Software Engineering Department
Braude College

Capstone Project Phase B – 61999

Mapping Changes in Large Networks

25-1-R-13

Students:

Shimron Ifrah shimron202@gmail.com

Yossi Shem-Tov Yossishemtov7@gmail.com

Supervisors:

Dr. Renata Avros

Prof. Zeev Volkovich

Git

https://github.com/yossishemtov/Mapping-Change-in-Large-Networks-final-project.git

Table of contents

Abstract

This project proposes a new framework for analyzing and mapping structural changes in large-scale dynamic networks, using journal citation networks as a case study. It focuses on identifying patterns, trends, and transformations over time, addressing the limitations of traditional statistical models in distinguishing meaningful trends from noise in complex networks.

The approach combines advanced algorithms GraphSAGE++, FastGCN, and HDBSCAN to process large-scale data efficiently. It generates dynamic node embeddings, clusters nodes with similar characteristics, and detects significant structural changes. Citation networks are analyzed across sequential periods, with HDBSCAN clustering identifying dense regions and labeling outliers as noise. Key transitions, such as cluster splits, mergers, and emerging groups, are visualized using alluvial diagrams to reveal trends and shifts in research focus.

By integrating these methods, the framework overcomes the challenges of scalability, noise filtering, and trend detection, providing insights into structural transformations applicable to fields like transportation, biology, and finance.

1. Introduction

Analyzing large networks is essential for understanding complex systems. Various tools are designed to highlight prominent features and simplify these structures. These tools effectively reveal the static organization of networks but often fall short in tracking how these structures change over time. Understanding such changes is critical in systems like global air traffic, social contact networks during disease outbreaks, financial markets, or gene regulatory networks in health and disease [9].

Mapping network changes requires distinguishing meaningful trends from random noise, which traditional statistical models often fail to do. Clustering techniques provide a more effective alternative, allowing the analysis of systems with diverse and interacting elements.

Real-world networks, such as global air traffic or citation networks, are singular and cannot be replicated. This uniqueness makes common sampling approaches ineffective. Instead, advanced techniques like parametric bootstrap resampling are used to assess the significance of network clusters. These methods preserve the network's integrity by resampling link weights while maintaining the unique characteristics of nodes.

To visualize structural and functional changes, alluvial diagrams provide an intuitive way to track and summarize network evolution. This approach has broad applications, such as studying changes in transportation, finance, and scientific disciplines, enabling a deeper understanding of the dynamics driving network transformations over time.

2. Related Work and Problem Definition

2.1 Problem Definition

Large networks are widely used to represent complex systems across various domains, such as social interactions, scientific collaborations, or biological processes. These networks consist of nodes (vertices) and edges, capturing relationships and interactions within the system. As these networks evolve, understanding and mapping their structural changes is critical to uncovering patterns, trends, and transformations.

The problem focuses on identifying and analyzing changes within large dynamic networks. The input consists of a sequence of large networks, each representing interactions or relationships over specific periods. These networks serve as snapshots of the system at various times, enabling an analysis of how they evolve.

The primary purpose is to cluster these networks using advanced methods designed for large-scale data. This clustering process identifies meaningful structural components within the networks, setting the foundation for tracking changes over time. By applying appropriate methodologies, the objective is to detect and visualize the trend of changes, highlighting how clusters or network structures transform.

However, this task presents significant challenges:

1. **Scalability**: Large networks often contain millions of nodes and edges, making computations intensive and requiring efficient algorithms.

2. **Noise Filtering**: Real-world networks include random movements and noise that can Hide meaningful trends. Distinguishing between actual structural changes and random shifts is a critical challenge.

3. **Detecting Significant Trends**: Identifying patterns and transformations that are statistically and structurally significant is complex, especially when dealing with large datasets over multiple periods.

This study focuses on replacing the existing solution algorithm with a new, more advanced approach that leverages modern technology. The goal is to enhance insights into the dynamic nature of large-scale networks and better track their evolution over time.

2.2 Related works

Mapping changes in large networks is critical to understanding the evolution of complex systems, and several methods are proposed to address this challenge. Rosvall and Bergstrom (2010) introduced a method using bootstrap resampling and significance clustering to distinguish between meaningful structural changes and noise in citation networks, visualizing these changes with alluvial diagrams [9]. Girvan and Newman (2002) proposed a modularity-based method to detect network community structures, showing that removing edges with high betweenness centrality reveals changing cluster structures over time [4]. Palla et al. (2005) presented the Clique Percolation Method (CPM) to detect overlapping communities and track the persistence of cliques across time intervals [8]. Giumarra and Amaral (2005) developed a method to identify functional modules in biological networks and studied how these modules change roles over time to adapt to different conditions [5]. Blondel et al. (2008) proposed the Louvain method for fast community detection and explored how community structures change by iteratively optimizing modularity in evolving networks [2]. Annaud and Guillaume (2010) explored methods for detecting communities in dynamic networks by comparing snapshots over time and introduced temporal smoothing to capture transitions between network states better [1]. The suggested articles explore methods for detecting changes in large networks, focusing on tracking evolving groups and visualizing structural changes over time in various domains such as social and biological networks. These approaches aim to analyze dynamic networks efficiently and effectively.

2.2.1. TRNN-GCN Algorithm: Limitations

The Transitional Recurrent Graph Convolutional Network (TRNNGCN) combines Recurrent Neural Networks (RNNs) and Graph Convolutional Networks (GCNs) for dynamic clustering [10]. This algorithm introduces a decay rate to adjust the weight of historical data in clustering decisions, which allows the model to:

1. **Adapt to Changing Clusters:** Nodes with higher turnover rates can decay faster to account for recent changes, while stable nodes retain their historical weights.
2. **Learn Decay Dynamically**: Using an RNN layer, the algorithm learns optimal decay rates based on the graph's evolution, applying this decay before clustering with GCN layers.
3. **Preserve Meaning and Clarity:** The decay rates clearly indicate how much importance is given to historical versus recent data for each cluster.

2.2.2. Key Features of the TRNNGCN Approach

- **Dynamic Stochastic Block Model**: The algorithm models dynamic graphs using a stochastic block model to estimate probabilities of connections within and between clusters over time.
- **Decay Rates**: Introduces a decay rate matrix that adjusts edges' weights based on their relevance over time, improving clustering accuracy.
- **Iterative Learning**: TRNNGCN refines its clustering decisions over time steps by updating cluster memberships and decay rates.
- **Visualization**: Decay rates make the process meaningful, showing how each cluster evolves and adapts.

The Transitional Recurrent Graph Convolutional Network (TRNNGCN) algorithm clusters evolving graphs by adapting to network changes over time. It works well with dynamic graphs but uses too much memory and processing power for exceptionally large networks, making it hard to handle fast-growing data.
To address these challenges, two advanced methods are applied to handle the clustering of large data networks.

2.2.3. Selected Clustering Algorithms for Large-Scale Networks

1. **FASTGCN**:
    a. FASTGCN (Fast Graph Convolutional Network) is a scalable variation of GCN designed to handle large datasets efficiently by lowering the heavy computational load that comes with processing the entire network at once. This method uses a sampling technique. Instead of aggregating data from every node's full neighborhood, it selects a smaller sample to represent the neighborhood. This

approach saves both memory and processing time, making the algorithm much more efficient.

   b. This method allows for real-time graph data processing while maintaining high accuracy in clustering results, making it ideal for analyzing large networks.

2. **GraphSAGE++**:
   a. GraphSAGE++ builds on the original GraphSAGE (Graph Sample and Aggregate) algorithm by enhancing the aggregation process to handle even larger datasets. It introduces optimizations for distributed processing and parallelism, enabling faster computation on large-scale networks.
   b. Unlike traditional GCNs, GraphSAGE++ learns node embeddings by sampling and aggregating features from a fixed-size neighborhood, making it scalable and efficient for large dynamic graphs.

In this capstone, we discuss both FASTGCN and GraphSAGE++ in detail, exploring their architecture, advantages, and applicability to dynamic network clustering. Combining these two methods aims to create a comprehensive framework for analyzing and mapping changes in large-scale networks over time. This dual-clustering strategy addresses the limitations of TRNNGCN while providing the scalability needed for real-world applications.

3. Mathematical Background

3.1 GraphSAGE++

Before delving into the GraphSAGE++ clustering algorithm, it is important first to outline the foundational methods upon which it is built. These methods offer the necessary background and context for understanding the algorithm's operation.

3.1.1. Aggregator Functions and Key Concepts and Definitions

The aggregator function is a crucial component of GraphSAGE, responsible for combining the features of a node's neighbors into a single representation. For GraphSAGE, the aggregator must satisfy symmetry (i.e., the output should be invariant to the order of neighbors) because the neighborhood of a node does not have an inherent ordering. Below are the three main types of aggregators used:

Mean Aggregator

The mean aggregator is one of the simplest and most used aggregation methods. It computes the element-wise mean of the feature vectors of the neighboring nodes.

Mathematically:

$$h_{N(v)}^{(k)} = \frac{1}{|N(v)|} \sum_{u \in N(v)}^{b} h_u^{(k-1)}$$

- $h_{N(v)}^{(k)}$: Aggregated features for node v at layer K.
- N(v): Set of neighbors of node v.
- $h_u^{(k-1)}$: Representation of neighbor from the previous layer.

This method is computationally efficient and performs well for tasks where the aggregated features can be represented as a linear combination.

LSTM Aggregator

The LSTM aggregator leverages a Long Short-Term Memory (LSTM) network to aggregate the neighbors' features. LSTMs can capture more complex relationships between features by processing them sequentially.

Steps:

1. The neighbors' feature vectors $h_u^{(k-1)}$are treated as a sequence.
2. A random permutation of the neighbors is used since LSTMs are not inherently symmetric.
3. The LSTM processes this sequence and outputs an aggregated representation $h_{N(v)}^{(k)}$.

Mathematically, the LSTM updates are:

$$h_{N(v)}^{(k)} = \text{LSTM}\left(\left\{ h_u^{(k-1)} \mid u \in N(v) \right\}\right)$$

Pooling Aggregator

The pooling aggregator is a two-step process:

- Each neighbor's feature vector is independently passed through a fully connected neural network:

$$h_u^{\text{transformed}} = f\left(W \cdot h_u^{(k-1)} + b\right),$$

where:

- W and b are trainable parameters.
- f is a non-linear activation function (e.g., ReLU).

- An element-wise max pooling operation aggregates the transformed neighbor features:

$$h_{N(v)}^{(k)} = \max\left(\{\, h_u^{\text{transformed}} \mid u \in N(v)\,\}\right),$$

where:

- Max is applied elementwise across all the feature vectors.

The pooling aggregator captures the most prominent features across the neighborhood, making it highly capable and suitable for capturing diverse characteristics.

S_k : Fixed Number of Neighbors

S_k represents the fixed number of neighbors sampled at each layer k for every node. Sampling ensures that the computation remains efficient, even in exceptionally large graphs where nodes might have hundreds or thousands of neighbors.

For example:

- If a node v has 100 neighbors, and $S_k = 10$,GraphSAGE will randomly select 10 neighbors of v during the aggregation step.
- This fixed sampling size S_k controls the computational complexity, ensuring that the number of operations is proportional to the depth K and the sample size S_k at each layer.

Wpool and bpool : Weights and Biases for Pooling

These are trainable parameters used in the pooling aggregator:

- Wpool: Weight matrix applied to the neighbor's feature vector $h_u^{(k-1)}$.
- pool: Bias vector added after applying the weight matrix.

In the pooling aggregator:

1. Each neighbor's feature vector $h_u^{(k-1)}$ is transformed using a fully connected layer:

$$h_u^{\text{transformed}} = \sigma\left(W_{\text{pool}} \cdot h_u^{(k-1)} + b_{\text{pool}}\right))$$

- Wpool: Captures the relationship between input features.
- bpool: Adds a shift to the transformation.

2. The transformed feature vectors are aggregated using element-wise max pooling:

$$h_{N(v)}^{(k)} = \max\left(\{\, h_u^{\text{transformed}} \;\mid\; u \in N(v)\,\}\right).$$

This step ensures that the model captures the most expressive features from the neighborhood.

3.1.3 GraphSAGE++ Algorithm steps

GraphSAGE++ objective is to generate low-dimensional embeddings for nodes in a graph by sampling and aggregating features from a node's local neighborhood. This approach is particularly useful for clustering large networks, as it captures both local and global structural information [7].

1. **Graph $G = (V, E)$:**
   a. V: Set of nodes.
   b. E: Set of edges connecting nodes.
2. **Node Features x_v :**
   a. $h_v^{(k)}$ Each node v has an associated feature vector x_v.
3. **Hyperparameters:**
   a. Depth K**:** Number of layers (or hops).
   b. Aggregator functions AGGREGATE_k for $k = 1, \ldots, K$.
   c. Weight matrices w_k for each layer.
   d. Non-linearity f (e.g., ReLU).

Algorithm's steps

1. Initialization: Set the initial representation of each node to its input features:

$$h_v^{(0)} = x_v, \quad \forall v \in V$$

2. Neighborhood Sampling: Define the neighborhood $N(v)$ for each node **v.** In large graphs, uniformly sample a fixed-size subset of neighbors S_k at each layer k to ensure computational efficiency.

3. Iterative Aggregation and Update: For k= 1 to K (where K is the depth of the algorithm):

- **Step 1 - Neighbor Aggregation:** Aggregate features of neighbors $N_{(v)}$ using the k-th aggregator function:

$$h_{N(v)}^{(k)} = \text{AGGREGATE}_k\left(\left\{ h_u^{(k-1)} \mid u \in N(v) \right\}\right)$$

Examples:

- **Mean Aggregator:**

$$h_{N(v)}^{(k)} = \frac{1}{|N(v)|} \sum_{u \in N(v)}^{b} h_u^{(k-1)}$$

- **Pooling Aggregator:**

$$h_{N(v)}^{(k)} = \max\left(\sigma\left(W_{\text{pool}} h_u^{(k-1)} + b_{\text{pool}}\right) \middle| u \in N(v) \right)$$

- **Step 2 - Update Node Representation** Combine the node's current representation with the aggregated neighbor features, then apply a dense layer with non-linearity:

$$h_v^{(k)} = f\left(W_k \cdot \text{CONCAT}\left(h_v^{(k-1)}, h_{N(v)}^{(k)}\right)\right)$$

Here, **CONCAT** denotes the concatenation of vectors.

4. Normalization (Optional) Normalize node embeddings after each layer:

$$h_v^{(k)} < -\frac{h_v^{(k)}}{\left\|h_v^{(k)}\right\|}$$

5. Output After K iterations, the final embeddings for each node are:

$$z_v = h_v^{(K)} \quad \forall v \in V$$

Output

The algorithm outputs z_v for all nodes $v \in V$, where:

- Z_k : Low-dimensional embedding of node v, capturing structural and feature-based information from its K-hop neighborhood.
- These embeddings can be used for downstream tasks such as clustering, node classification, or link prediction.

Computational Complexity

GraphSAGE reduces the memory and computational footprint by:

Sampling neighbors instead of using the full neighborhood.

Fixing per-batch complexity at $O(K \cdot S_k)$, where S_k is the sample size at depth k.

Summary of the GraphSAGE Pipeline

1. **Input:** Graph, node features, depth K, and hyperparameters.
2. **Neighbor Aggregation:** Aggregates information from sampled neighbors using symmetric functions.
3. **Feature Propagation:** Combines node features with aggregated neighborhood features across K layers.
4. **Output:** Node embeddings representing the graph's local and global structure.

3.2 FastGCN algorithm

3.2.1 Definitions and Core Concepts

Before delving into the phases of FastGCN, it is necessary to clarify the mathematical and conceptual elements that the algorithm relies on. These elements are crucial for understanding its phases and overall functioning.

Graph Convolution:

- Traditional graph convolution in GCN updates the node embeddings by aggregating information from neighbors.
- The mathematical formulation is:

$$H^{(l+1)} = \sigma\left(AH^{(l)}W^{(l)}\right),$$

where:

- $H^{(l)}$: Node embeddings at the layer l .
- A: Normalized adjacency matrix of the graph.
- $W^{(l)}$: trainable weight matrix for layer l.
- σ: Non-linear activation function (e.g., ReLU).

Monte Carlo Sampling:

- FastGCN views graph convolutions as integral transforms:

$$h_v^{(l+1)} = \int^{A}(v,u)h_u^{(l)}W^{(l)}\,dP(u)$$

where $P(u)$ is a probability distribution over nodes u . This formulation enables sampling nodes u to estimate the integral using Monte Carlo methods.

Importance Sampling:

- To reduce the difference, nodes are sampled using a probability distribution $Q(u)$ designed to minimize the variance of the embedding updates. The optimal $Q(u)$ It is created to focus more on important nodes by considering their role and connections within the network.

Embedding Generation:

- Node embeddings are iteratively refined through layers. For sampled nodes, the embeddings are updated as:

$$H^{(l+1)}(v) = \sigma\left(\frac{1}{|S|}\sum_{u\in S}^{A}(v,u)H^{(l)}(u)W^{(l)}\right)$$

  where S is the sampled set of neighbors.

3.2.2 STEPS of the FastGCN Algorithm

1. **Input:**
    - A graph $G = (V, E)$
    - with:
        - V : Nodes,
        - E : Edges,
    - Node feature matrix $H^{(0)} \in R^{n\times d}$

      (where n is the number of nodes and d is the feature dimension).
    - Normalized adjacency matrix A.
    - Number of layers M.
    - Sampling distribution $Q(u)$.
    - Trainable weight matrices $W^{(l)}$ for each for each layer l .

2. **Node Sampling:**

- Instead of aggregating all neighbors, FastGCN samples a fixed number of nodes S_l at each layer l .
- Sampling is performed using a probability distribution $Q(u)$, where:

$$q(u) \propto \left||A(:,u)|\right|_2^2$$

3. **Layer-wise Convolution:**
   - For each layer l :
     - Nodes S_l are sampled,
     - The embeddings are updated using the sampled neighbors:

       $.H^{(l+1)}(v) = \sigma\left(\frac{1}{|S_l|}\sum_{u\in S_l}\frac{A(v,u)}{q(u)}\, H^{(l)}(u)W^{(l)}\right)$

4. **Variance Reduction:**
   - Importance sampling is employed to ensure that frequently sampled nodes contribute proportionally to their influence in the graph.

5. **Loss Calculation:**
   - The loss function **L** is defined over a subset of labeled nodes for semi-supervised learning. For classification:

$$L = -\sum_{i\in\text{labels}_i^{y}\log} \hat{y}_i$$

Where $\hat{y}_i$ predicted label distribution for node i .

6. **Optimization:**
   - Stochastic Gradient Descent (SGD) is used to minimize the loss. Gradients are calculated only for sampled nodes at each layer.

7. **Output:**
   - The final embeddings $H^{(M)}$ are used for downstream tasks like node classification or clustering.

3.3 HDBSCAN cluster algorithm

After generating node embeddings for the graph using algorithms such as GraphSAGE++ or FastGCN, the graph is clustered based on these embeddings using the HDBSCAN clustering algorithm [7].

3.3.1 Algorithm Steps

Step 1: Core Distance Calculation

For each point Z_i :

- Define the core distance, $core_k(Z_i)$, as the distance to its k-th nearest neighbor, where $k = m_{\text{samples}}$.

- Mathematically:

$$\text{core}_k(z_i) = d\left(z_i, z_{k^{\text{th nearest neighbor}}}\right)$$

Step 2: Mutual Reachability Distance

For any two points z_i and Z_j , compute the mutual reachability distance:

$$mreach_k\left(z_i, z_j\right) = \max\left(\text{core}_k(z_i), \text{core}_k\left(z_j\right), d\left(z_i, z_j\right)\right)$$

- Ensures points are only connected if they both belong to dense regions.
- Introduces robustness to variations in density.

Step 3: Build the Mutual Reachability Graph

- Construct a graph G where:
  - **Nodes**: Points in X .
  - **Edges**: Weighted by $mreach_k\left(z_i, z_j\right)$.

Step 4: Minimum Spanning Tree (MST)

- Compute a minimum spanning tree (MST) on G , which organizes points into a hierarchical structure:
  - **Nodes**: Same as G .
  - **Edges**: A subset of G that connects all nodes with minimal total weight.
  - Total weight:

$$\text{Total weight} = \sum_{(i,j)\in\text{edges}}^{m} reach_k(z_i, z_j)$$

Step 5: Cluster Stability

To evaluate the stability of clusters:

1. Define λ as the inverse of the mutual reachability distance:

$$\lambda = \frac{1}{mreach_k(z_i, z_j)}$$

   a. $Stability\ of\ a\ cluster\ \boldsymbol{C}$: $stability(C) = \sum_{p\in C}(\lambda_{death} - \lambda_{birth})$

   b. λ_{birth} : Value of λ when the cluster forms.

   c. λ_{death} : Value of λ when the cluster dissolves.

2. Select clusters with the highest stability.

Step 6: Extract Clusters

- Extract clusters by analyzing the dendrogram and retaining only stable clusters.
- Points not belonging to any stable cluster are labeled as noise.

Output

1. **Cluster Labels:**

   a. Assign a cluster ID (e.g., 0, 1, 2, ...) to each point in $mreach_k(z_i, z_j)$.

   b. Noise points are labeled as -1.

2. **Cluster Properties:**

   a. Number of clusters, sizes, and representative points.

4. Project Overview

4.1 Workflow

The workflow involves analyzing large-scale networks over different periods to identify patterns, filter out irrelevant data, and track how the network structure changes over time. The process is divided into stages, with each step building on the results of the previous one.

Before graph construction, the dataset undergoes a one-time preprocessing phase to ensure consistency and semantic clarity. First, articles are filtered by publication year and validated for the presence of Field of Study (FOS) metadata. Irrelevant citations are removed to create a self-contained dataset. Then, articles are mapped to predefined macro-categories using Sentence-BERT embeddings and cosine similarity. Only confidently classified articles are retained, ensuring a clean and semantically unified input for downstream analysis.

next, citation network data are prepared by creating graphs representing different periods. Nodes represent entities, such as journals, and edges denote relationships, such as citations. To improve data quality, self-citations are removed, resulting in cleaned graphs.

These graphs are then processed using embedding techniques, such as GraphSAGE++ or FastGCN, which generate compressed vector representations for each node. These embeddings capture structural and feature information, enabling clustering tasks.

The embeddings are input into the HDBSCAN algorithm, which identifies clusters without requiring a predefined number of clusters.
HDBSCAN assigns cluster labels based on dense regions in the feature space and Labels outliers as irrelevant data(noise), forming meaningful groupings such as research domains.

Using the Hungarian Method, transitions between clusters in consecutive time points are optimally matched based on shared nodes or feature similarity. This ensures precise identification of transitions such as splits, mergers, and new clusters.

Next, the clusters are analyzed to assign meaningful names based on node attributes, making computational results easier for humans to understand. This naming process makes the clusters more understandable and relevant.

Then, cluster evolution over time is visualized using alluvial diagrams, showing transitions such as splits, merges, and node movements between clusters. These visualizations highlight key trends, reveal emerging fields and focus shifts, and help distinguish significant changes from noise.

Finally, the alluvial diagrams and clustering results are reviewed to detect key trends, explain changes, and generalize the findings to other network types.

4.2 Pseudocode

Input: Dataset of citation networks for multiple time points: $\{G_1, G_2, \dots, G_T\}$, where each graph G_t consists of:

- N: Nodes (journals/articles).
- E : Edges (citations between journals).
- W_{ab}: Weight of the edge from node a to node b (number of citations).
- X: Node features matrix.

Output: Insights into network structural changes, visualized through alluvial diagrams, and interpreted meaningful trends.

Step 0. One-Time Data Preprocessing

Objective: Filter and normalize the raw dataset before constructing graphs.

Input: Raw dataset file containing:

- Article ID
- Year of publication
- List of references (citations)
- Field of Study (FOS) metadata

Output: Filtered and semantically normalized dataset suitable for analysis.

a. Filter articles based on selected year range and ensure presence of FOS metadata.

- Remove articles outside the time window [start_year, end_year].
- Remove articles without valid FOS fields.
- Eliminate citation links to papers not included in the filtered set.

b. Normalize Field of Study terms into a fixed set of macro-categories:

- Encode FOS terms using a pre-trained Sentence-BERT model.
- Compare FOS embeddings to predefined macro-category embeddings.
- Assign each article to the closest macro-category (if similarity > threshold).
- Remove articles with low confidence or no match.

Save the resulting dataset as a cleaned JSONL file to be used in subsequent steps.

Step 1. Data Preparation

Objective: Prepare and preprocess the input networks for analysis.

Input: Raw citation networks $\{G_1, G_2, \dots, G_T\}$.

Output: Preprocessed graphs $\{G_1, G_2, \dots, G_T\}$ with normalized weights and feature vectors.

a. Collect citation network data for all time points $\{G_1, G_2, \dots, G_T\}$.

b. Represent each graph G_t as a directed, weighted graph:

- Nodes N : Represent articles.
- Edges E : E_{ab} represent citation of article b in article a.

d. Integrate node feature vectors X (e.g., word occurrence vectors or metadata features).

Step 2. Embedding Generation

Objective: Transform each node in the graph into an embedding vector.

Input: Preprocessed graphs $\{G_1, G_2, \dots, G_T\}$ with node features X.

Output: Embedding vectors Z_v for each node in all graphs.

a. Input preprocessed graphs $\{G1, G2, \dots, G_T\}$ and node features X into GraphSAGE++ or FastGCN.

b. For each graph G_t, generate embedding vectors Z_v for all nodes v :

- For GraphSAGE++, aggregate features from neighbors iteratively.

OR

- For FastGCN, sample nodes layer-wise to compute embeddings.

c. Output the embedding vectors z_v for all nodes across all graphs.

Step 3. Clustering

Objective: Cluster nodes based on their embeddings to groups of similar nodes.

Input: Embedding vectors z_v for nodes in $\{G_1, G_2, \dots, G_T\}$

Output: $\{C_1, C_2, \dots, C_T\}$ Cluster labels for each node in all graphs $\{G_1, G_2, \dots, G_T\}$

a. Input the embedding vectors Z_v into the HDBSCAN clustering algorithm.

b. Use HDBSCAN to assign cluster labels to nodes in each graph G_t:

- Clusters represent groups of similar nodes.
- Noise points are labeled separately.

c. Output cluster labels C_t for all nodes in G_T.

Step 4. Tracking Cluster Evolution

Objective: Identify how clusters evolve.

Input: Cluster labels $\{C_1, C_2, \dots, C_T\}$.

Output: Cluster transitions (mergers, splits, new clusters).

- **a**. Use the Hungarian method to match clusters in $\mathrm{C_t}$ with clusters in $\mathrm{C_{t+1}}$ based on the size of shared nodes or feature similarity.

b. Identify transitions between clusters:

- Mergers: When two or more clusters in C_t combine into one cluster in C_{t+1}.
- Splits: When a cluster in C_t divides into multiple clusters in C_{t+1}.
- New Clusters: Clusters in C_{t+1} that have no counterpart in C_t.

c. Quantify transitions: Use the Hungarian method to compute the cost matrix based on shared nodes or similarity between clusters in $\mathrm{C_t}$ and C_{t+1} .

- Quantify the size and type of transitions (e.g., the percentage of shared nodes).

d. Output a map of cluster transitions across time points.

Step 5. Constructing Alluvial Diagrams

Objective: Visualize structural changes in the networks over time.

Input**:** Cluster transitions from the Hungarian method between graphs.

Output: Alluvial diagram illustrating cluster evolution.

a. Represent clusters for each time point as blocks in columns:

- The height of each block corresponds to the cluster size (number of nodes).

b. Use stream fields to connect blocks across time points:

- The width of each stream represents the transition size.
- Splits, mergers, and new clusters are highlighted.

c. Optimize visualization:

- Use smooth splines for streams.
- Exclude streams below a predefined size threshold to reduce clutter.

d. Output an alluvial diagram visualizing cluster evolution.

Step 6. Insights and Interpretation

Objective: Derive meaningful conclusions from the analysis.

Input: Alluvial diagrams and clustering results.

Output: Key trends, patterns, and their implications.

a. Analyze the alluvial diagrams and clustering results.

b. Identify key trends:

- Emerging fields or growing clusters.
- Declining clusters or merged topics.
- Stable clusters over time.

c. Explain observed changes:

- Link structural changes to external factors (e.g., research focus shifts, funding changes).

d. Generalize findings to other types of networks (e.g., social, economic).

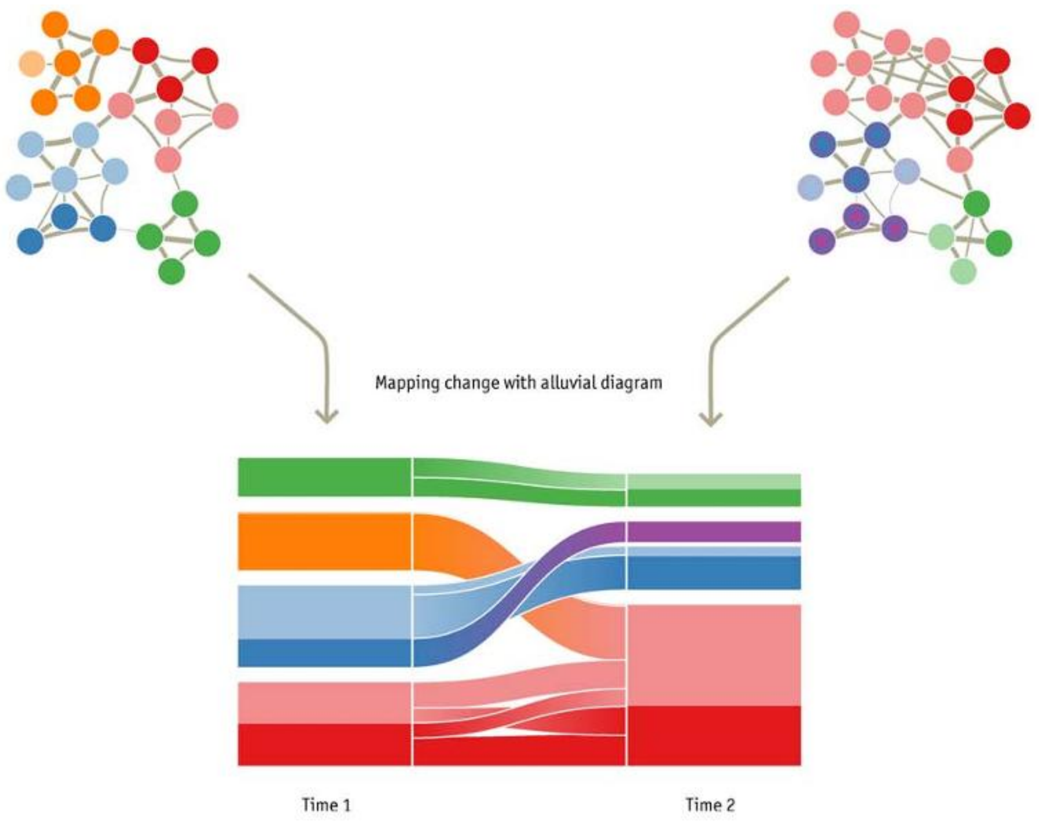


Fig. 1. The Alluvial Diagram.
This figure is taken from: [9]

5. Research / Engineering process

5.1 The Process

The research is divided into two main phases: Phase A and Phase B.

Phase A: Research and Learning

Phase A focuses on understanding core algorithms related to dynamic networks. The process began by studying "Mapping Changes in Large Networks" [9] to establish a foundation for tracking network changes over time. Following this, "Interpretable Clustering on Dynamic Graphs with Recurrent Graph Neural Networks" [10] was analyzed to explore a newer Cluster algorithm intended to replace the original method from [9].

Upon further evaluation, the Cluster algorithm from [10] was found to be insufficient for handling large citation networks. Therefore, the focus shifted to more advanced algorithms, including GraphSAGE++ [6], FastGCN [3], and HDBSCAN [7], to improve clustering performance and scalability.

Phase B: Implementation

Phase B involved implementing the selected algorithms to perform clustering on large citation networks. The methods presented in [3], [6], and [7] were adapted to manage dynamic graphs efficiently.

Following the implementation of clustering and the generation of embedding vectors from large-scale networks, the overall algorithm will be developed based on the ideas presented in "Mapping Changes in Large Networks" [9], using methods such as GraphSAGE++, FastGCN, HDBSCAN, and Hungarian Method.

This approach provides a powerful and efficient tool for mapping and tracking changes in large citation networks over time, enabling the detection of significant structural transformations and emerging research trends.

This research aims to develop a new algorithm based on modern technologies to map and track structural changes in large citation networks over time. The proposed model consists of several steps to process, analyze, and visualize changes in large-scale networks over time.

The process begins by collecting citation networks from different periods, represented as directed, weighted graphs where nodes are articles and edges are citations. The networks are preprocessed by removing self-citations and normalizing edge weights.

Once the citation networks are prepared, they are fed into a graph embedding algorithm, either GraphSAGE++ or FastGCN, to transform each node into an embedding vector. These embedding vectors capture the structural and contextual information of each node in the network.

In the clustering phase, the embedding vectors generated from the previous step are input into the HDBSCAN algorithm. This step groups nodes with similar characteristics into clusters, where each cluster represents a group of related articles or research topics.

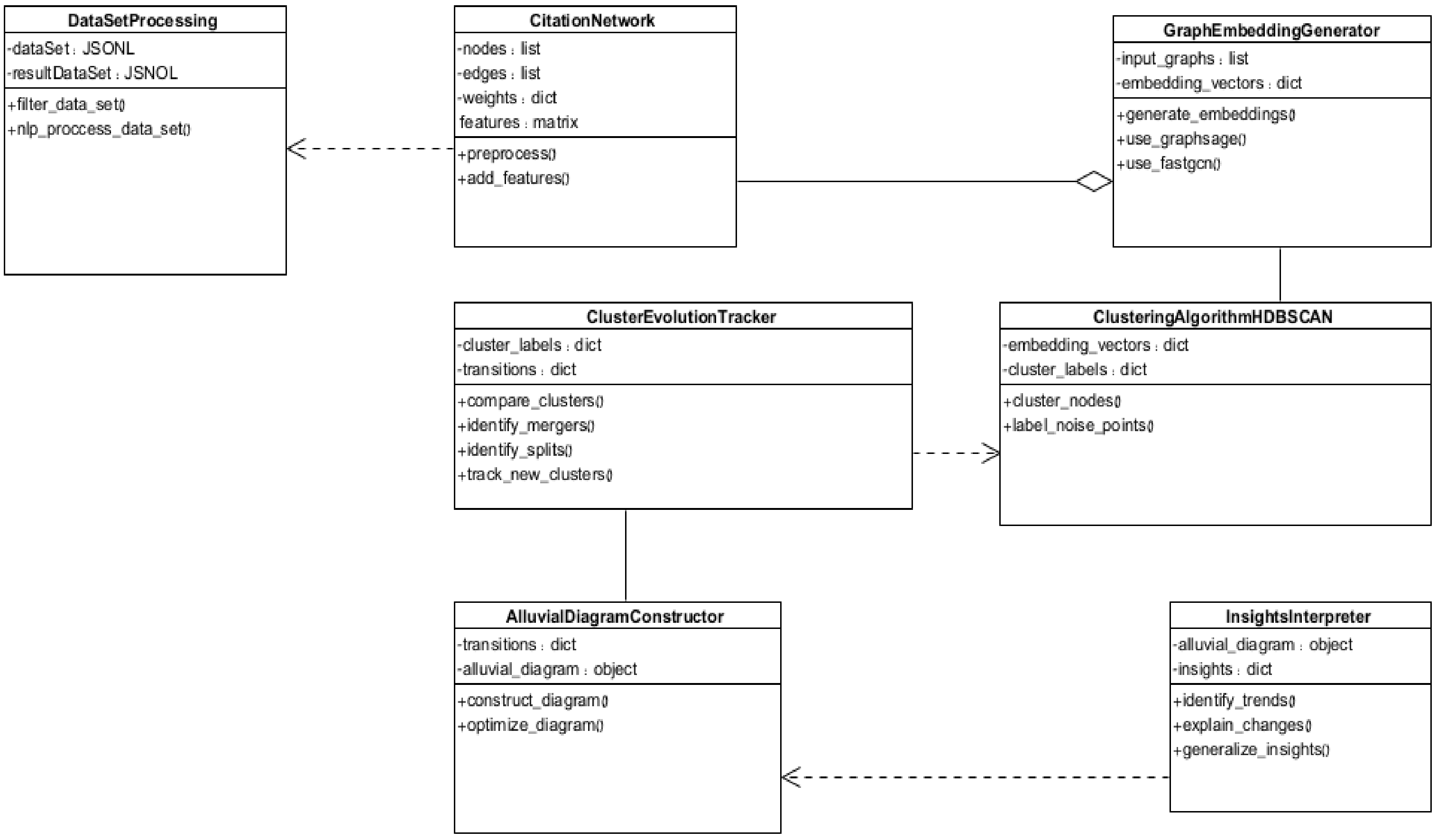
The next step is tracking how these clusters evolve over time by comparing cluster labels between consecutive graphs. To ensure precise tracking, the Hungarian Method is applied to optimally match clusters across time points based on shared nodes or feature similarity. This enhances the accuracy of identifying transitions, including mergers, splits, and the emergence of new clusters.

To visualize these changes, an alluvial diagram is constructed. The diagram represents clusters as blocks, with streamlines connecting them across different time points. The width of each streamline indicates the size of the transition, highlighting key events such as cluster splits, mergers, and the emergence of new clusters.

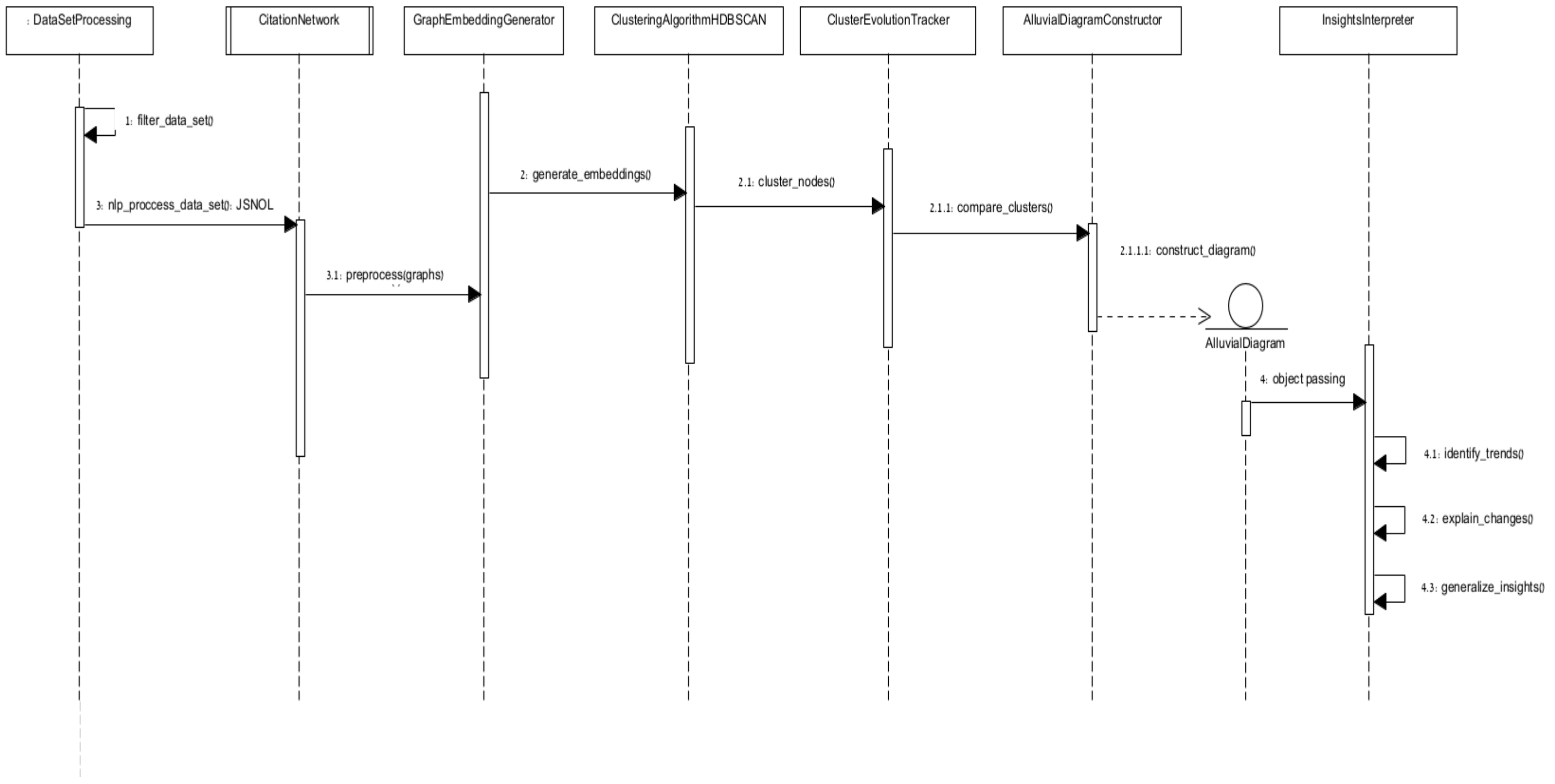
Finally, insights are derived from the clustering results and alluvial diagrams. Key trends, such as emerging research areas, declining topics, and stable domains, are identified.

5.2 Diagrams

Class diagram



Sequence Diagram



6. Evaluation and Testing Plan

In part B of the research, we will focus on implementing the advanced algorithms FastGCN, GraphSAGE++, and HDBSCAN to enhance the clustering process in large citation networks. Additionally, we will implement the complete main process described in the "Mapping Changes in Large Networks" algorithm [9], which is designed to track significant structural changes in dynamic networks over time. This main algorithm processes large-scale citation networks by detecting evolving clusters, identifying trends such as emerging research areas, and recognizing topic splits or mergers.

6.1 Evaluation

After fully implementing the framework, we will evaluate its accuracy and correctness by running the same dataset through our and existing reference implementations. The evaluation will focus on comparing the accuracy of the results produced by both implementations, ensuring that the proposed approach achieves equivalent precision in detecting clusters, emerging research areas, and structural changes in citation networks. By validating the consistency of the output, the objective is to confirm that the implementation correctly replicates the expected results while optimizing computational performance.

6.2 Testing plan

Index	Expected Result	Test Description	Component
1	The graph preprocessing test ensures the graph is free of self-citations and has normalized weights	Test graph preprocessing	CitationNetwork
2	Nodes contain feature vectors	Test feature integration	CitationNetwork
3	Embedding vectors for all nodes in the graph	Generate embeddings using GraphSAGE++	GraphEmbedding Generator
4	Embedding vectors for all nodes in the graph	Generate embeddings using FastGCN	GraphEmbedding Generator
5	Cluster labels assigned to each node	Perform clustering on embedding vectors	ClusteringAlgorith mHDBSCAN
6	Noise points labeled separately	Detect noise points	ClusteringAlgorith mHDBSCAN
7	Verify consistent clustering results across multiple runs	Cluster Stability Test	ClusteringAlgorith mHDBSCAN
8	Identify mergers, splits, and new clusters	Compare clusters between two consecutive graphs	ClusterEvolutionTr acker
9	List of newly formed clusters	rack new clusters over time	ClusterEvolutionTr acker
10	Alluvial diagram object created	Construct an alluvial diagram from cluster transitions	AlluvialDiagramC onstructor
11	List of trends, including growing and declining clusters	Identify trends based on the alluvial diagram	InsightsInterpreter
12	Explanation of why certain clusters merged, split, or appeared	Explain changes in clusters	InsightsInterpreter

7. Proposed Research Solution

This chapter presents the implementation of the proposing solution for modeling and analyzing the evolution of research domains in academic citation networks. The methodology combines semantic classification, dynamic graph modeling, node embeddings, clustering, and temporal flow visualization. The full pipeline includes:

- Dataset filtering and semantic classification
- Construction of year-wise citation graphs
- Graph embedding via GraphSAGE
- Clustering of papers and label assignment

- Per year transition analysis of clusters
- Visualization using alluvial diagrams

7.1 Dataset Overview

This project uses the Open Academic Graph (OAG v2.1), which contains metadata for millions of academic publications. Each record includes key attributes such as:

- id: Unique paper identifier
- year: Year of publication
- references: List of cited paper IDs
- FOS: Field of Study (FOS) terms

These fields serve as the foundation for constructing cumulative citation graphs. The system incrementally built a growing graph where each yearly slice included all papers and citation links from previous years, along with new publications and citations from the current year. This cumulative structure ensures that long-range citation dynamics is preserved and that topic transitions can be analyzed with temporal continuity.

Only the fields listed above were actively used in the graph construction and embedding pipeline.

7.2 Data Filtering and Preparation

The raw dataset is subjected to two major preprocessing phases to ensure its structural and semantic quality.

a. Structural Filtering and Temporal Isolation
Publications are filtered based on the selected time range, and only those actively participating in citation relationships (i.e., both citing and cited within the same range) are retained. All references to papers outside the filtered scope are removed, thereby producing self-contained and temporally isolated graphs suitable for cumulative year-wise modeling. This approach aligns with the best practices in temporal network analysis and supports consistency in longitudinal tracking.

b. Semantic Classification via Natural Language Processing

Given the variability and granularity of raw FOS terms, a semantic normalization step is introduced. Each paper's FOS list is embedded using a pre-trained Sentence-BERT model (all-MiniLM-L6-v2) and mapped to one of several predefined macro-categories. The classification process involved:

- Encoding each FOS list into a sentence embedding
- Embedding a fixed user-defined set of macro-domains (e.g., *Artificial Intelligence*, *Mathematics*)
- Computing cosine similarity between paper vectors and domain vectors
- Assigning the closest domain if the similarity exceeded a threshold (typically 0.3)

This technique reduced semantic noise and enabled consistent interpretation across temporal slices.

c. Interactive Parameterization via User Interface

A graphical interface (GUI) developed to enable users to configure parameters such as filtering years, scientific categories for classification, and visualization preferences. This modular interface enhanced flexibility and reproducibility across experimental runs.

7.3 Graph Construction and Node Embedding

- For each year in the dataset, a cumulative citation graph is constructing, Nodes represent academic publications from the current year and all previous years.
- Directed edges denote citation relationships, including both past and same-year references.

Each paper is initialized with a 32-dimensional vector derived from its macro-category (either one-hot or weighted). These embeddings are used as input to a GraphSAGE model trained on the cumulative graph. The model aggregated neighborhood information to refine node embeddings, optimizing for mean squared error (MSE) between input and output vectors-thus preserving both semantic relevance and the evolving topological structure of the network.

This cumulative approach follows standard practices in temporal network analysis and supports consistent tracking of research trends over time.

7.4 Clustering and Label Assignment

Once embeddings are generated, unsupervised clustering is performed using the Minibatch K-Means algorithm. The number of clusters was dynamically inferred from the number of distinct macro-categories identified in each yearly subset.

To make the results easier to understand, each cluster is given a clear label.
This label is chosen based on the most common Field of Study (FOS) among a sample of papers in the cluster.
The result is a short, meaningful name that helped with both analysis and visualization.

7.5 Temporal Transition Analysis

To capture the dynamics of topic evolution over time, clusters are analyzed longitudinally:

- Year-to-year mappings of cluster membership were tracked
- Transitions between clusters are inferred based on citation patterns and thematic consistency
- A scoring model is employed to account for cluster stability, growth, and influence
- "Credit transfer" mechanisms are used to simulate the impact of significant cross-cluster citations

This analytical layer enabled the identification of complex structural phenomena such as cluster mergers, splits, emergent fields, and diminishing research areas.

7.6 Visualization of Topic Evolution

Two complementary visualization tools were developed to depict the evolution of scientific domains:

- **Static Alluvial Diagrams** (HTML-based): Rendered using cumulative cluster scores and transitions, ideal for archiving and presentation.
- **Interactive Sankey Diagrams** (Plotly): Allowed users to explore topic flows and inspect cluster-level transitions across years dynamically.

The visual output offers an effective way to interpret the results, highlighting how scientific domains evolve throughout the analyzed period and supporting clear communication of the findings.

8. Reflections and Conclusions

This chapter summarizes the key insights gained, technical decisions made, and challenges encountered during the development of a system for analyzing the temporal evolution of academic research topics via citation networks. The process involved iterative experimentation, problem-driven design adjustments, and continuous evaluation of algorithmic and visual outputs.

8.1 Advantages of the Proposed Solution

The implementation of the system demonstrates several strengths that contributed to its effectiveness:

- **Scalability**: The pipeline successfully processed large academic datasets spanning multiple years, demonstrating suitability for high-volume scholarly corpora containing tens of thousands of papers.
- **Semantic Coherence**: The inclusion of a semantic classification phase, utilizing a pre-trained Sentence-BERT model, enhanced the robustness of topic representation by resolving inconsistencies in the original Field of Study metadata.
- **User Customization**: The graphical interface enabled users to define the list of macro-categories used in classification, offering adaptability across different research domains and allowing for targeted analysis configurations.

8.3 Testing Methodology

The development follows a step-by-step approach. Each part of the system is built and tested separately before it is combined with others. First, basic parts like data filtering and semantic classification are checked to make sure they work correctly. Then, the process moves on to other parts like embedding, clustering, and visualization. This method helps to catch problems early and makes sure everything works properly before moving forward.

To further validate system behavior, a synthetic dataset was constructed with known structural patterns and expected transitions. This enabled the evaluation of algorithmic correctness under controlled conditions and provided a reference baseline before applying the method to real-world academic data.

8.4 Evaluation Strategy

Evaluation efforts focus on how clear, useful, and well-organized the system's results are.

- **Cluster Label Consistency**: The topic labels that result from clustering are compared to the macro-categories defined by the user during semantic preprocessing. This helps confirm that the clusters reflect meaningful topics and produce understandable results.

- **Visualization Usability**: Alluvial diagrams are reviewed to assess the clarity and readability of the topic flows across time. Special care is taken to verify that merging, splitting, and topic persistence are visually represented in a manner that supports intuitive understanding.

- **Iterative Refinement**: Throughout the project, output is reviewed regularly, and feedback is included to improve the quality of each stage, from embeddings to final visualization.

8.5 Technical Challenges and Limitations

Despite the system's overall success, the development process presented several technical and conceptual challenges. Each challenge was addressed through algorithmic enhancements and design decisions aiming to ensure efficiency, accuracy, and usability.

Scalability and Memory Constraints

Handling large-scale academic citation graphs, especially across multiple years, requiring careful memory and performance management. Embedding and clustering operations are computationally demanding due to the size of the dataset. To overcome this, the system employed Minibatch K-Means, which allows efficient clustering by processing the data in smaller batches. Additionally, GraphSAGE models are trained independently for each year, enabling distributed processing without requiring the full temporal graph to load into memory.

Clustering Quality and Semantic Coherence

At first, clustering based on the raw Field of Study (FOS) data didn't work well because there are too many similar or overly specific terms, which lead to confusing and inconsistent groupings. This challenge is resolved by introducing a semantic normalization layer using Sentence-BERT embeddings. Each paper's FOS is embedded and matched to a predefined list of macro-categories using cosine similarity, which significantly improves the semantic coherence of clusters and the interpretability of their labels.

Complexity of Alluvial Visualization

Designing a clear and informative alluvial flow diagram to represent multi-year topic transitions requires precise layout control. The main challenge is managing vertical spacing, flow thickness, and visual continuity across years, especially as cluster sizes varied. To make the diagrams clear and easy to read, the system uses special visualization logic in Plotly. It adjusts the size of each cluster based on its importance, separates main flows from smaller ones, and adapts the layout to fit different types of data. This helps keep the diagrams organized and understandable, no matter what data is used.

Identifying Cluster Transitions and Topic Dynamics

Identifying transitions between clusters over time, especially when topics gradually change in meaning (semantic drift), or when clusters merge or split, is a challenging task. The difficulty comes from the lack of clear labels or known reference points to compare against, which made it hard to detect these changes with certainty. This is resolved through an automated transition scoring mechanism based on a penalty-reward model. The system analyzes citation links between papers across years and adjusted cluster transition scores, accordingly, assigning higher importance to transitions where papers in one cluster cited those in another. This data-driven approach enables accurate mapping of topic evolution without the need for manual inspection or subjective validation.

9. Results

This chapter presents the main findings from the execution of the system designed to track structural transformations in large-scale citation networks. The analysis is conducted in academic publications from 2000 to 2020, utilizing a filtered subset of the Open Academic Graph (OAG v2.1). The output includes visual and analytical insights into the evolution of scientific domains over time.

9.1 Cluster Evolution and Structural Trends

The system successfully identifies meaningful topic transitions between years, as reflected in the generated alluvial diagram. This diagram visualizes how research clusters emerge, merge, or decline across time. For example, a new cluster labeled "Computational Science" emerged around 2010, originating from two previously distinct clusters: "Computer Science" and "Physics." This reflects an interdisciplinary trend driven by computational methods entering physical sciences.

In contrast, fields like "Mathematics" and "Physics" remained relatively stable in structure, with gradual decreases in size, due to shifts in focus toward applied fields like "AI" and "Data Science."

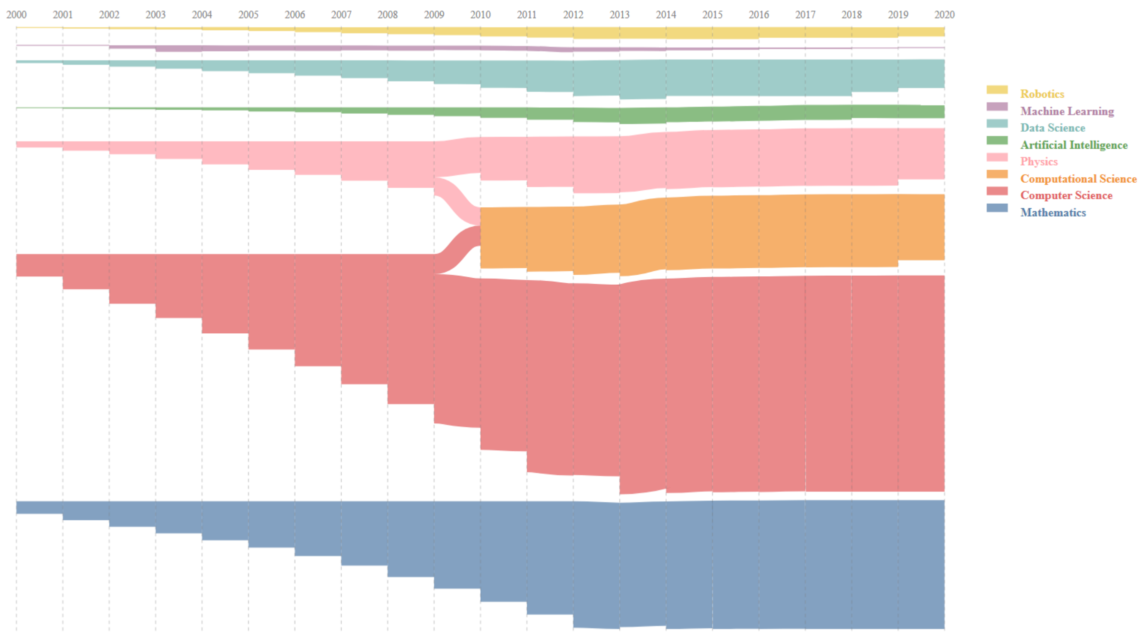


Fig. 2. Result Alluvial Diagram using Sankey.

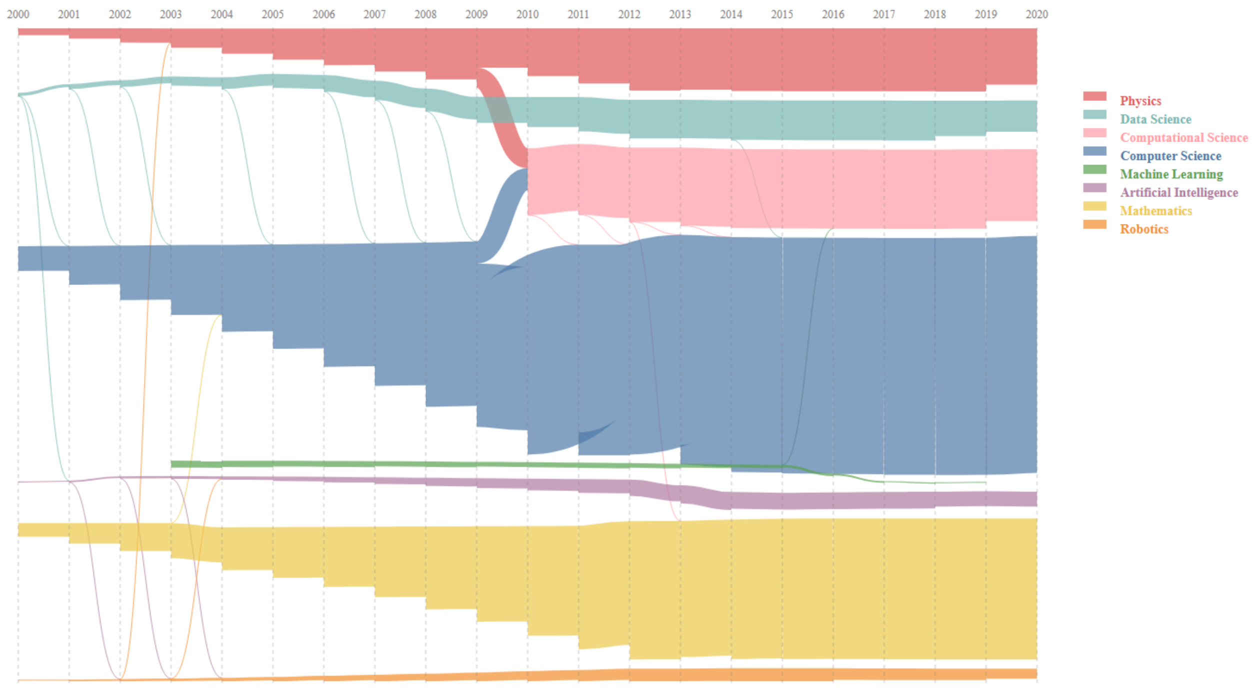


Fig. 3. Result Alluvial Diagram using Sankey (lower cited ratio).

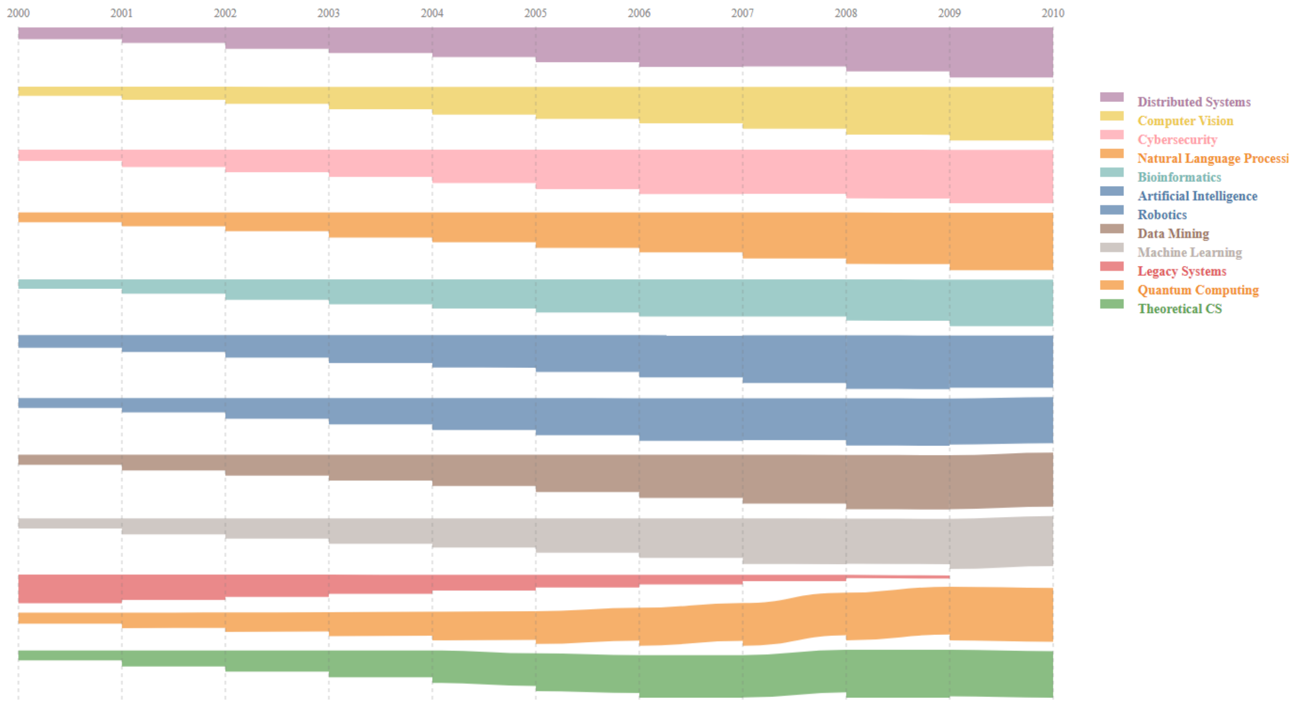


Fig. 4. Result Alluvial Diagram using Sankey.

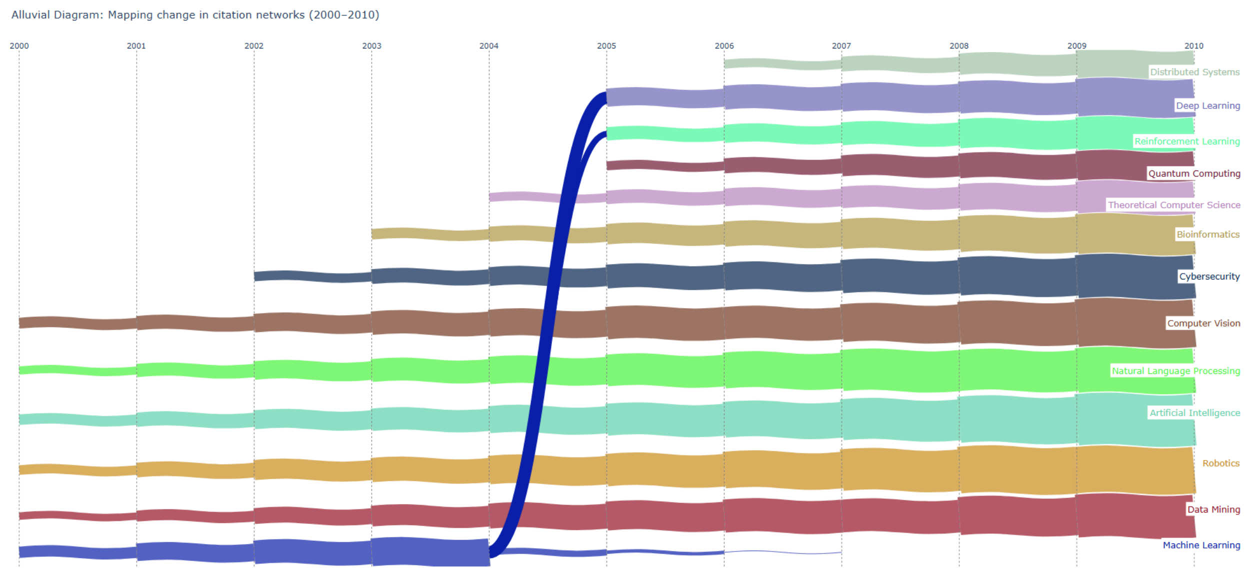


Fig. 5. Result Alluvial Diagram using Plotly.

9.2 Insights

- **Clustering Algorithm:** Minibatch K-Means ensured scalable and memory-efficient performance across time slices.
- **Label Assignment:** Semantic labeling using Sentence-BERT embeddings improves consistency and reduces noise from raw metadata.
- **Transition Detection:** The penalty-based scoring model highlights clear merging, splitting, and stable patterns in clusters.

These results align well with the expected achievements outlined earlier in the project. The system achieves its main goals by successfully detecting topic transitions, handling large-scale input efficiently, and producing understandable visualizations. Together, these outcomes demonstrate both the robustness of the methodology and the clarity of the insights it delivers.

10. User Manual

10.1. Dataset Requirements

To successfully run the system, your dataset must contain the following fields for each record:

- **year**: Year of publication
- **fos**: Field of Study
- **field_of_view**: Manually or semantically assigned macro category
- **references**: List or number of referenced paper IDs
- **id**: Unique identifier for each paper

If any of these fields are missing, the dataset will be rejected.

10.2. One-Time Data Processing

This step prepares the dataset for future analysis:

Steps:

1. Input your macro-categories (e.g., "Computer Science, Physics") in the GUI.
2. Set the year range to filter papers (e.g., 2000–2020).
3. Provide the path to the raw dataset file (.txt format).
4. Click the "Run Analysis and Generate Diagram" button.

Output Files:

- filtered_data_set.jsonl: Contains only valid records after structural filtering.
- processed_data.jsonl: Contains macro-category assignments using semantic similarity.

10.3. Running the Analysis

Once data is processed, the analysis step detects topic evolution:

Steps:

1. Define your Citing Percent and Cited Ratio thresholds.
2. Choose one or both visualization tools: Sankey (static) or Plotly (interactive).

3. Click the "Run Analysis and Generate Diagram" button.

Output:

- Sankey or Plotly-based flow diagram will open in a new window.
- Visual connections between clusters are based on strong citation flows.

10.4. Visualization Tools

The system supports two visualization options:

- **Sankey Diagram**: A static HTML-based flow chart that shows topic transitions with hover-over labels.
- **Plotly Diagram**: An interactive interface that supports zooming, saving, and tooltip inspection of flows.

Both modes help analyze how academic fields evolve across time.

10.5. Threshold Parameters Explanation

- **Citing Percent**: The percentage of articles in a source cluster that cites the target cluster.
- **Cited Ratio**: The percentage of articles cited in the target cluster referenced by the source cluster.

Only connections meeting both thresholds are displayed in the final diagram, ensuring meaningful transitions.

10.6. Output and Results

All outputs are stored in a dedicated results folder. Files are never overwritten.

Hovering over the diagram flows displays:

- Year
- Source cluster name

- Target cluster name
- Citation weight or influence score

Users can re-run the system with different inputs to compare multiple results.

11. Maintenance guide

Installation Process

- Open the project folder in your preferred Python IDE (e.g., PyCharm, VSCode).
- In PyCharm, create a new interpreter:
  Go to Settings → Project → Python Interpreter
  Click Add Interpreter → Virtualenv Environment
  Choose:
  Base Interpreter: Python 3.12
  Location: .venv inside your project
  Click OK to finish
- Open the terminal in the IDE.
- Run the following command to install all required dependencies:
  ```
  pip install -r requirements.txt
  ```
- Once dependencies are installed, launch the system with:
  ```
  python main_executor.py
  ```

Files and Paths

- Make sure the raw dataset path points to a valid .txt file.
- The output files (filtered_data_set.jsonl, processed_data.jsonl) will be saved automatically in the data_set_Processing folder.
- All generated diagrams (html) are saved in the results folder.

If you rename or move files, update the path accordingly in the interface before running any process.

Thresholds and Parameters

- Adjust Citing Percent and Cited Ratio to control the strength of connections shown in the diagram.
- These can be changed directly from the UI before generating the diagram.

Lower values = more links, higher values = stricter filtering.

Updating Macro-Categories

- If you want to analyze different scientific fields, you can change the list of macro-categories in the UI (comma-separated).
- After updating, rerun the One-Time Processing step.
- The new macro-categories will apply only after re-processing the dataset.

Frontend / UI Changes

- The user interface (Tkinter) allows you to update paths, thresholds, and plot type.
- If you update the structure of the dataset or config dictionary, make sure to reflect those changes in the UI code (main_UI.py).

Diagram Engines

- You can choose between Sankey or Plotly (or both).
- These settings can be toggled inside the interface before clicking Generate Diagram.
- No need to update the code unless you are adding a new visualization type.

Troubleshooting

- File Not Found: Make sure the dataset path is valid and ends with .txt.
- No Flows Shown: Try reducing the *Citing Percent* or *Cited Ratio* thresholds.
- Slow Performance: Avoid loading extremely large datasets; consider splitting the file.
- Python Errors: Ensure all required packages are installed via pip install -r requirements.txt.

Input Validation

- All input fields are validated automatically in the GUI.

- Citing Percent must be a number between 0 and 100.
- The Cited Ratio must be a number between 0 and 1.
- The raw dataset path is verified before filtering the system checks that the file exists.

Error Handling Logic

- If you run NLP processing only without a valid filtered file, an error will be raised.
- If you attempt to generate a diagram without a valid processed data.jsonl file, the GUI will immediately notify you.
- If no diagram type is selected (Sankey or Plotly), the GUI will also prompt an error message.

Logging

A detailed log tracks every step of the program from start to finish, including file creation, thresholds applied, transitions detected, and diagram generation status. This log can help you trace any unexpected behaviour during processing or visualization.

12. Conclusion

This project presents a complete and modular system for analyzing the evolution of scientific domains over time through academic citation networks. By leveraging real-world data from the Open Academic Graph (OAG v2.1), the system integrates key technologies: data preprocessing, semantic classification using Sentence-BERT, graph-based embeddings (GraphSAGE), unsupervised clustering (MiniBatch KMeans), and dynamic visualizations via alluvial diagrams.

The results show that this pipeline not only handles large-scale data efficiently but also provides meaningful insights. The introduction of semantic normalization significantly improves the coherence of topic groupings, resolving inconsistencies in the original metadata. The clustering process reveals interpretable research domains, and the visualizations clearly illustrate how these domains emerge, evolve, split, or decline over time.

From a usability perspective, the graphical interface allows researchers to customize parameters and explore trends interactively. This makes the system accessible and practical for both academic and applied research purposes.

In conclusion, the project achieves its main goals: it detects structural trends in citation networks, supports long-term temporal analysis, and offers an intuitive platform for exploring scientific developments. It lays a solid foundation for future studies in scientific trend discovery, knowledge mapping, and the evolution of complex systems.

13. References

[1] Aynaud, T., & Guillaume, J.-L. (2010). Static community detection algorithms for evolving networks. In *WiOpt'10: Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks* (pp. 508–514).- Static community detection algorithms for evolvingnetworks

[2] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *arXiv:0803.0476v2 [physics.soc-ph]*. - Fast unfolding of communities in large networks

[3] Chen, J., Ma, T., & Xiao, C. (2018). FastGCN: Fast learning with graph convolutional networks via importance sampling. In *Proceedings of the International Conference on Learning Representations (ICLR)*. - FastGCN Article

[4] Girvan, M., & Newman, M. E. J. (2001). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*. - Community Structure in Social and Biological Networks

[5] Guimerà, R., & Nunes Amaral, L. A. (2005). Functional cartography of complex metabolic networks. *Nature, 433*(7028), 895–900.- Functional Cartography of Complex Metabolic Networks

[6] Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *Proceedings of the Thirty-First Conference on Neural Information Processing Systems (NIPS)*. - GrapghSAGE++

[7] Ma, B., Yang, C., Li, A., Chi, Y., & Chen, L. (2023). A faster DBSCAN algorithm based on self-adaptive determination of parameters. *Procedia Computer Science, 221*, 113–120.- HDBSCAN article

[8] Palla, G., Derényi, I., Farkas, I., & Vicsek, T. (2005). Uncovering the overlapping community structure of complex networks in nature and society. *Nature, 435*(7043), 814–818.- Uncovering the overlapping community structure of complex networks in nature and society .

[9] Rosvall, M., & Bergstrom, C. T. (2010). Mapping change in large networks. *PLoS ONE, 5*(1), e8694. - Mapping Change in Large Networks | PLOS ONE

[10] Yao, Y., & Joe-Wong, C. (2021). Interpretable clustering on dynamic graphs with recurrent graph neural networks. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)* (pp. 4608–4616). - TRNGCN article