Attention Based High Dimension Representative for R2-UMAP

Yunfei Wang
M.S. in Business Analytics, Columbia University

We replace the fuzzy graph with a learned, attention-based whole graph per time window. Subgraphs are extracted via community detection, stable cores are identified, and these cores guide whole-graph reconstruction. This pipeline yields a more stable high-dimensional representative for the downstream R2-UMAP algorithm.

Step 1: Attention Whole Graph Construction

An attention mechanism learns the pairwise affinities between nodes. Models are trained using rolling window data.

Algorithm 1 Attention-Based Whole Graph Construction

Input: Rolling window $\{X_{t-w+1}, \ldots, X_t\}$, each $X \in \mathbb{R}^{N \times D}$
Output: Whole-graph adjacency A_t

Concatenate features: $X^{(\text{win})} = \left[X_{t-w+1} \parallel \ldots \parallel X_t\right] \in \mathbb{R}^{N \times (D \cdot w)}$.

Compute attention adjacency: $A_t = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)$.

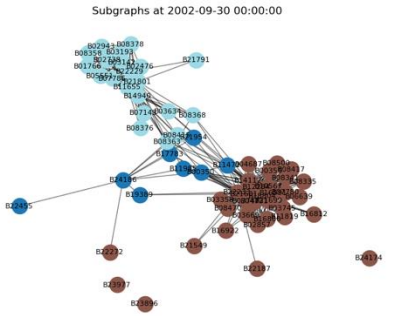
Symmetrize: $A_t \leftarrow f(A_t)$, where $f \in \{\text{fuzzy}, \max, \text{mean}\}$.

Sparsify: keep Top-K per row or edges with weight $\geq \tau$.

Output: A_t.

Step 2: Subgraph Partition via Community Detection

Given the attention-based whole graph $\tilde{A}_t$, we apply Louvain community detection to identify densely connected subgraphs at each timestamp. This yields partitions $\{C_{t,1}, C_{t,2}, \ldots\}$ that serve as candidate structures for the temporal core detection.



Step 3: Core Detection (Temporal Stability)

We identify temporally stable cores by grouping similar subgraphs across time and selecting nodes that persist within these groups, supported by stable edge categories.

Algorithm 2 Core Detection (Temporal Stability)

Input: Subgraphs $\{C_{t,i}\}$ across time; edge-category stats
Output: Core sets $\{\text{Core}_1, \text{Core}_2, \ldots\}$

Similarity: for subgraphs C_i, C_j at different timesteps, compute

$$J(C_i, C_j) = \frac{|V(C_i) \cap V(C_j)|}{|V(C_i) \cup V(C_j)|}.$$

Link similar subgraphs: add an edge if $J(C_i, C_j) \geq \theta$.
Components: take connected components as groups of similar subgraphs.
Node coverage: for each component, count timesteps a node appears: $\#\{t : v \in C_t\}$.
Stable edges: mark an edge category as stable if it recurs in $\geq T_{\text{edge}}$ timesteps.
Core rule: include v in core if $\#\{t : v \in C_t\} \geq T_{\text{node}}$ *and* v connects via stable edges.
Output: multiple core groups $\{\text{Core}_k\}$.

Step 4: Core-First Graph Reconstruction

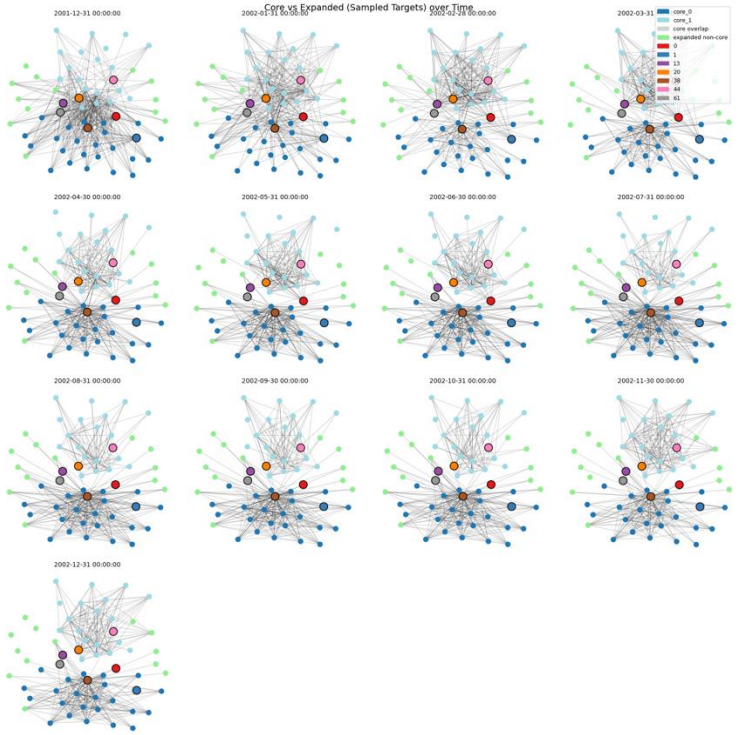
Each node preserves strong links to core nodes and then fills remaining edges up to a Top-K quota with its strongest neighbors. This ensures that reconstructed graphs remain anchored on stable cores while retaining structural diversity.

Algorithm 3 Core-First Expansion and Graph Reconstruction

Input: Base graph G_t; core set Core; Top-K, m_{core}, τ_{core}
Output: Expanded graph G_t^{recon}

foreach *node* u *in* G_t **do**
  Guarantee core links: keep at least m_{core} neighbors $v \in$ Core with weight $\geq \tau_{\text{core}}$.
  Fill to Top-K: add strongest remaining neighbors until K edges are kept.

Symmetrize final edges (e.g., by max or fuzzy).
Output: G_t^{recon}.



Future Work & Outlook

1.Parameter Exploration
The workflow contains a many variables and hyperparameters. Systematic tuning could improve stability and performance.

2. Core Detection with Memory
At present, core detection is performed independently within each time window. Future work could incorporate memory mechanisms across windows.