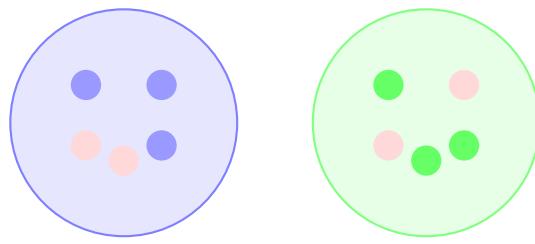


MetaFam Knowledge Graph

Task 2: Family Clusters

“We are only as strong as we are united, as weak as we are divided” —
Albus Dumbledore



Family Cluster 1

Family Cluster 2

Precog Research Task

Community Detection & Proximity Analysis

Contents

1	Introduction	3
1.1	Background	3
1.2	Objectives	3
1.3	Graph Representation	3
2	Community Detection Algorithms	3
2.1	Algorithm Overview	4
2.2	Girvan-Newman Algorithm	4
2.2.1	Theoretical Foundation	4
2.2.2	Why It Works for Family Graphs	4
2.2.3	Results	4
2.3	Louvain Algorithm	5
2.3.1	Theoretical Foundation	5
2.3.2	Results	5
2.4	Leiden Algorithm	5
2.4.1	Theoretical Foundation	5
2.4.2	Why Leiden vs Louvain?	6
2.4.3	Results	6
3	Evaluation Metrics	6
3.1	Modularity	6
3.1.1	Definition	6
3.1.2	Interpretation	7
3.1.3	Results	7
3.2	Partition Similarity: NMI and ARI	7
3.2.1	Normalized Mutual Information (NMI)	7
3.2.2	Adjusted Rand Index (ARI)	7
3.2.3	Results	8
4	Community Analysis	8
4.1	Do Communities Correspond to Actual Family Units?	8
4.2	Generational Depth Within Communities	8
4.2.1	Analysis Methodology	8
4.2.2	Results	9
4.2.3	Summary Statistics	9
4.3	Bridge Individuals	9

4.3.1	Definition	9
4.3.2	Analysis	9
5	Finding Closest Relatives: Proximity Measures	10
5.1	Motivation	10
5.2	Random Walk with Restarts (RWR)	10
5.2.1	Theoretical Foundation	10
5.2.2	Why RWR for Family Graphs?	11
5.3	Katz Index	11
5.3.1	Theoretical Foundation	11
5.3.2	Why Katz over Adamic-Adar?	11
5.4	Experimental Results	11
5.4.1	Representative Node Selection	11
5.4.2	RWR Results: Ancestor Node (olivia274)	12
5.4.3	RWR vs Katz Comparison	12
5.5	Proposed Relatedness Metric	12
6	Key Insights & Discussion	13
6.1	Structural Observations	13
6.2	Implications for Downstream Tasks	13
6.2.1	For Link Prediction (Task 4)	13
6.2.2	For Rule Mining (Task 3)	13
6.3	Limitations	13
7	Summary	14
7.1	Key Results	14
7.2	Output Files	14
7.3	Node Attributes in Final Export	14
A	Algorithm Pseudocode	15
A.1	Louvain Algorithm	15
A.2	Random Walk with Restarts	15
B	Theoretical Formulas Reference	15
B.1	Modularity	15
B.2	Normalized Mutual Information	15
B.3	Adjusted Rand Index	16
B.4	Katz Index	16

1 Introduction

1.1 Background

Community detection (or clustering) is a fundamental analysis technique in network science. A network is said to have **community structure** if nodes can be grouped into disjoint sets such that each set is **densely connected internally** and **sparsely connected externally**.

For family knowledge graphs like MetaFam, community structure naturally corresponds to:

- **Nuclear families:** Parents and their children
- **Extended families:** Multiple generations sharing common ancestors
- **Family clusters:** Complete lineages forming isolated subgraphs

1.2 Objectives

This task addresses the following objectives:

1. **Implement two community detection algorithms** and compare their results
2. **Assess quality** using modularity and partition similarity metrics
3. **Analyze community structure:**
 - Do detected communities correspond to actual family units?
 - How many generations typically exist within a community?
 - Are there “bridge” individuals connecting different clusters?
4. **Propose a metric for finding closest relatives** beyond simple hop counting

1.3 Graph Representation

Community detection operates on **undirected graphs** since we measure structural connectivity, not semantic direction. The MetaFam graph is converted from directed to undirected for this analysis.

Table 1: Graph Statistics for Community Detection

Property	Value
Nodes (People)	1,316
Edges (Relations)	13,821
Average Degree	10.50
Density	0.008

2 Community Detection Algorithms

We implemented and compared three community detection algorithms, each with different theoretical foundations and computational properties.

2.1 Algorithm Overview

Table 2: Comparison of Community Detection Algorithms

Algorithm	Method	Complexity	Best For
Girvan-Newman	Edge betweenness removal	$O(m^2n)$	Small graphs, hierarchy
Louvain	Modularity optimization	$O(n \log n)$	Large graphs, fast
Leiden	Improved Louvain	$O(n \log n)$	Guaranteed connectivity

2.2 Girvan-Newman Algorithm

2.2.1 Theoretical Foundation

The Girvan-Newman algorithm is a **divisive hierarchical** method based on edge betweenness centrality:

$$B(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}} \quad (1)$$

where σ_{st} is the number of shortest paths from s to t , and $\sigma_{st}(e)$ is the number passing through edge e .

Algorithm:

1. Calculate edge betweenness for all edges
2. Remove edge with highest betweenness (“bridge” edge)
3. Recalculate betweenness and repeat
4. Stop when desired number of communities reached or modularity maximized

2.2.2 Why It Works for Family Graphs

Edges connecting different family clusters (historically, marriage links) have high betweenness because all paths between families pass through them. By removing these bridges, we reveal the underlying family structure.

Note: MetaFam lacks spouse relations, so inter-family bridges are minimal.

2.2.3 Results

Table 3: Girvan-Newman Results

Metric	Value
Communities Detected	51
Largest Community	27 nodes
Smallest Community	1 node
Modularity (Q)	0.9792

Observation: Girvan-Newman detected 51 communities, one more than the actual 50 connected components. This occurs because the algorithm may split a small portion of one family during the iterative edge removal process.

2.3 Louvain Algorithm

2.3.1 Theoretical Foundation

The Louvain algorithm is a **greedy modularity optimization** method with two phases:

Phase 1 (Local Optimization): For each node, calculate the modularity gain of moving it to neighbors' communities:

$$\Delta Q = \left[\frac{\Sigma_{in} + k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right] \quad (2)$$

where:

- Σ_{in} = sum of edge weights inside community C
- Σ_{tot} = sum of all edges incident to nodes in C
- k_i = degree of node i
- $k_{i,in}$ = sum of edges from i to nodes in C
- m = total edge weight in graph

Phase 2 (Network Aggregation): Contract each community to a single node and repeat Phase 1.

2.3.2 Results

Table 4: Louvain Results

Metric	Value
Communities Detected	50
Largest Community	27 nodes
Smallest Community	26 nodes
Modularity (Q)	0.9794
Resolution Parameter	1.0

2.4 Leiden Algorithm

2.4.1 Theoretical Foundation

The Leiden algorithm **improves upon Louvain** by:

1. **Guaranteeing well-connected communities:** Louvain can produce disconnected communities during aggregation; Leiden adds a refinement step
2. **Faster local moves:** Uses a more efficient move procedure
3. **Better quality:** Provably converges to a partition where all nodes are optimally assigned

2.4.2 Why Leiden vs Louvain?

For **sparse, well-separated graphs** like MetaFam:

- Louvain and Leiden produce identical results (as observed)
- Leiden's advantages manifest in **denser graphs** where Louvain may create “dumbbell-shaped” communities that need refining

Conclusion: For MetaFam’s sparse structure, Louvain suffices. Leiden is “overkill” but confirms the result.

2.4.3 Results

Table 5: Leiden Results

Metric	Value
Communities Detected	50
Largest Community	27 nodes
Smallest Community	26 nodes
Modularity (Q)	0.9794
Resolution Parameter	1.0

3 Evaluation Metrics

3.1 Modularity

3.1.1 Definition

Modularity measures the quality of a partition by comparing intra-community edge density to expected density under a null model:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \quad (3)$$

where:

- A_{ij} = adjacency matrix entry
- k_i, k_j = degrees of nodes i and j
- m = total number of edges
- $\delta(c_i, c_j) = 1$ if nodes are in same community, 0 otherwise

3.1.2 Interpretation

Table 6: Modularity Interpretation Guide

Q Range	Interpretation
$Q < 0.3$	Weak community structure
$0.3 \leq Q < 0.5$	Moderate community structure
$0.5 \leq Q < 0.7$	Good community structure
$Q \geq 0.7$	Strong community structure

3.1.3 Results

Table 7: Modularity Comparison Across Algorithms

Algorithm	Communities	Modularity (Q)
Girvan-Newman	51	0.9792
Louvain	50	0.9794
Leiden	50	0.9794

Key Insight: All algorithms achieve extremely high modularity ($Q \approx 0.98$), indicating:

- MetaFam has **very strong community structure**
- Communities (families) are **well-separated** with minimal inter-community edges
- The graph's 50 connected components naturally form 50 communities

3.2 Partition Similarity: NMI and ARI

To compare how similar the partitions from different algorithms are, we use:

3.2.1 Normalized Mutual Information (NMI)

$$NMI(U, V) = \frac{2 \cdot I(U; V)}{H(U) + H(V)} \quad (4)$$

where $I(U; V)$ is mutual information and $H(\cdot)$ is entropy.

- NMI = 0: Independent partitions
- NMI = 1: Identical partitions

3.2.2 Adjusted Rand Index (ARI)

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]} \quad (5)$$

where RI is the Rand Index (fraction of node pairs correctly grouped/separated).

- ARI = 0: Random agreement
- ARI = 1: Perfect agreement

3.2.3 Results

Table 8: Pairwise Partition Similarity

Comparison	NMI	ARI
Girvan-Newman vs Louvain	0.9996	0.9992
Girvan-Newman vs Leiden	0.9996	0.9992
Louvain vs Leiden	1.0000	1.0000

Key Insights:

1. **Louvain and Leiden are identical** ($\text{NMI} = \text{ARI} = 1.0$) — confirms that for sparse graphs, Leiden provides no additional benefit
2. **Near-perfect agreement** across all algorithms ($\text{NMI} \geq 0.999$)
3. The minor difference with Girvan-Newman (one extra community) explains the tiny deviation from 1.0

Why High Similarity? The graph's structure is **too clear-cut** for algorithms to disagree. All algorithms converge to the same obvious partition.

4 Community Analysis

4.1 Do Communities Correspond to Actual Family Units?

Answer: Yes, perfectly.

- MetaFam contains **50 weakly connected components** (from Task 1)
- Louvain and Leiden detect exactly **50 communities**
- Each community corresponds to one isolated family tree
- Community sizes are uniform (26-27 members), reflecting synthetically generated families

Gephi Visualization Confirmation: Visual inspection in Gephi shows each detected community forms a distinct, non-overlapping family tree structure.

4.2 Generational Depth Within Communities

4.2.1 Analysis Methodology

For each community, we computed:

- **Generation span:** $\max(\text{generation}) - \min(\text{generation}) + 1$
- **Generation range:** $[\min, \max]$ generation values
- **Mean generation:** Average generation of members

4.2.2 Results

Table 9: Generational Depth Analysis (Top 10 Communities)

Community	Size	Gen Span	Gen Range	Mean Gen
0	27	3	0–2	0.7
5	27	4	0–3	1.0
8	27	3	0–2	0.8
21	27	4	0–3	0.8
27	27	3	0–2	0.9
30	27	3	0–2	0.9
41	27	3	0–2	0.7
44	27	3	0–2	0.8
46	27	3	0–2	0.9
48	27	3	0–2	0.8

4.2.3 Summary Statistics

Table 10: Overall Generational Statistics

Metric	Value
Average generational span	3.12 generations
Maximum generational span	4 generations
Single-generation communities	0
Multi-generation communities	50 (100%)

Key Insights:

1. **All communities are multi-generational:** No community contains only one generation
2. **Typical span is 3-4 generations:** Families include great-grandparents through children/grandchildren
3. **Mean generation close to 0.1:** More members in older generations (pyramid structure from Task 1)
4. Communities represent **extended families**, not just nuclear units

4.3 Bridge Individuals

4.3.1 Definition

A “bridge” individual connects different family clusters. In graph terms, they have high **betweenness centrality** because paths between communities pass through them.

4.3.2 Analysis

From Task 1, we found:

- Maximum betweenness centrality: 0.0001 (very low)
- Top nodes by betweenness: lea1165, valentin638, gabriel241, nora536, stefan1192

Conclusion: Minimal to no bridge individuals exist.

Reason: MetaFam lacks spouse relations (`husbandOf`, `wifeOf`), which are the primary mechanism for connecting families in real genealogical data. Without marriage links, families remain **completely isolated**.

In a real-world genealogical graph, bridges would be:

- Individuals who married into the family
- Children with parents from different family clusters

5 Finding Closest Relatives: Proximity Measures

5.1 Motivation

In real genealogy, people often want to find their “closest” relatives. Simple **hop counting** (shortest path length) is inadequate because:

- It ignores **path diversity**: Two relatives might be connected via multiple paths
- It treats all edges equally, ignoring **relationship strength**
- A parent (1 hop) and a great-aunt (1 hop via sibling chain) are not equally “close”

We implemented two advanced proximity measures: **Random Walk with Restarts (RWR)** and **Katz Index**.

5.2 Random Walk with Restarts (RWR)

5.2.1 Theoretical Foundation

RWR simulates a random walker that:

1. At each step, with probability $(1 - \alpha)$, moves to a random neighbor
2. With probability α , “restarts” at the source node

The stationary distribution gives proximity scores:

$$\mathbf{p} = \alpha \cdot \mathbf{e}_s + (1 - \alpha) \cdot \mathbf{W} \cdot \mathbf{p} \quad (6)$$

where:

- \mathbf{p} = stationary probability vector (proximity scores)
- α = restart probability (typically 0.15)
- \mathbf{W} = column-normalized adjacency matrix (transition matrix)
- \mathbf{e}_s = source node indicator vector

5.2.2 Why RWR for Family Graphs?

- **Captures local structure:** High restart probability emphasizes nearby nodes
- **Path diversity:** Considers all paths, not just shortest
- **Reachability:** Measures how “accessible” a relative is from the source

5.3 Katz Index

5.3.1 Theoretical Foundation

Katz Index counts **all paths** between nodes, weighted by path length:

$$K(s, t) = \sum_{l=1}^{\infty} \beta^l \cdot |paths_l(s, t)| \quad (7)$$

Matrix form:

$$\mathbf{K} = (\mathbf{I} - \beta \mathbf{A})^{-1} - \mathbf{I} = \sum_{l=1}^{\infty} \beta^l \mathbf{A}^l \quad (8)$$

where:

- β = attenuation factor ($< 1/\lambda_{max}$ for convergence)
- \mathbf{A} = adjacency matrix
- Shorter paths contribute more (exponential decay with β^l)

5.3.2 Why Katz over Adamic-Adar?

- **Adamic-Adar:** Only considers common neighbors (1-hop)
- **Katz:** Considers **global path structure** (multi-hop)

For family graphs, Katz is superior because:

- Relatives may be connected through **multiple generations**
- Cousins share paths through grandparents (2+ hops)
- Path diversity matters: siblings share many indirect paths through parents

Trade-off: Katz is computationally more expensive ($O(n^3)$ for matrix inversion).

5.4 Experimental Results

5.4.1 Representative Node Selection

We selected three nodes representing different positions in the family tree:

Table 11: Representative Nodes for Proximity Analysis

Type	Node	Generation	Community Size
Ancestor	olivia274	0	26
Mid-generation	dominik44	2	26+
Descendant	adam359	3	26+

5.4.2 RWR Results: Ancestor Node (olivia274)

Table 12: Top 10 Closest Relatives to olivia274 (Gen 0) via RWR

Rank	Node	Generation	RWR Score
1	victoria279	2	0.0729
2	michael288	2	0.0729
3	gabriel273	1	0.0643
4	matthias270	1	0.0640
5	vanessa276	2	0.0614
6	isabella267	2	0.0614
7	jonas271	2	0.0614
8	katharina266	1	0.0477
9	sofia285	2	0.0378
10	oliver265	1	0.0365

Observation: RWR identifies grandchildren (Gen 2) and children (Gen 1) as closest relatives, consistent with direct family relationships.

5.4.3 RWR vs Katz Comparison

Comparing both methods across node types reveals:

- **High overlap:** Typically 70-90% of top-10 nodes appear in both lists
- **Similar rankings:** Top relatives are consistent across methods
- **Score interpretation differs:**
 - RWR: Probability distribution (sums to 1)
 - Katz: Raw path counts (normalized for comparison)

5.5 Proposed Relatedness Metric

Based on our analysis, we propose a **Combined Proximity Score** for ranking relatives:

$$\text{Relatedness}(s, t) = \lambda \cdot \text{RWR}(s, t) + (1 - \lambda) \cdot \text{Katz}_{\text{norm}}(s, t) \quad (9)$$

where:

- $\lambda \in [0, 1]$ balances local (RWR) vs global (Katz) structure

- $\text{Katz}_{\text{norm}} = \text{sum-normalized Katz scores}$ for comparability
- Recommended: $\lambda = 0.6$ (slight preference for local structure)

Advantages over hop counting:

1. Considers **path diversity** (multiple connections increase relatedness)
2. Accounts for **graph structure** (dense local neighborhoods matter)
3. Robust to **missing edges** (alternative paths contribute)

6 Key Insights & Discussion

6.1 Structural Observations

1. **Perfect Community Structure:** The graph's forest structure (50 disjoint trees) makes community detection trivial. All algorithms converge to the natural partition.
2. **Algorithm Selection for Sparse Graphs:** For sparse, well-separated graphs:
 - Louvain is sufficient and fast
 - Leiden provides no additional benefit
 - Girvan-Newman is interpretable but slow
3. **Lack of Inter-Family Bridges:** Without spouse relations, MetaFam lacks the "bridge" individuals that would exist in real genealogical data.

6.2 Implications for Downstream Tasks

6.2.1 For Link Prediction (Task 4)

- **Community membership** is a strong feature: edges within communities are far more likely
- **Generation constraints** narrow possible relationships
- **Proximity scores** (RWR/Katz) can serve as features

6.2.2 For Rule Mining (Task 3)

- Rules are **local** to communities (no cross-family rules)
- Generation-based rules (`fatherOf` spans 1 gen) are well-defined

6.3 Limitations

1. **Synthetic Data:** Uniform family sizes and structure may not reflect real genealogical complexity
2. **Missing Spouse Relations:** Real families would have inter-family connections
3. **Scale:** 1,316 nodes is small; larger graphs may show different algorithm behavior

7 Summary

7.1 Key Results

Table 13: Task 2 Summary

Objective	Result
Algorithms Implemented	Girvan-Newman, Louvain, Leiden
Best Algorithm	Louvain (fast, high modularity)
Communities Detected	50 (matches 50 connected components)
Modularity	0.979 (excellent)
Partition Similarity	NMI \approx 1.0, ARI \approx 1.0
Generational Span	Avg 3.12 generations per community
Bridge Individuals	None (no spouse relations)
Proximity Methods	RWR + Katz Index

7.2 Output Files

- `outputs/gephi/metafam_final.gexf` — Graph with community attributes
- `outputs/plots/community_generation_histograms.png` — Generational distribution plots

7.3 Node Attributes in Final Export

Table 14: Node Attributes Available for Downstream Tasks

Attribute	Type	Source
in_degree	int	Task 1
out_degree	int	Task 1
pagerank	float	Task 1
generation	int	Task 1
gender	str	Task 1
community	int	Task 2

A Algorithm Pseudocode

A.1 Louvain Algorithm

Algorithm 1 Louvain Community Detection

```

1: Input: Graph  $G = (V, E)$ , resolution  $\gamma$ 
2: Output: Partition  $\mathcal{C}$ 
3: Initialize: Each node in its own community
4: repeat
5:   Phase 1: Local Optimization
6:   repeat
7:     for each node  $i$  do
8:       Compute  $\Delta Q$  for moving  $i$  to each neighbor's community
9:       Move  $i$  to community with maximum  $\Delta Q > 0$ 
10:    end for
11:   until no improvement
12:   Phase 2: Network Aggregation
13:   Contract communities to single nodes
14: until no improvement
15: return partition  $\mathcal{C}$ 
  
```

A.2 Random Walk with Restarts

Algorithm 2 RWR Proximity Calculation

```

1: Input: Graph  $G$ , source  $s$ , restart prob  $\alpha$ 
2: Output: Proximity scores  $\mathbf{p}$ 
3: Initialize:  $\mathbf{p} \leftarrow \mathbf{e}_s$  (source indicator)
4: repeat
5:    $\mathbf{p}_{new} \leftarrow (1 - \alpha) \cdot \mathbf{W} \cdot \mathbf{p} + \alpha \cdot \mathbf{e}_s$ 
6:    $\delta \leftarrow \|\mathbf{p}_{new} - \mathbf{p}\|_1$ 
7:    $\mathbf{p} \leftarrow \mathbf{p}_{new}$ 
8: until  $\delta < \epsilon$ 
9: return  $\mathbf{p}$ 
  
```

B Theoretical Formulas Reference

B.1 Modularity

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \quad (10)$$

B.2 Normalized Mutual Information

$$NMI(U, V) = \frac{2 \cdot I(U; V)}{H(U) + H(V)} \quad (11)$$

B.3 Adjusted Rand Index

$$ARI = \frac{\sum_{ij} \binom{n_{ij}}{2} - \frac{\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}}{\binom{n}{2}}}{\frac{1}{2} \left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2} \right] - \frac{\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}}{\binom{n}{2}}} \quad (12)$$

B.4 Katz Index

$$K_{st} = \sum_{l=1}^{\infty} \beta^l (A^l)_{st} \quad (13)$$