Graduate Self-Study Plan

Graph Networks, Community Detection, Louvain & Leiden — Applied to LibTrails Digital Libraries

Prepared for: Sean | Date: 2026-02-12

This plan turns our chat into a structured, graduate-level self-study sequence for learning graph/network fundamentals and modern community detection, with an applied track for building semantic community maps inside your LibTrails digital library pipeline (topic/chunk graphs, kNN similarity graphs, and navigation graphs).

What you'll be able to do by the end

- Explain community detection objectives (modularity, CPM, Map Equation) and their assumptions.
- Implement Louvain and Leiden on weighted graphs; interpret the role of the resolution parameter gamma.
- Diagnose failure modes (resolution limit, disconnected communities, hubness in embedding graphs).
- Design evaluation protocols (stability across seeds, NMI/VI, size distributions, synthetic benchmarks).
- Ship an interpretable "library map" feature: multi-resolution thematic clusters + overlap handling + browse UX.

How to use this document

Each phase includes: (1) readings (with arXiv IDs), (2) the key ideas to extract, (3) a coding checkpoint, and (4) how it applies to LibTrails. If you do only one thing beyond Leiden: add CPM and do multi-resolution sweeps with stability evaluation.

Prerequisites and Setup

You can complete this plan with Python + NetworkX + (optional) igraph/leidenalg. For larger graphs, consider GPU acceleration via NVIDIA nx-cugraph.

- **Python stack:** networkx, numpy, scipy, pandas, matplotlib; plus one of: (a) leidenalg + igraph, (b) graspologic (SBM), (c) infomap package.
- **Graph representations:** store weighted edges; preserve node metadata (book_id, chunk_id, topic_id, embedding centroid, author, year).
- **Two graph types for LibTrails:** (1) *Similarity graphs* (kNN on embeddings) and (2) *Flow graphs* (navigation/clickstream, citation, recommendation transitions).
- **Core practice:** always run multi-resolution sweeps and stability checks; never trust a single partition.

Suggested folder structure

```
repo/ docs/graph-study-plan.pdf notebooks/ 01_graph_basics.ipynb
02_louvain_vs_leiden.ipynb 03_gamma_sweeps_stability.ipynb 04_cpm_vs_modularity.ipynb
05_infomap_flow_graphs.ipynb 06_overlap_link_communities.ipynb src/libtrails_graph/
build_graph.py community.py evaluate.py viz.py
```

Roadmap at a Glance

Phase	Goal	Key readings (arXiv)	Primary deliverable
0	Community detection mental model + vocabulary	0906.0612	Notes: definitions, taxonomy, evaluation metrics
1	Modularity fundamentals	cond-mat/0308217	Implement modularity score + null model intuition
2	Why gamma matters (resolution limit)	cond-mat/0603718	Gamma sweep plan + expected effects
3	Louvain mechanics	0803.0476	Run Louvain on your similarity graph; inspect pathologies
4	Leiden guarantees + practice	1810.08473	Leiden baseline + stability across seeds
5	CPM objective for similarity graphs	1104.3083	Leiden-CPM multi-resolution + interpretability checks
6	Flow-based communities	0906.1405	Infomap on navigation/citation graphs; compare with Leiden
7	Generative baseline (SBM)	1008.3926	Use SBM as a diagnostic: is structure real or artifact?
8	Overlap + validation	0903.3178 0805.4770 0709.2938	Overlapping/edge communities + synthetic benchmark harness

Recommended pacing: ~8 weeks at 4–6 focused hours/week. If you want faster: do phases 0–5 first (the Leiden+CPM core), then add flow (phase 6) once you have navigation data.

Phase 0 — Community Detection: What It Is (and Is Not)

Goal: Build a clean mental model: community definitions, taxonomy, evaluation, and why “the” correct partition rarely exists.

Readings

- Fortunato (survey): arXiv:0906.0612 — https://arxiv.org/abs/0906.0612

What to extract

- Different notions of community (density-based, flow-based, statistical/generative).
- Hard partitions vs overlapping vs hierarchical communities.
- Why evaluation is difficult without ground truth; which metrics are used anyway (NMI/VI, modularity, conductance).
- Common failure modes: resolution limit, degeneracy (many near-optimal partitions), sensitivity to small perturbations.

Coding checkpoint

Write a one-page “community definition for LibTrails” that states: what nodes represent, what edges represent, and what a ‘community’ should mean for a user.

How this applies to LibTrails

LibTrails is inherently multi-theme. Treat community detection as a user-facing organizing lens, not as an objective truth. Your product goal is an interpretable, navigable map that remains stable under small changes.

Phase 1 — Modularity Fundamentals and the Null Model

Goal: Understand modularity as an objective and why its null model matters.

Readings

- Newman (modularity): arXiv:cond-mat/0308217 — https://arxiv.org/abs/cond-mat/0308217

What to extract

- Modularity as: observed intra-community edges minus expected intra-community edges under a degree-preserving null model.
- The configuration model intuition: why degree distributions dominate expectations.
- Weighted modularity: what changes when edges have weights.

Coding checkpoint

Implement modularity scoring for a partition of your weighted graph. Verify your score matches a library implementation on small graphs.

How this applies to LibTrails

In embedding-derived kNN graphs, degree and weight distributions can be artifacts of geometry (hubness). Understanding the null model helps you avoid overclaiming semantic ‘communities’ that are actually degree effects.

Phase 2 — The Resolution Limit: Why Gamma Sweeps Are Mandatory

Goal: Learn why modularity can miss small communities and how resolution parameters control scale.

Readings

- Fortunato & Barthelemy (resolution limit): arXiv:cond-mat/0603718 — https://arxiv.org/abs/cond-mat/0603718

What to extract

- Why modularity maximization can merge small real communities into larger ones.
- How resolution interacts with graph size and total edge weight.
- Practical implication: single-run clustering is not defensible.

Coding checkpoint

Design a gamma sweep grid and decide your tracking metrics: number of communities, size distribution, stability across random seeds, and a small sample of human-read ‘cluster labels’.

How this applies to LibTrails

LibTrails will have both micro-topics (niche) and macro-themes (genres). Gamma sweeps let you expose a zoomable ‘map’ where users can traverse scales instead of forcing one partition.

Phase 3 — Louvain: Greedy Multi-Level Modularity Optimization

Goal: Learn the classic Louvain method and its multi-level aggregation mechanics.

Readings

- Blondel et al. (Louvain): arXiv:0803.0476 — https://arxiv.org/abs/0803.0476

What to extract

- Two phases: local moving + aggregation; why it scales.
- Sensitivity to node order / randomness; local optima.
- Pathology: communities can be internally disconnected.

Coding checkpoint

Run Louvain on: (a) a toy graph where you know the answer and (b) your LibTrails similarity graph. Inspect for disconnected communities and unstable results across seeds.

How this applies to LibTrails

Louvain is valuable as a baseline and a cautionary tale. If your clusters split into disconnected pieces, users will experience incoherent shelves. This motivates Leiden.

Phase 4 — Leiden: Connectivity Guarantees and Better Partitions

Goal: Understand how Leiden fixes Louvain and why it is the default for production.

Readings

- Traag, Waltman & van Eck (Leiden): arXiv:1810.08473 — https://arxiv.org/abs/1810.08473

What to extract

- Refinement step and guarantees about well-connected communities.
- Why convergence is more reliable than Louvain.
- Practical parameters: random seed, iterations, resolution.

Coding checkpoint

Establish your production baseline: Leiden (modularity) with a documented gamma sweep. Record stability (NMI/VI) across seeds and small graph perturbations (e.g., remove 1% edges).

How this applies to LibTrails

Leiden is the right default for LibTrails similarity graphs. The refinement step typically yields more coherent thematic groupings and avoids ‘split cluster’ UX issues.

Phase 5 — Constant Potts Model (CPM): A Better Fit for Similarity Graphs

Goal: Add CPM as an alternative objective that often behaves more predictably on embedding graphs.

Readings

- Traag, Van Dooren & Nesterov (CPM): arXiv:1104.3083 — https://arxiv.org/abs/1104.3083

What to extract

- CPM objective and why it avoids the modularity resolution limit in a precise sense.
- How the CPM resolution parameter directly controls minimum internal density.
- When CPM beats modularity: geometric graphs, kNN graphs, and graphs with strong local neighborhoods.

Coding checkpoint

Run Leiden under both objectives (modularity vs CPM) across the same gamma grid. Compare: stability, community size distribution, and semantic coherence of labels.

How this applies to LibTrails

For LibTrails, CPM often yields more intuitive ‘topic neighborhoods’ on kNN similarity graphs, making it easier to produce clean shelves and map regions without surprise merges.

Phase 6 — Flow-Based Communities: Infomap and the Map Equation

Goal: Learn when you should prefer flow-based communities (navigation, citations, transitions) over density-based objectives.

Readings

- Rosvall & Bergstrom (Infomap): arXiv:0906.1405 — https://arxiv.org/abs/0906.1405

What to extract

- Why flow-based objectives capture ‘paths’ through a network.
- Interpretation: compressing random walks (codebooks) instead of counting edges.
- Why Infomap can reveal different, often more navigable, partitions than modularity/CPM.

Coding checkpoint

If you have navigation signals (clicks, reading sequences, recommendations), build a directed/weighted flow graph and run Infomap. Compare the resulting communities against Leiden on the same node set.

How this applies to LibTrails

LibTrails can combine two maps: a semantic similarity map (CPM/Leiden) and a user-flow map (Infomap). Agreement suggests robust themes; disagreement highlights ‘semantic vs behavioral’ differences worth surfacing.

Phase 7 — A Statistical Baseline: Degree-Corrected Stochastic Block Models

Goal: Adopt a generative lens to detect when communities are real vs artifacts of degree/geometry.

Readings

- Karrer & Newman (DC-SBM): arXiv:1008.3926 — https://arxiv.org/abs/1008.3926

What to extract

- Why standard SBM fails with heavy-tailed degrees; how degree correction fixes it.
- Interpretation: communities as parameters of a generative process (likelihood).
- Using SBM as a diagnostic rather than a production algorithm.

Coding checkpoint

Fit a simple SBM/DC-SBM (even on a subgraph) and compare partitions to Leiden. Use disagreements to diagnose hubness, overly generic topics, or edge-weight issues.

How this applies to LibTrails

Embedding graphs often contain hubs (generic chunks/topics). A generative sanity check helps you avoid building product features on clusters that are driven by hubs rather than meaning.

Phase 8 — Overlap and Validation: Link Communities, LFR Benchmarks, and Baselines

Goal: Round out your toolkit: overlap handling and evidence-based evaluation.

Readings

- Ahn, Bagrow & Lehmann (Link communities): arXiv:0903.3178 — https://arxiv.org/abs/0903.3178
- Lancichinetti, Fortunato & Radicchi (LFR benchmark): arXiv:0805.4770 — https://arxiv.org/abs/0805.4770
- Raghavan, Albert & Kumara (Label propagation): arXiv:0709.2938 — https://arxiv.org/abs/0709.2938

What to extract

- Why overlap is natural in text libraries (books belong to multiple themes).
- How LFR produces graphs with known ground truth for testing parameter sensitivity.
- Why you should always keep a simple baseline (label propagation) to prevent over-fitting complexity.

Coding checkpoint

Build a small evaluation harness: generate LFR graphs; compare Louvain/Leiden/CPM/Infomap; report stability and accuracy vs ground truth. Then apply the same metrics to LibTrails graphs (without ground truth).

How this applies to LibTrails

Overlap support enables richer LibTrails UX: a book can appear in multiple regions; chunks can bridge themes; authors can sit at intersections. Your evaluation harness prevents ‘pretty maps’ that collapse under perturbations.

Applied Track — Building the LibTrails Library Map

Use this track alongside the phases above. The goal is to move from ‘I can run Leiden’ to ‘I can ship a stable, interpretable, multi-resolution community map for a digital library’.

- **A. Define graph objects**: Choose node types (chunks, books, topics, authors). Start with two graphs: chunk-similarity (kNN) and book-similarity (aggregate chunk signals).
- **B. Edge construction**: Prefer mutual-kNN; store cosine similarity as weights; consider weight transforms (e.g., max(sim-τ,0)) to suppress weak ties.
- **C. Multi-resolution sweep**: Run Leiden with modularity and CPM across gamma grid. Track: #communities, size distribution, stability (VI/NMI across seeds), and human interpretability (auto labels from top terms).
- **D. Stability-first selection**: Pick 2–3 resolutions that are stable and useful (micro, meso, macro). Make these zoom levels in the UI.
- **E. Overlay flow communities** (optional): If you capture user navigation, run Infomap and display ‘paths’ or ‘corridors’ between semantic regions.
- **F. Overlap**: Add link-community or soft assignment (e.g., top-2 communities by affiliation strength) for boundary objects (books bridging themes).
- **G. Product UX**: Provide a ‘map’ view + ‘shelves’ view. Each community should have: label, centroid exemplars, representative books, and edges to neighboring communities.

Recommended evaluation signals (no ground truth)

- **Stability**: partitions should be consistent across random seeds and small edge perturbations.
- **Coherence**: community label quality (top keywords/topics) and exemplar similarity.
- **Separation**: neighboring communities should have distinct labels; avoid duplicates caused by fragmentation.
- **Coverage**: avoid a ‘giant misc cluster’; tune gamma/graph construction instead.
- **User utility**: can a user quickly find a relevant shelf/region and jump between adjacent themes?

References and Links

Primary arXiv papers (core + additions)

- Fortunato, ‘Community detection in graphs’ — arXiv:0906.0612 — https://arxiv.org/abs/0906.0612
- Newman, ‘Modularity and community structure in networks’ — arXiv:cond-mat/0308217 — https://arxiv.org/abs/cond-mat/0308217
- Fortunato & Barthelemy, ‘Limits of modularity maximization…’ — arXiv:cond-mat/0603718 — https://arxiv.org/abs/cond-mat/0603718
- Blondel et al., ‘Fast unfolding of communities…’ (Louvain) — arXiv:0803.0476 — https://arxiv.org/abs/0803.0476
- Traag, Waltman & van Eck, ‘From Louvain to Leiden…’ — arXiv:1810.08473 — https://arxiv.org/abs/1810.08473
- Traag, Van Dooren & Nesterov, ‘Narrow scope for resolution-limit-free…’ (CPM) — arXiv:1104.3083 — https://arxiv.org/abs/1104.3083
- Rosvall & Bergstrom, ‘The map equation’ / Infomap — arXiv:0906.1405 — https://arxiv.org/abs/0906.1405
- Karrer & Newman, ‘Stochastic blockmodels and community structure…’ (DC-SBM) — arXiv:1008.3926 — https://arxiv.org/abs/1008.3926
- Ahn, Bagrow & Lehmann, ‘Link communities…’ — arXiv:0903.3178 — https://arxiv.org/abs/0903.3178
- Lancichinetti, Fortunato & Radicchi, ‘Benchmark graphs for testing…’ (LFR) — arXiv:0805.4770 — https://arxiv.org/abs/0805.4770
- Raghavan, Albert & Kumara, ‘Near linear time algorithm…’ (Label propagation) — arXiv:0709.2938 — https://arxiv.org/abs/0709.2938

High-quality non-arXiv resources

- Barabasi, ‘Network Science’ (free online book): https://networksciencebook.com/
- Stanford CS224W (Graph ML course): https://cs224w.stanford.edu/
- leidenalg documentation: https://leidenalg.readthedocs.io/en/stable/intro.html
- NVIDIA blog (GPU Leiden / nx-cugraph): https://developer.nvidia.com/blog/how-to-accelerate-community-detection-in-python-using-gpu-powered-leiden/

Tip: keep Fortunato (0906.0612) open for conceptual grounding, and the Leiden paper + leidenalg docs open for implementation details while coding.