Knowledge Graph Theory

Formal Construction

Graph Definition

Our knowledge graph is a **typed heterogeneous multigraph** defined as:

$G = (V, E, \tau, \lambda)$

where: - V is the vertex set - E is the edge set (multiset, allowing parallel edges) - $\tau : E \to \Sigma$ is the edge type function - $\lambda : V \cup E \to A$ assigns attributes to vertices and edges

Vertex Set Construction

The vertex set V comprises two disjoint types:

$V = V_{\text{seq}} \cup V_{\text{feat}}$

where: - $V_{\text{seq}} = \{\text{token strings from genomic sequences}\}$ (sequence tokens) - $V_{\text{feat}} = \{0, 1, 2, \ldots, d-1\}$ (SAE feature indices)

with $d = \text{expansion\_factor} \times \text{hidden\_dimension}$ (typically $d = 10240$ for ef8).

Edge Construction

For each genomic sequence s with identifier $\text{id}(s)$, we construct edges as follows:

1. **Tokenization**: Sequence s is tokenized into tokens $T(s) = [t_1, t_2, \ldots, t_n]$
2. **SAE Feature Extraction**: For each token position $i \in \{1, \ldots, n\}$:
   - Compute SAE activation vector $\mathbf{f}_i \in \mathbb{R}^d$
   - Select strongest feature: $f_i^* = \text{argmax}_j(f_i[j])$
3. **Edge Addition**: For each token-feature pair (t_i, f_i^*):
   - Add edge $e = (t_i, f_i^*)$ to E
   - Assign edge type: $\tau(e) = t_i$ (the token itself types the edge)
   - Assign edge attributes: $\lambda(e)$ includes:
     - `sequence`: $\text{id}(s)$ (sequence identifier)
     - `chrom`: chromosome location (if applicable)
     - `start`, `end`: genomic coordinates
     - `strand`: strand orientation (+/-)
     - `annotations`: genomic feature metadata (genes, transcripts, exons, etc.)

Graph Properties

Directional Structure: The graph is directed: edges point from sequence tokens (V_{seq}) to feature nodes (V_{feat}), representing the activation relationship: "token t_i activates feature f_i^*"

Heterogeneity: Two vertex types with fundamentally different semantics: - Sequence vertices represent genomic context (strings) - Feature vertices represent learned SAE components (integers)

Edge Multiplicity: Multiple edges can exist between the same (token, feature) pair if that token appears in different sequences or genomic contexts.

Edge Typing: Unlike traditional knowledge graphs where edge types represent relationship semantics (e.g., "contains", "regulates"), our edges are typed by the **genomic token** that mediates the activation. This preserves the sequence-level information at the edge level.

Formal Construction Algorithm

Input: Sequences $S = \{s_1, s_2, \ldots, s_m\}$, SAE model M **Output:** Knowledge graph $G = (V, E, \tau, \lambda)$

1. **Initialize:** $V \leftarrow \emptyset, E \leftarrow \emptyset$
2. **For each** sequence $s \in S$:
    a. **Tokenize:** $T \leftarrow \text{tokenize}(s)$
    b. **Extract features:** $F \leftarrow M.\text{forward}(s)$
    c. **For each** position $i \in \{1, \ldots, |T|\}$:
        i. Get token t_i and feature activation f_i
        ii. Select strongest feature: $f_i^* \leftarrow \text{argmax}(f_i)$
        iii. Add vertices: $V \leftarrow V \cup \{t_i, f_i^*\}$
        iv. Add edge: $E \leftarrow E \cup \{(t_i, f_i^*)\}$
        v. Set edge type: $\tau((t_i, f_i^*)) \leftarrow t_i$
        vi. Assign attributes: $\lambda((t_i, f_i^*)) \leftarrow \text{extract\_metadata}(s, i)$
3. **Return** $G = (V, E, \tau, \lambda)$

Implications for Analysis

1. **Token-to-Feature Mapping**: The graph explicitly represents which genomic tokens activate which SAE features across all sequences.
2. **Feature Centrality**: Feature vertices with high in-degree represent features activated by many different tokens/contexts, indicating broad genomic patterns.
3. **Token Centrality**: Token vertices with high out-degree (to different features) represent genomic motifs that trigger diverse feature activations.
4. **Path-Based Queries**: Paths of length 2 through shared features ($\text{token}_1 \rightarrow \text{feature} \rightarrow \text{token}_2$) identify genomic contexts with similar SAE representations.
5. **Subgraph Analysis**: Community detection on feature subgraphs reveals co-activated feature clusters representing higher-level genomic patterns.

Implementation Notes

- Implemented in `hsg/featureanalysis/featureKG.py`
- Uses NetworkX MultiDiGraph for graph representation
- Genomic annotations from NCBI RefSeq GTF enriches edge attributes
- Serialized to JSON using `networkx.readwrite.json_graph.node_link_data`