Lecture 1 - Introduction to Graph Mining

Sunday, January 11, 2026 5:43 PM

Plan for today:

- About me and the course ✓
- Go over syllabus ✓
- Go over website ✓
- Discuss project ✓
- Discuss paper presentations and lit review ✓
- Lecture
  - Define graphs
  - Real-world graph properties
  - Graph Mining and applications
  - Graph computations and processing
  - NetworkX and basic connected components

Project details

- Groups of 1-2
- Take a look at the website's data repositories
- Only requirement
  → must involve graph data

Prior project ideas:

* Predicting chess match outcomes via a competition network
* Vertex centrality to identify population centers on a road network

* stock market / coin price prediction via transaction networks

* Recommender systems using SNAP data

Grad students: use your own ongoing project (if it can use graphs)

Everyone: find something you're interested in

Computer resources:

- CCI for multi-GPU or distributed systems
- Slota's systems
  * 2TB large-mem
  * 4x A100 GPU system

Lit review (G280)

Go through if an undergrad

Go through if an undergrad wants to do one, you can replace your paper presentations

→ 5+ relevant papers to your project or graduate work

If you've done a research qualifier, you can usmg the sme papers

Presentation
→ 30-ish minutes

Writeup
→ short summary that would be expected for an Intro/Background section in a real publication

---

Paper presentations (4280)

- Two papers for each student
- Selected first-come first-served

- Selected first-come first-served

- 15-ish minute presentations

Homeworks (for both 4/6000-level)

2-4 problems individually/as a team

→ present solution to the class

TBD on specifics

Graph Definitions

Graph is tuple of:

$G = \{ V(G) = \text{vertex set}$

$E(G) = \text{edge set}$

$W_v(G) = \text{vertex weights}$

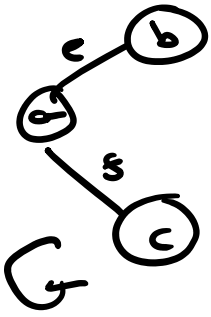
$W_E(G) = \text{edges weights}$

$D_v(G) = \text{vertex meta-data}$

$D_E(G) = \text{edge meta-data} \}$

$G = \{ V, E, W_v, W_E, D_v, D_E \}$

$G = \{V, E\}$ → at a minimum



$V(G) = \{a, b, c\}$

$E(G) = \{e = (a,b), f = (a,c)\}$

For our GM context:

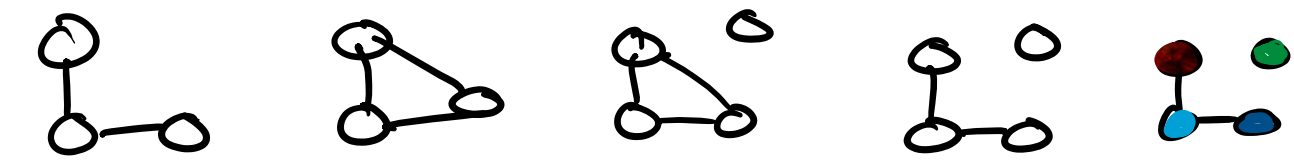
vertices => represent discrete people, places, entities, etc.

edges => representation of some form of connection or interaction between those people, entities, etc.

weights => define strength of connections (for edges) or importance (for vertices)

metadata => generic catch-all for other vertex/edge labels or properties

Temporal graphs => graphs that change over time



$G_{t=0}$ $G_{t=1}$ $G_{t=2}$ $G_{t=3}$ $G_{t=4}$

Real world graph

↳ social networks, info networks, biological network

These graphs all have similar properties

Sparsity: $|E(G)| << |V(G)|^2$

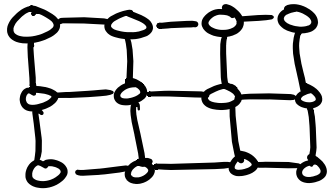
Degree skew: # of low-degree vertices >>> high degree vertices

Hubs: high degree or otherwise "important" vertices

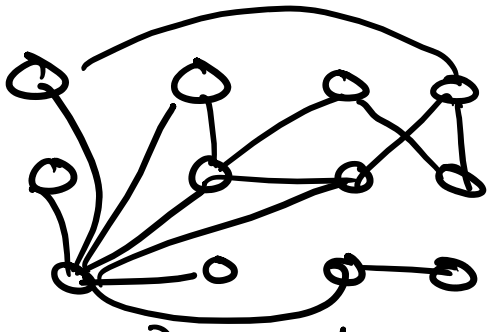
↳ a major topic we'll discuss in detail

Irregularity: information, social, etc. networks are often not physically constrained

physically constrained



regular



irregular

Small-world: average shortest path length is small relative to $|V(G)|$

"6 degrees of Kevin Bacon"

Note ✉: Properties are typical but not every real graph has all of them (or any of them)

Graph Processing

"Vertex-centric" processing

→ Each $v \in V(G)$ has some state $S(v)$

→ We iteratively update each $S(v)$ using the state's of v's neighbors

using the states of its neighbors

Big idea: many (if not all) graph computations can be represented by this basic computation and model

Algorithmically:

Input: $G = \{V, E\}$

For all $v \in V$:
  $S(v) =$ initialize()

For some # of iterations:
  updateAlgoState()

  For all $v \in V$:
    For all $(v, u) \in E$:
      $S(v) = f(S(v), S(u))$

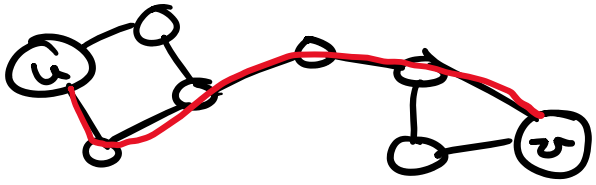
  same computation over all edges

For all $v \in V$:
  $S(v) =$ finalize()

Connectivity decomposition

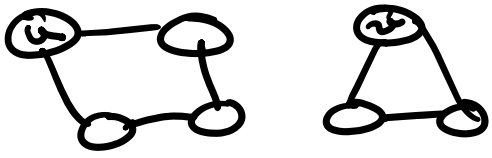
↳ u, v are connected if there exists a path along edges

exists a path along edges from u to v



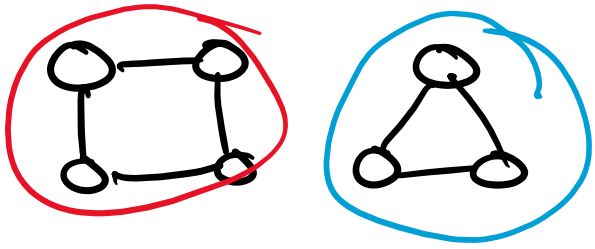
u, v are connected

G is connected if for all possible u, v pairs, u and v are connected

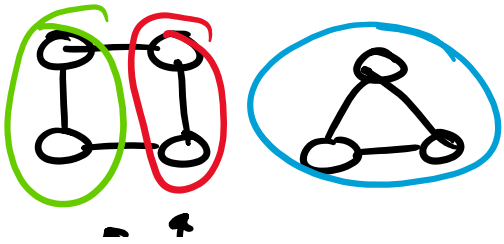


G' is <u>not</u> connected

Connectivity decomposition: find all maximal connected subgraphs in G



maximal connected components



not maximal → not a conn. decomp.

EZ way to solve connectivity using our framework:

Each $S(w)$ is set to unique values

do
  all vertices assume max label over their 1-hop neighborhood
while some vertex updates its state

Output: each unique label correspond to a unique component