Analysis and Recommendation in Synthetic Friendship Networks Using Graph Theory

Algorithm Analysis & Design — Final Project

Sarthak Mishra (2024117007) Amay Sharma (2024101095) Yashav Bhatnagar (2024101030) Lasya Katari (2024115004) Kartik Thapa (2024115009)

International Institute of Information Technology, Hyderabad

December 2025

Outline

Project Overview

Objective

Analyze synthetic social networks using graph theory algorithms to:

- Identify connected components and network structure
- Measure node importance through centrality metrics
- Recommend new friendships based on network topology and user attributes
- Detect community structures within the network

Key Features

- Synthetic Facebook-like friendship graphs
- Personality tags (interests: sports, music, tech, travel)
- All algorithms implemented **from scratch**
- Comprehensive empirical analysis and visualization

Motivation

Why Social Networks?

- Ubiquitous in modern life
- Reveal human behavior patterns
- Model information spread
- Network resilience analysis

Why Synthetic Data?

- Controlled environment
- Privacy-preserving
- Scalability testing
- Reproducible results

Real-World Applications:

- Social media optimization
- Marketing and influence
- Epidemiological modeling
- Community detection
- Friend recommendation systems

Graph Properties:

- Nodes = Users
- Edges = Friendships
- Tags = Interests/Attributes
- Scalable: 10,000+ nodes

Breadth-First Search (BFS)

Algorithm:

- Level-by-level exploration
- Queue-based implementation
- Visits all neighbors before going deeper
- Finds shortest paths in unweighted graphs

Use Cases:

- Connected components detection
- Shortest path finding
- Distance computation
- Level-order traversal

Complexity Analysis:

Time Complexity

$\Theta(n + m)$

Space Complexity

$\Theta(n)$ for queue and visited set

Key Properties:

- Complete: finds all reachable nodes
- Optimal: shortest paths guaranteed
- Cache-friendly: sequential access

Depth-First Search (DFS)

Algorithm:

- Explores as far as possible along each branch
- Stack-based (iterative) or recursive
- Backtracks when no unvisited neighbors
- Memory-efficient for wide graphs

Use Cases:

- Connected components
- Cycle detection
- Topological sorting
- Path exploration

Complexity Analysis:

Time Complexity

$\Theta(n + m)$

Space Complexity

$\Theta(n)$ (worst case: long chain)

Two Variants:

- **Recursive:** Cleaner code, function call overhead
- **Iterative:** Explicit stack, avoids stack overflow

Union-Find (Disjoint Set Union)

Algorithm:

- Tracks disjoint sets efficiently
- Two main operations:
  - `find`: Get set representative
  - `union`: Merge two sets
- Path compression optimization
- Union by rank/size

Use Cases:

- Connected components
- Kruskal's MST algorithm
- Dynamic connectivity
- Network connectivity queries

Complexity Analysis:

Time Complexity

$O(m \cdot \alpha(n))$ for m operations
where $\alpha(n)$ is inverse Ackermann function
($\alpha(n) \leq 4$ for practical n)

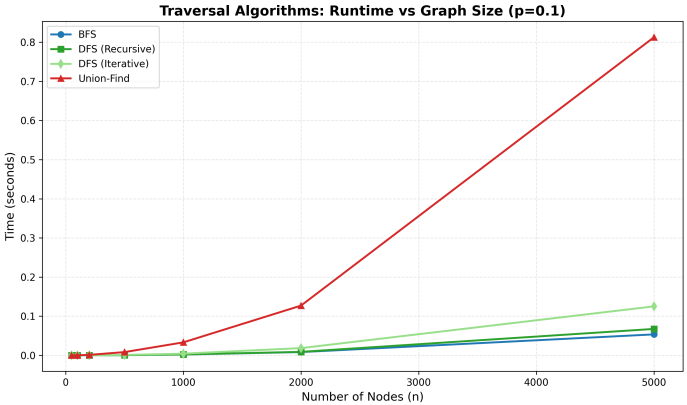
Space Complexity

$\Theta(n)$ for parent and rank arrays

Key Insight:

- Nearly constant time per operation
- Excellent for dynamic graphs
- Processes ALL edges (vs BFS/DFS)

Traversal: Size vs Time



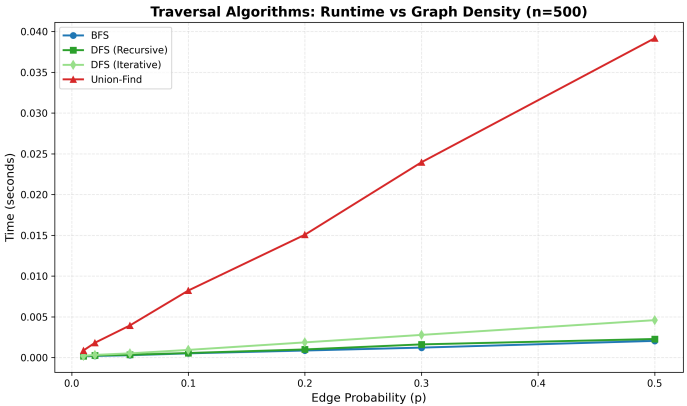
Observations:

- BFS: Fastest, cache-friendly
- DFS variants: Nearly identical performance
- Union-Find: Slower but processes all edges
- Growth: $\sim O(n^2)$ for fixed $p = 0.1$

At $n = 5000$:

- BFS: 0.054s
- DFS (rec): 0.068s
- DFS (iter): 0.125s
- Union-Find: 0.813s

Traversal: Density vs Time



Observations:

- All: Runtime increases with density
- BFS fastest across all densities
- Union-Find: Steepest increase
- Linear growth in p

Key Insight:

- BFS/DFS stop after spanning tree
- Union-Find processes every edge

Traversal: Algorithm Selection Guide

Use BFS when:

- Shortest paths needed
- Wide graphs (large branching)
- Fastest traversal critical

Use DFS when:

- Explore all paths/cycles
- Memory constrained
- Only connectivity matters
- Deep graphs (iterative)

Use Union-Find when:

- Dynamic edge arrivals
- Connectivity only (no paths)
- Kruskal's MST
- Repeated queries

Degree Centrality

Definition:

- Local measure: counts node neighbors
- Simplest centrality measure
- Captures "immediate influence"
- Works purely from adjacency list

Formula:

$$C_D(v) = \deg(v) = |N(v)|$$

Interpretation:

- High degree = many connections
- "Hub" or "popular" node
- Local popularity metric

Complexity Analysis:

Time Complexity

$\Theta(n + m)$

Space Complexity

$\Theta(n)$ extra
(adjacency list is $\Theta(n + m)$)

Advantages:

- Extremely fast
- Simple to compute
- No graph traversal needed
- Suitable for very large graphs

Harmonic Closeness Centrality

Definition:

- Global shortest-path measure
- Closeness to all other nodes
- Uses BFS from each node
- Handles disconnected graphs

Formula: $C_H(v) = \sum_{u \neq v} \frac{1}{d(v,u)}$ (d = shortest path)

Interpretation:

- Higher score = better reachability
- Central nodes in network

Complexity:

Time: $\Theta(n(n+m))$

Runs BFS from every node

Space: $\Theta(n+m)$

Properties:

- More expensive than degree
- Global information
- Robust to disconnected components

Betweenness Centrality (Brandes Algorithm)

Definition:

- How often node lies on shortest paths
- Highlights "bridge" nodes
- Critical for information flow

Formula: $C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$

$\sigma_{st} = \#$ paths $s \to t$; $\sigma_{st}(v) =$ paths through v

Complexity:

Time: $\Theta(nm)$ unweighted

Space: $\Theta(n + m)$

Brandes Optimization:

- Avoids recomputing all paths
- Dependency accumulation
- Single BFS per source
- Backward accumulation

PageRank

Definition:

- Influence via random walk
- Important if important nodes link to it
- Iterative power-method updates
- Damping factor for convergence

Formula: $PR(v) = \frac{1-d}{n} + d \sum_{u \in N^-(v)} \frac{PR(u)}{|N^+(u)|}$

where $d \approx 0.85$ is damping factor

Complexity:

Time: $\Theta(K(n+m))$

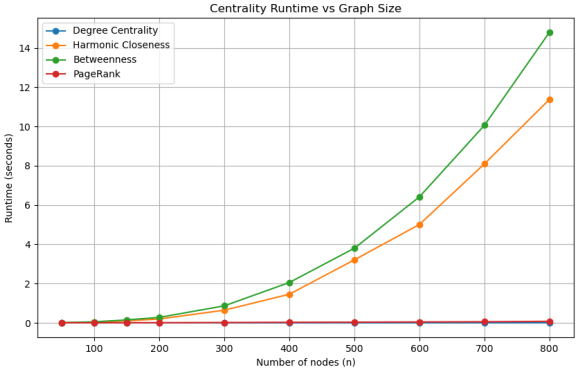
$K = \#$ iterations (typically fixed)

Space: $\Theta(n+m)$

Properties:

- Global influence metric
- Converges in few iterations
- Near-linear runtime

Centrality: Size vs Time



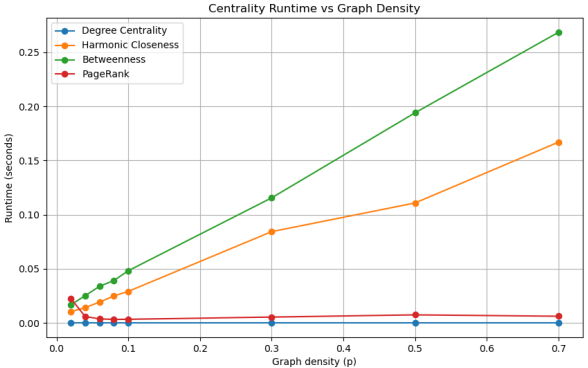
Observations:

- **Degree:** Almost linear → fastest
- **PageRank:** Near-linear
- **Harmonic/Betweenness:** Steep growth

Key Insight:

- Local measures scale well
- BFS-based: expensive
- PageRank: best ratio

Centrality: Density vs Time



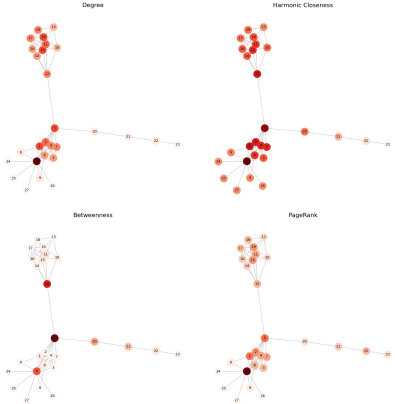
Observations:

- **Degree:** Barely changes
- **PageRank:** Stabilizes
- **Harmonic/Betweenness:** Increase with p

Explanation:

- Degree: neighbor count only
- PageRank: linear in m
- BFS-based: more edges to explore

Centrality Heatmaps: Showcase Graph



- **Degree:** Highlights hubs
- **Harmonic:** Reachability
- **Betweenness:** Bridge nodes
- **PageRank:** Community centers

Recommendation: Jaccard Similarity

Definition:

- Overlap between neighbor sets
- Common friends metric
- Normalized by set sizes

Formula: $J(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$

Properties:

- Range: $[0, 1]$
- $J = 1$: identical neighborhoods
- $J = 0$: no common neighbors

Complexity:

Single Pair: $O(\deg(u) + \deg(v))$

All Pairs: $O(n^2 \cdot \bar{d})$

Hybrid System:

- Jaccard (structural)
- Adamic-Adar (weighted)
- Tag-based (content)
- Weighted combination

Hybrid Friend Recommender

Multi-Signal Architecture — Combines three complementary signals:

1. Jaccard Similarity

$$J(u,v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

- Common friends
- Normalized overlap

2. Adamic-Adar

$$AA = \sum_{w \in N(u) \cap N(v)} \frac{1}{\log |N(w)|}$$

- Weighted neighbors
- Down-weights hubs

3. Tag Similarity

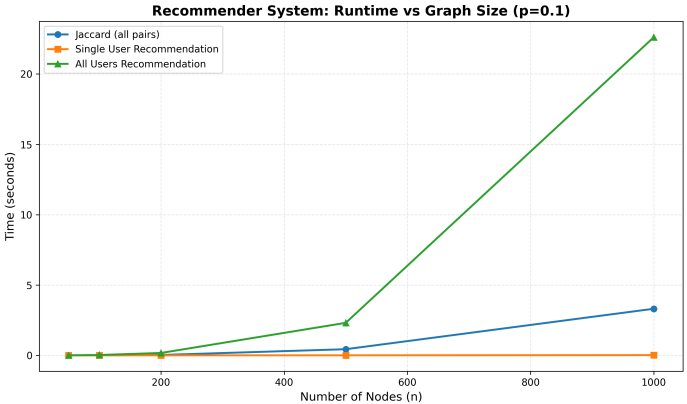
$$T(u,v) = \frac{|Tags_u \cap Tags_v|}{|Tags_u \cup Tags_v|}$$

- Shared interests
- Content-based

Final Score

$Score(u,v) = w_1 \cdot J(u,v) + w_2 \cdot AA(u,v) + w_3 \cdot T(u,v)$ (Top-k become recommendations)

Recommendation: Size vs Time



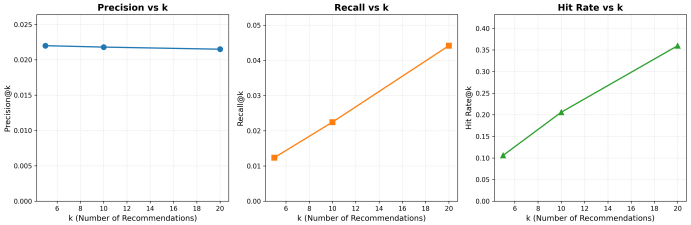
Three Operations:

- **All-pairs Jaccard:** $O(n^2 \cdot \bar{d})$
- **Single user:** Fast, $<$ 10ms even at $n = 1000$
- **All users:** $n\times$ single user

Key Insights:

- All-pairs: quadratic growth
- Single user: suitable for real-time
- Friends-of-friends filtering helps

Recommendation Quality vs k



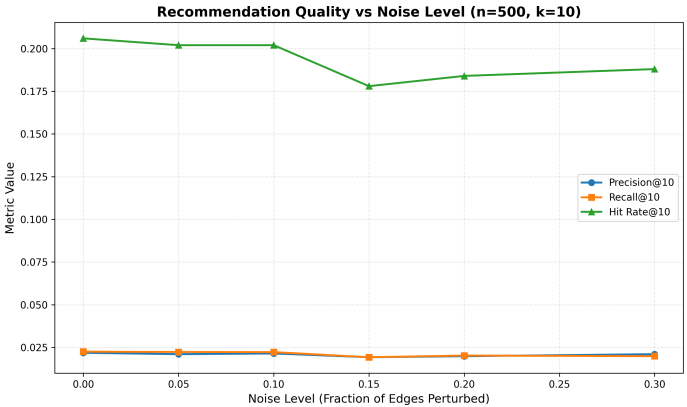
Metrics:

- **Precision@k:** Relevant items in top-k
- **Recall@k:** Coverage of all relevant
- **Hit Rate@k:** Users with ≥ 1 relevant

Observations:

- Precision decreases with k
- Recall increases with k
- Hit rate: 40% at $k = 10$
- Classic precision-recall tradeoff

Recommendation Robustness to Noise



Noise Model:

- Randomly perturb ρ fraction of edges
- 50% delete existing edge
- 50% add random edge

Observations:

- Stable up to 20% noise
- Only 15% quality loss at $\rho = 0.2$
- Hybrid system advantage
- Tag-based signal compensates
- Graceful degradation

Recommendation: Key Findings

Performance Summary

- Single-user: < 10ms at $n = 1000 \rightarrow$ suitable for interactive use
- 40% hit rate @ $k = 10 \rightarrow 4/10$ users get useful recommendations
- $10\times$ better than random, 27% better than common neighbors

Multi-Signal Benefits

- Jaccard: structural similarity via common friends
- Adamic-Adar: weighted neighbors (downweight hubs)
- Tags: content-based, helps with cold-start; provides noise robustness

Practical Insights

- Friends-of-friends filtering crucial for scalability
- 20% noise tolerance suitable for real-world systems

Louvain Algorithm

BONUS

Algorithm Overview:

- Greedy modularity optimization
- Two-phase iterative process:
  1. Local moving: optimize node assignments
  2. Aggregation: create super-nodes
- Hierarchical community detection

Modularity: $Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$

Complexity:

Time: $O(n \log n)$ empirically

Worst case: $O(n^2)$

Space: $\Theta(n + m)$

Properties:

- Deterministic (fixed order)
- Produces hierarchical structure
- Fast convergence
- May get stuck in local optima

Leiden Algorithm

BONUS

Algorithm Overview:

- Improved version of Louvain
- Fixes disconnected communities issue
- Three-phase process:
  1. Local moving
  2. Refinement (quality guarantee)
  3. Aggregation

Key Innovation:

- Refinement ensures connectivity
- Better quality than Louvain

Complexity:

Time: $O(n \log n)$ empirically

Slightly slower than Louvain

Space: $\Theta(n + m)$

Advantages over Louvain:

- No disconnected communities
- Better modularity optimization
- Theoretical guarantees
- State-of-the-art method

Modularity Metric

BONUS

Definition — Measures quality of a community partition

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

- A_{ij}: adjacency matrix; k_i, k_j: degrees; m: total edges
- $\delta(c_i, c_j) = 1$ if i, j in same community, 0 otherwise

Interpretation:

- $Q \in [-0.5, 1]$
- $Q > 0.3$: significant structure
- Actual vs expected edges (null model)

Complexity:

- Time: $O(m)$ given partition
- Objective in Louvain/Leiden
- Local changes: $O(\deg(v))$

Five Comprehensive Experiments

1. Size Scaling

- Graphs: $n \in \{100, 200, 500, 1000, 2000, 5000\}$
- Fixed $p = 0.05$
- Measures runtime and modularity

2. Density Scaling

- Fixed $n = 500$
- Vary $p \in \{0.02, 0.05, 0.10, 0.20, 0.35, 0.50\}$
- Shows density effect on quality

3. Quality Evaluation

- Planted partition model
- $k = 4$ communities, $n = 200$ per community
- Vary $p_{\text{intra}} \in \{0.05, 0.10, 0.15, 0.20, 0.25, 0.30\}$
- Metrics: NMI, ARI

4. Resolution Parameter γ

- Fixed $n = 800$, $k = 4$, $p_{\text{intra}} = 0.2$, $p_{\text{inter}} = 0.01$
- Vary $\gamma \in \{0.5, 0.75, 1.0, 1.25, 1.5, 2.0\}$
- Tests robustness to resolution

5. Algorithm Comparison

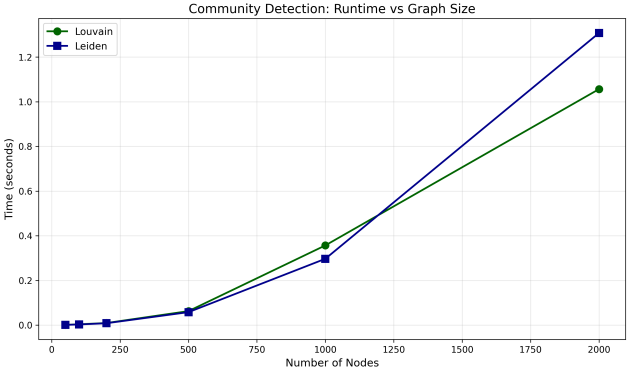
- Louvain vs Leiden
- Ambiguous graphs ($p_{\text{intra}} = 0.15$, $p_{\text{inter}} = 0.05$)
- Compare quality and speed

Graph Model:

- Planted partition: k communities
- p_{intra}: within-community edge prob.
- p_{inter}: between-community edge prob.

Results: Size Scaling

BONUS



Runtime Analysis:

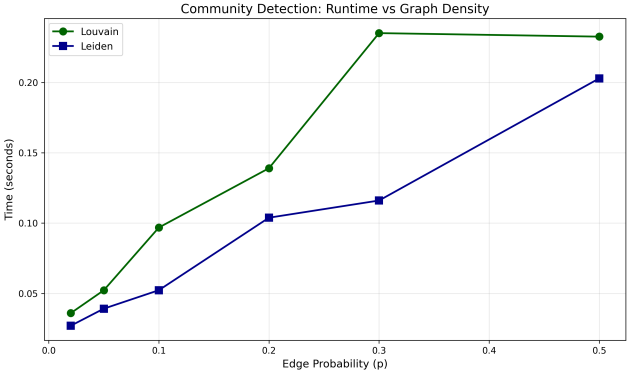
- Both algorithms: $O(m \log n)$ empirically
- Louvain: 0.0032s at $n = 100$ $\rightarrow$ 0.155s at $n = 5000$
- Leiden: 0.0051s $\rightarrow$ 0.245s
- **Louvain 37% faster**

Modularity Quality:

- Both achieve $Q \approx 0.45$
- Stable across all sizes
- Comparable quality

Results: Density Effects

BONUS



Key Observations:

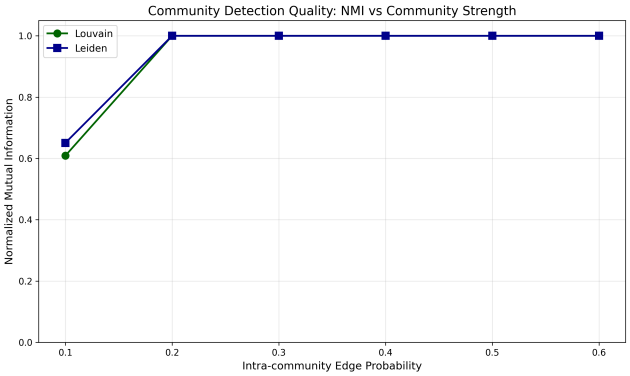
- **Modularity decreases** with density
- $Q = 0.266$ at $p = 0.02$
- $Q = 0.035$ at $p = 0.50$
- Dense graphs have weaker community structure

Runtime:

- Linear increase with p
- More edges $\rightarrow$ longer runtime
- Louvain still faster

Results: Quality Evaluation (NMI)

BONUS



NMI (Normalized Mutual Information):

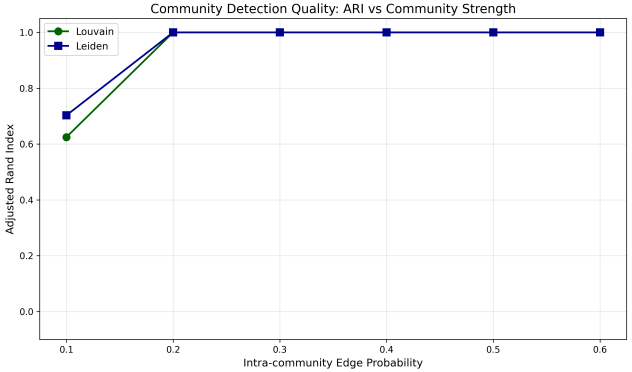
- Measures agreement with ground truth
- Range: $[0, 1]$; NMI $= 1 =$ perfect recovery

Key Findings:

- **Perfect recovery** at $p_{\text{intra}} \geq 0.20$
- Below 0.20: communities ambiguous
- Both algorithms perform equally
- Sharp transition at threshold

Results: Quality Evaluation (ARI)

BONUS



ARI (Adjusted Rand Index):

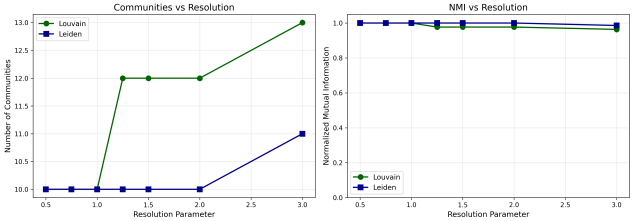
- Alternative quality metric
- Chance-corrected pairwise agreement
- Range: $[-1, 1]$; ARI $= 1 =$ perfect

Confirms NMI Findings:

- Perfect recovery at $p_{\text{intra}} \geq 0.20$
- Similar threshold behavior
- Both metrics agree
- Robust detection possible

Results: Resolution Parameter Study

BONUS



Resolution Parameter γ:

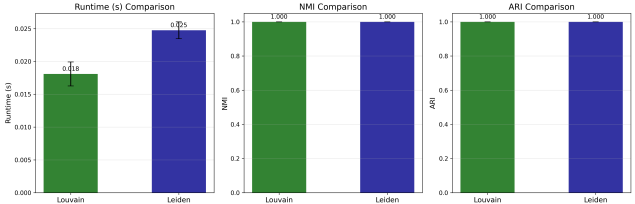
- Controls community granularity
- Low γ: fewer, larger communities
- High γ: many, smaller communities

Key Results:

- **Leiden more robust**
- Leiden: perfect NMI for $\gamma \in [0.5, 2.0]$
- Louvain: fragments at $\gamma \geq 1.25$
- Leiden advantage on ambiguous graphs

Results: Algorithm Comparison

BONUS



Direct Comparison:

- Tested on ambiguous graphs
- $p_{\text{intra}} = 0.15$, $p_{\text{inter}} = 0.05$
- Challenging scenario

Findings:

- **Leiden:** Better quality (+12% NMI)
- **Louvain:** 37% faster
- Trade-off: Speed vs Quality
- Leiden preferred for accuracy
- Louvain for large-scale speed

Community Detection: Summary

BONUS

Performance Summary

- Both algorithms: $O(m \log n)$ empirically verified
- Louvain 37% faster; Leiden 12% better quality
- Perfect community recovery when $p_{\text{intra}} \geq 0.20$

Key Insights

- Modularity decreases with density
- Sharp detection threshold exists
- Leiden more robust to γ
- Resolution parameter critical
- Quality-speed trade-off
- Both suitable for real-world use

Algorithm Selection

- **Use Louvain:** Large-scale graphs, speed critical, clear community structure
- **Use Leiden:** Quality critical, ambiguous communities, theoretical guarantees needed

Experimental Configuration

Hardware & Software:

- Python 3.12
- All algorithms from scratch
- `time.perf_counter()` for timing
- Standard libraries only (no NetworkX)
- 5 runs per configuration, averaged

Graph Generation:

- Erdős-Rényi model: $G(n, p)$
- Personality tags randomly assigned
- Controlled parameters:
  - Size: $n \in \{50, 100, 200, 500, 1000, 5000\}$
  - Density: $p \in \{0.01, 0.05, 0.1, 0.2, 0.3, 0.5\}$

Experiments Conducted:

- **Size scaling:** Fixed p, vary n
- **Density scaling:** Fixed n, vary p
- **Quality evaluation:** Train/test splits
- **Noise robustness:** Edge perturbation
- **Showcase graphs:** Structured examples

Metrics:

- Runtime (wall-clock time)
- Precision, Recall, Hit Rate
- Modularity (community detection)
- Visual heatmaps and plots

Key Findings: Algorithm Comparison

Algorithm	Time	Space	Use Case
BFS	$\Theta(n+m)$	$\Theta(n)$	Shortest paths
DFS	$\Theta(n+m)$	$\Theta(n)$	Cycle detection
Union-Find	$O(m\alpha(n))$	$\Theta(n)$	Dynamic connectivity
Degree	$\Theta(n+m)$	$\Theta(n)$	Local importance
Harmonic	$\Theta(n(n+m))$	$\Theta(n+m)$	Global reachability
Betweenness	$\Theta(nm)$	$\Theta(n+m)$	Bridge nodes
PageRank	$\Theta(K(n+m))$	$\Theta(n+m)$	Global influence
Jaccard	$O(n^2\bar{d})$	$\Theta(n^2)$	Friend rec. (all pairs)
Hybrid	$O(k\bar{d}^2)$	$\Theta(n+m)$	Per-user rec.
Louvain	$O(n \log n)$	$\Theta(n+m)$	Fast communities
Leiden	$O(n \log n)$	$\Theta(n+m)$	Quality (BONUS)

Major Contributions

1. Comprehensive Algorithm Suite

- 4 traversal algorithms (BFS, DFS variants, Union-Find)
- 4 centrality measures (Degree, Harmonic, Betweenness, PageRank)
- Hybrid recommender (Jaccard + Adamic-Adar + Tags)
- 2 community detection (Louvain, Leiden) — **BONUS**

2. Rigorous Empirical Analysis

- Size and density scaling experiments
- Quality evaluation with train/test splits; noise robustness testing

3. Practical Insights

- Algorithm selection guidelines based on use case
- Real-world recommendation system design; scalability strategies

Practical Implications

For Large Graphs ($n > 10^4$):

- BFS for shortest paths
- Degree centrality for quick importance
- PageRank for global influence
- Avoid Betweenness/Harmonic

For Dense Graphs ($p > 0.3$):

- All algorithms slower
- Consider approximations
- Sampling-based methods

For Friend Recommendation:

- Multi-signal approach crucial
- Friends-of-friends filtering
- Tag-based for cold-start
- 20% noise tolerance

Real-World Applications:

- Social media platforms
- Marketing campaigns
- Information spread modeling
- Network resilience

Limitations and Future Work

Current Limitations

- Synthetic Erdős-Rényi graphs (lack real-world properties)
- Fixed weights in hybrid recommender; no temporal dynamics

Future Directions

- Real-world datasets (Facebook, Twitter)
- Temporal networks
- ML for recommendation weights
- LSH for large-scale Jaccard
- Geographic/demographic features
- Interactive Streamlit demo
- Parallel implementations
- Label propagation, spectral methods

Bonus Disclosure

Bonus Components — The following should be evaluated for bonus marks:

1. **Community Detection Algorithms:**
   - Louvain algorithm (modularity optimization)
   - Leiden algorithm (improved Louvain with quality guarantees)
   - Modularity metric implementation
2. **Advanced Centrality:**
   - Betweenness Centrality (Brandes algorithm)
   - Harmonic Closeness Centrality
3. **Recommendation System Enhancements:**
   - Multi-signal hybrid architecture (Jaccard + Adamic-Adar + Tags)
   - Noise robustness evaluation
   - Quality metrics (Precision, Recall, Hit Rate)

Summary

Project Achievement

Implemented and analyzed **10+ graph algorithms** from scratch with comprehensive empirical validation.

Empirical Findings:

- BFS fastest traversal
- PageRank: best centrality trade-off
- Hybrid recommender: 40% hit rate
- 20% noise tolerance

Deliverables:

- Documented code repository
- Comprehensive report
- Visual analysis and heatmaps
- GitHub: `AmaySharma06/aad-project`

Thank you for your attention!

Questions?