 2 — each shortest path was counted from both endpoints.

Example: a node reached via B (label 3) and G (label 1) sends ¾ of its credit up to B and ¼ to G.

FROM BETWEENNESS TO COMMUNITIES

Clustering by removing weak ties

GIRVAN–NEWMAN COMMUNITY DETECTION

Edges between communities carry the most shortest paths — highest edge betweenness. So we cut them:

— Repeat until a suitable number of communities remains:

— — Compute the betweenness of every edge.

— — Remove the edge with the largest betweenness.

— The graph splits into connected components = communities. Divisive & hierarchical, but expensive (recompute after each cut).

PART IV

Spectral clustering

Represent the graph as a matrix, then use its eigenvectors to find communities.

WHAT MAKES A GOOD PARTITION?

Graph partitioning

A GOOD PARTITION

Split V into groups that **maximise within-group** connections and **minimise between-group** connections.

— Connected components are the easy case — no edges between them at all.

— Real communities are *almost* separate: dense inside, a few bridging edges outside.

— Spectral clustering uses properties of the graph matrix to find these cuts.

MATRIX REPRESENTATIONS · A 6-NODE EXAMPLE

Adjacency matrix A

	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	0	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	0	0	1	0	1
6	0	0	0	1	1	0

$n \times n$, $a_{ij} = 1$ if there is an edge between i and j.

***Symmetric** for undirected graphs → its eigenvectors are **real and orthogonal**.*

FROM A TO THE LAPLACIAN

Degree matrix D and Laplacian L = D − A

Degree matrix D (diagonal, d_{ii} = deg i)

3	0	0	0	0	0
0	2	0	0	0	0
0	0	3	0	0	0
0	0	0	3	0	0
0	0	0	0	3	0
0	0	0	0	0	2

Laplacian L = D − A (symmetric)

3	−1	−1	0	−1	0
−1	2	−1	0	0	0
−1	−1	3	−1	0	0
0	0	−1	3	−1	−1
−1	0	0	−1	3	−1
0	0	0	−1	−1	2

Every row of L sums to 0 → the all-ones vector is an eigenvector with eigenvalue $\lambda_1 = 0$.

WHY THE MATRIX VIEW HELPS

What does A·x mean?

NEIGHBOUR AVERAGING

If x assigns a label x_j to each node j, then $y = A \cdot x$ gives $y_i = \sum_{j \in N(i)} x_j$ — each entry is the **sum of its neighbours' labels**.

EIGENVECTORS

An eigenvector x with $A \cdot x = \lambda x$ is a labelling that is reproduced (up to scale λ) by neighbour-summing.

For a d-regular graph the all-ones vector is an eigenvector with eigenvalue λ = d. These special labellings encode global structure — the basis of spectral clustering.

THE SECOND EIGENVECTOR OF L

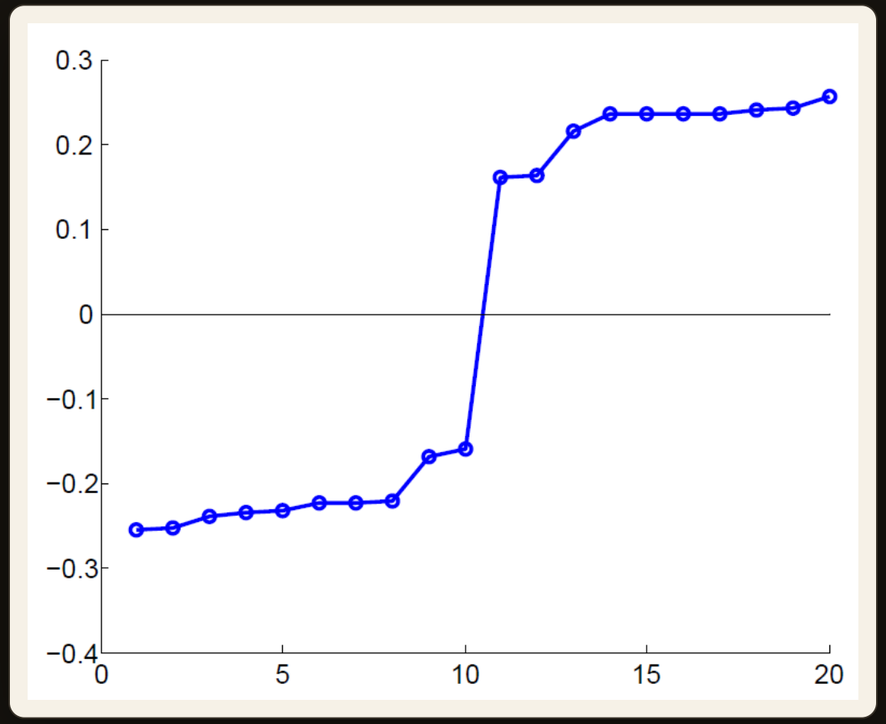
λ2 and the Fiedler vector

KEY FACT

The smallest Laplacian eigenvalue is $\lambda_1 = 0$. The **second** eigenvalue λ2 (algebraic connectivity) and its eigenvector **x2** — the **Fiedler vector** — reveal the best split.

— λ2 solves a relaxed minimum-cut optimisation.

— Sorting nodes by their value in x2 puts the two communities on opposite signs.



The sorted x2 values show a sharp jump near zero — a clean two-way split.

THE ALGORITHM IN THREE STAGES

Spectral clustering

1 · Pre-processing

Build the matrix

Construct a matrix representation of the graph — the (unnormalised) Laplacian L = D − A.

2 · Decomposition

Eigen-decompose

Compute eigenvalues/eigenvectors of L. Map each node to a low-dimensional representation using one or more eigenvectors (start with x_2).

3 · Grouping

Assign clusters

Split the nodes in the new representation into two (or more) clusters.

STAGE 3 · WHERE TO SPLIT

Choosing the splitting point

— Sort the components of the reduced 1-D vector (x_2); identify clusters by splitting the sorted vector in two.

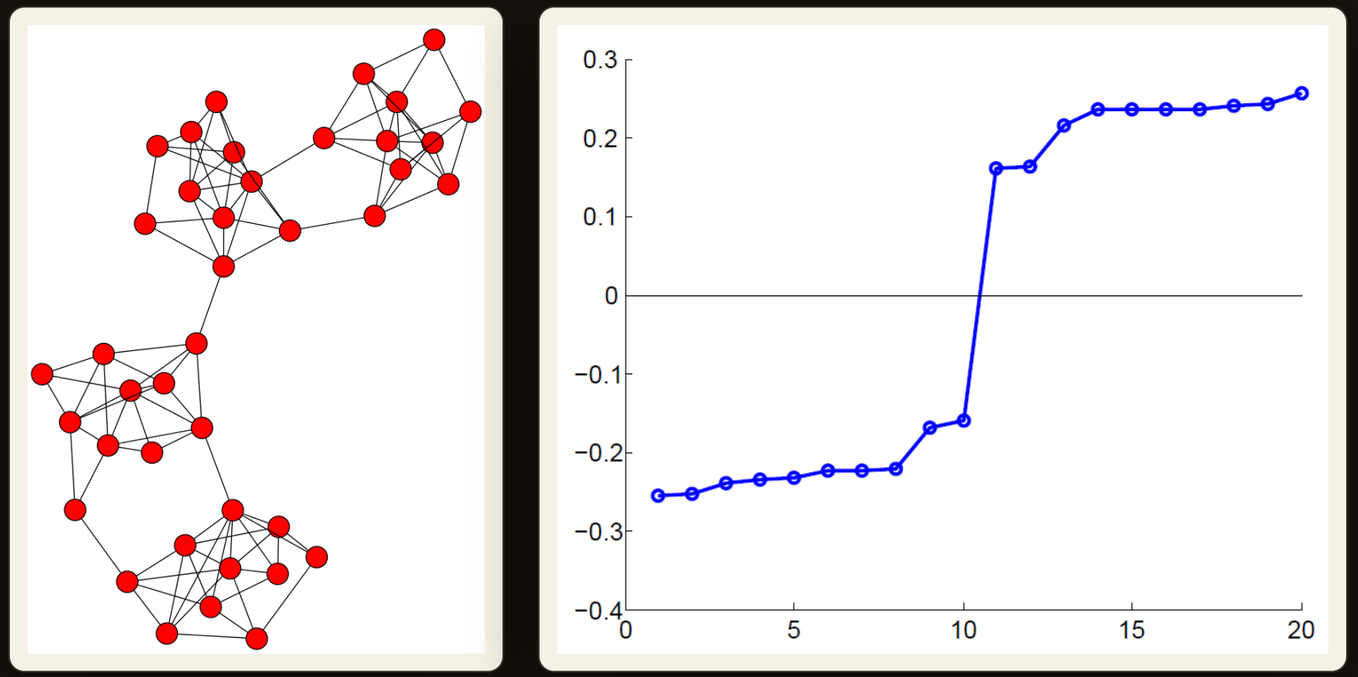
— Naïve: split at 0 or at the median — cluster A = positive values, cluster B = negative.

— More expensive: sweep over the node ordering induced by x_2 and pick the split that minimises the normalized cut.

The eigenvector gives the *order*; the sweep picks the single best *cut point* along it.

SPECTRAL PARTITIONING IN ACTION

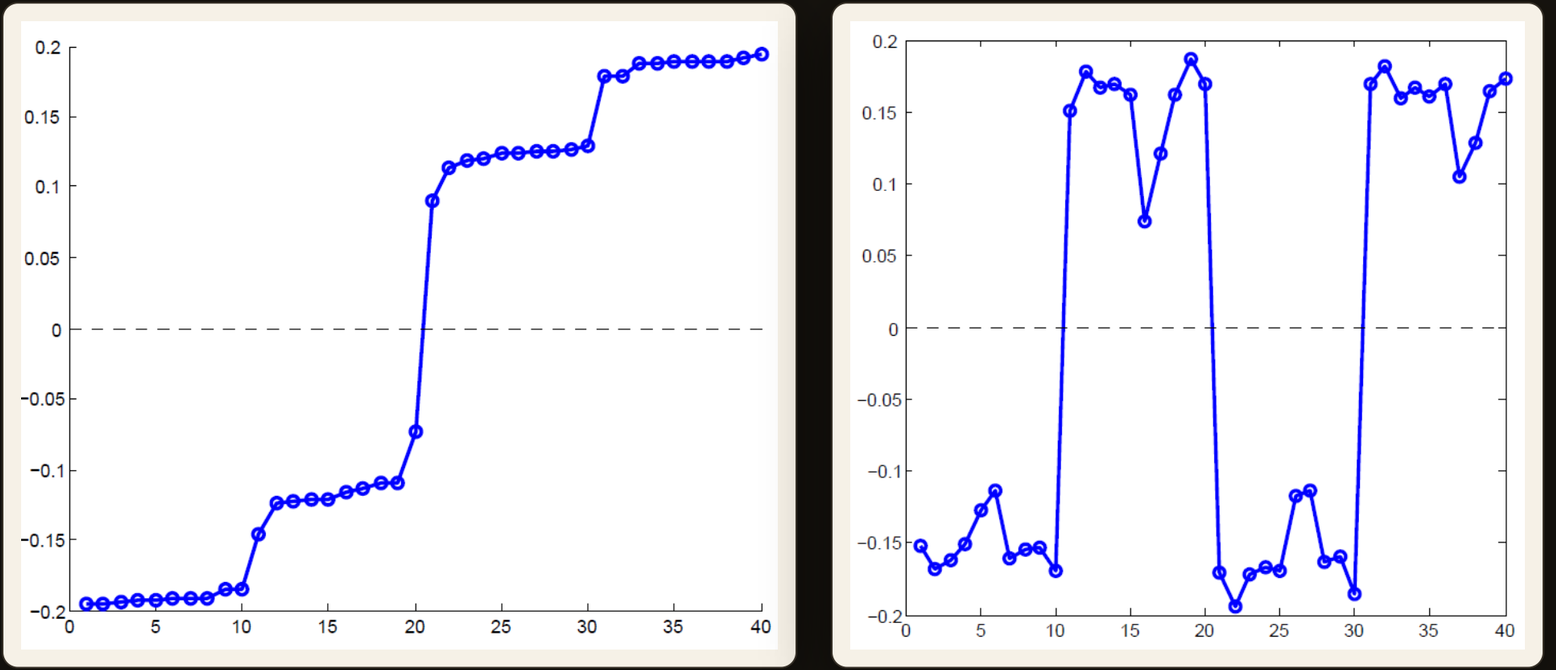
Example: two communities



Left: dense clusters joined by sparse bridges. Right: the sorted Fiedler vector x2 — the single jump at the middle is exactly where the two communities separate.

MORE THAN TWO COMMUNITIES

Reading several clusters from the spectrum



The sorted x_2 (left) shows several plateaus — candidate groups. Using more eigenvectors (x_3, right) exposes additional block structure for finer partitions.

PARTITIONING INTO K CLUSTERS

k-way spectral clustering

Recursive bi-partitioning

- Hagen et al., 1992
- Apply the 2-way split recursively, hierarchically
- Disadvantage: **inefficient** and **unstable**

Multiple eigenvectors

- Shi–Malik, 2000
- Build a reduced space from the **first k eigenvectors**, then cluster (e.g. k-means)
- The **preferred** modern approach

Using several eigenvectors at once gives a stable k-dimensional embedding of each node.

EIGENVECTORS AS NODE EMBEDDINGS

From eigenvectors to embeddings

SPECTRAL EMBEDDING

Stacking a node's values across the top eigenvectors gives a **low-dimensional vector** per node — a **graph embedding** where connected / similar nodes land close together.

TRANSDUCTIVE VS INDUCTIVE

Transductive: the embedding is computed for the fixed graph you have — a new node means recomputing the eigenvectors.

Inductive: learn a function that embeds any node, so unseen nodes get vectors without redoing the decomposition.

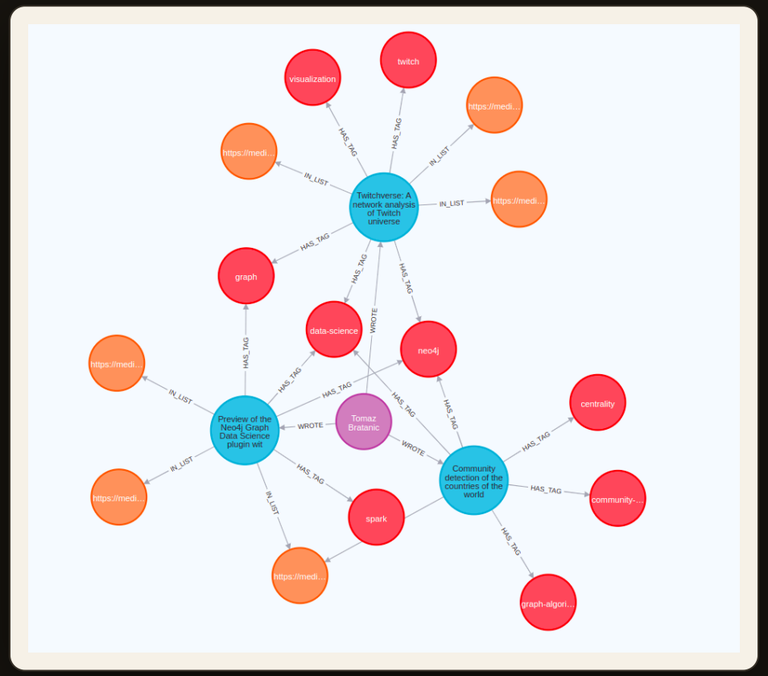
PART V

Node embeddings & the lab

Random walks, Node2Vec / DeepWalk, and the Medium-articles assignment.

LAB ASSIGNMENT · MEDIUM ARTICLES

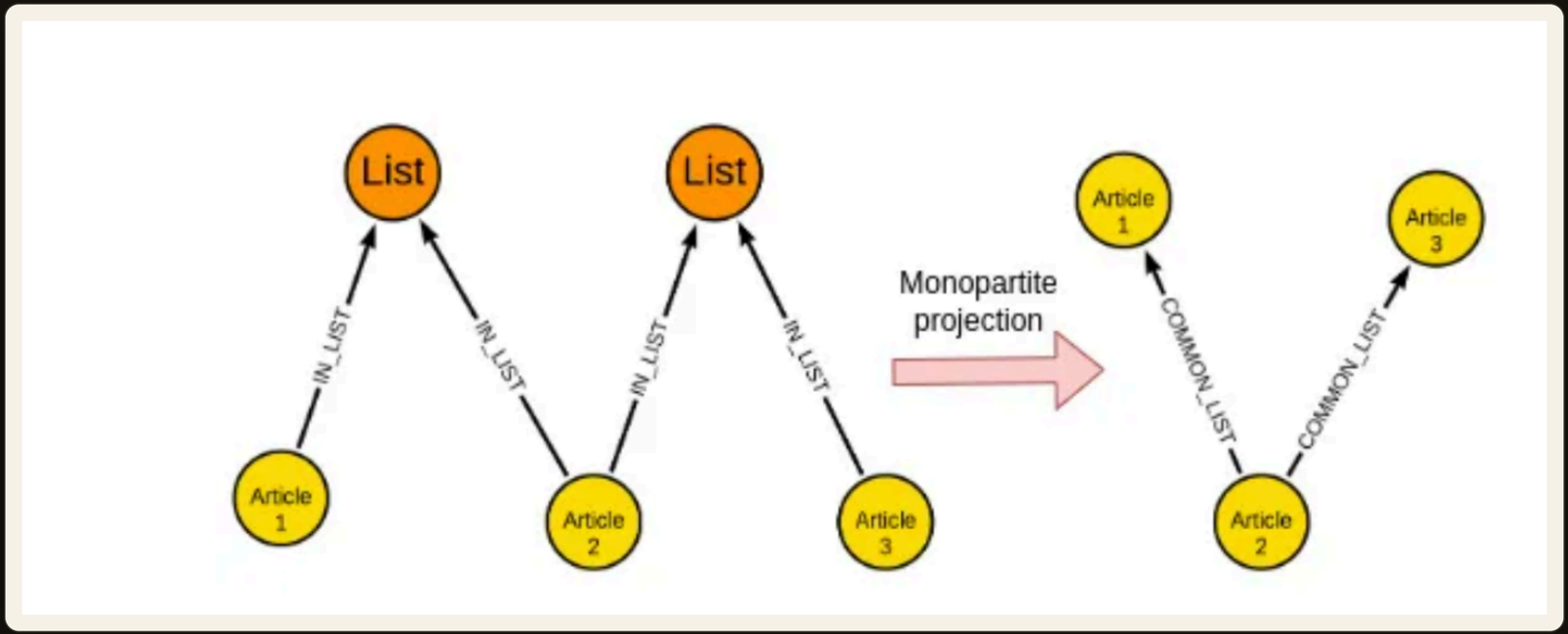
A real-world graph: Medium articles



Articles connect to tags, reading lists and authors. Task: a 3-way classification of articles into software-dev, AI and UX from the graph structure.

BUILDING A NODE GRAPH FROM RELATIONS

Monopartite projection



The raw graph is bipartite (articles ↔ lists). Project it to an article–article graph: two articles sharing a list get a COMMON_LIST edge — now spectral / walk methods apply.

THE ASSIGNMENT PIPELINE

Dataset → embedding → classifier

1
Graph construction
Build the Medium article graph from subscription / tag / list relations.

2
Random walks & embedding
Generate node embeddings — spectral (Laplacian) or random-walk based.

3
Classification
Feed embeddings to a classifier for the 3-way topic prediction.

Two embedding routes to compare: spectral (eigenvectors of L) vs random walks.

TWO WAYS TO EMBED NODES

Spectral embeddings vs random walks

Spectral embeddings

- Build the (unnormalised) Laplacian L
- Compute its eigenvalues & eigenvectors
- Use the top eigenvectors as node vectors

Random walks

- Start at a node, randomly traverse edges
- Fixed-length walks from **every** node
- Walks → "sentences" → embeddings

Random walks power the popular node-embedding methods DeepWalk and Node2Vec: treat each walk as a sentence of node-IDs and learn vectors with a word2vec-style model.

LECTURE 13 · WRAP-UP

Summary

TAKE AWAY

Graphs are everywhere and large — stored as adjacency matrix / list / edge list. Real networks are small-world (log paths) and scale-free (power-law degrees).

Centrality ranks nodes — degree, eccentricity, closeness, betweenness; Girvan–Newman cuts high-betweenness edges to find communities.

Spectral clustering uses the Laplacian's eigenvectors (the Fiedler vector x_2) to partition a graph, and generalises to node embeddings — spectral or random-walk based (DeepWalk, Node2Vec).