Mathematical Formulation of Intermediate Generation

Let:

- $\mathcal{T}_V$: a finite set of *element types* (e.g., types of vertices or nodes)
- $\mathcal{T}_E$: a finite set of *edge types* (e.g., bond types or connection types)

Define a graph $G = (V, E, \tau_V, \tau_E)$ where:

- V is a finite set of *vertices* (elements)
- $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ is a set of *undirected edges* (no loops or multi-edges)
- $\tau_V : V \to \mathcal{T}_V$ assigns a type to each vertex
- $\tau_E : E \to \mathcal{T}_E$ assigns a type to each edge

Equivalence of Graphs

Two graphs $G_1 = (V_1, E_1, \tau_{V_1}, \tau_{E_1})$ and $G_2 = (V_2, E_2, \tau_{V_2}, \tau_{E_2})$ are *isomorphic* (or equivalent) if there exists a bijection

$$\phi : V_1 \to V_2$$

such that:

$$\begin{gathered} \tau_{V_1}(v) = \tau_{V_2}(\phi(v)) \quad \text{for all } v \in V_1 \\ \{u, v\} \in E_1 \iff \{\phi(u), \phi(v)\} \in E_2 \\ \tau_{E_1}(\{u, v\}) = \tau_{E_2}(\{\phi(u), \phi(v)\}) \end{gathered}$$

This ensures both structure and labeling are preserved under isomorphism.

Subgraph Definition

A graph $H = (V_H, E_H, \tau_{V_H}, \tau_{E_H})$ is a *subgraph* of G (denoted $H \subseteq G$) if:

- $V_H \subseteq V$
- $E_H \subseteq \{\{u, v\} \in E \mid u, v \in V_H\}$
- $\tau_{V_H} = \tau_V|_{V_H}$, i.e., the restriction of τ_V to V_H
- $\tau_{E_H} = \tau_E|_{E_H}$

Trivially, $G \subseteq G$.

Question: How to Get All Non-Trivial Connected Unique Subgraphs of G?

Let $G = (V, E, \tau_V, \tau_E)$. To enumerate all non-trivial (at least one vertex) unique subgraphs up to isomorphism. In addition, the subgraph cannot be split into two or more disconnected parts:

Step-by-Step Procedure

1. **Enumerate All Vertex Subsets:**
   For every non-empty subset $V' \subseteq V$, proceed to edge enumeration.

2. **Enumerate All Edge Subsets:**
   For each vertex subset V', consider all subsets $E' \subseteq \{\{u, v\} \in E \mid u, v \in V'\}$.

3. **Filter by Full Connectivity:**
   Accept the subgraph $H = (V', E')$ only if:

   - If $|V'| = 1$, the single vertex must have at least one edge in the original graph G
   - If $|V'| \geq 2$, then every vertex in H must have degree ≥ 1 in H

4. **Construct Labeled Subgraphs:**
   Each valid (V', E') pair defines a candidate subgraph H with inherited labeling functions $\tau_V|_{V'}$ and $\tau_E|_{E'}$.

5. **Canonical Labeling (Isomorphism Reduction):**
   Use a graph isomorphism algorithm that respects vertex and edge types to convert each subgraph into a canonical form.

6. **Filter Unique Subgraphs:**
   Maintain a set of canonical representations to eliminate isomorphic duplicates.

Output

The set of all non-trivial (i.e., $|V'| \geq 1$) and fully connected (if $|V'| \geq 2$) unique subgraphs of G, up to isomorphism.

Problem Motivation and Hashing Approach

Directly comparing each candidate subgraph via isomorphism testing is computationally expensive, especially as the number of candidate subgraphs grows exponentially with graph size.

To address this, we make use of the *Weisfeiler-Lehman (WL) graph hash* as a surrogate for canonical labeling. This enables us to efficiently test whether two graphs are isomorphic up to node and edge types, without performing pairwise isomorphism tests.

Why Full Isomorphism Checking is Expensive

Let $\mathcal{G}_k$ denote the set of all size-k subgraphs from the input graph G. To identify all unique subgraphs up to isomorphism, a naïve approach would require $O(|\mathcal{G}_k|^2)$ pairwise isomorphism checks, where each check can be expensive depending on labeling.

Weisfeiler-Lehman Graph Hashing

The Weisfeiler-Lehman method is a vertex refinement technique used to encode graph structure into compact hash values. It works as follows:

- Initially, each node is labeled with its original label (e.g., node type).
- At each iteration, the label of each node is updated by hashing its current label together with a multiset of its neighbors' labels (along with edge types, if specified).
- This process iteratively refines the labels in a way that captures structural information, producing a final string representation that is invariant under graph isomorphism.

This process yields a graph-invariant fingerprint: if two graphs have different WL hashes, they are guaranteed to be non-isomorphic. If they share the same hash, they are *likely* isomorphic (WL test is not complete for all graphs, but is highly effective in practice, especially for graphs with rich labelings).

Implementation of Weisfeiler-Lehman Hashing in `networkx`

In practice, the Weisfeiler-Lehman (WL) graph hash can be computed efficiently using the `networkx` library, which provides a built-in function called `weisfeiler_lehman_graph_hash`.

API Call and Usage

The function is accessed as follows:

```
from networkx.algorithms.graph_hashing import weisfeiler_lehman_graph_hash

hash_string = weisfeiler_lehman_graph_hash(
    G,
    node_attr="type",
```

```
    edge_attr="type"
)
```

Parameters:

- `G`: the input graph (can be a subgraph)
- `node_attr`: name of the node attribute to use in hashing (e.g., `"type"`)
- `edge_attr`: name of the edge attribute to use (e.g., `"type"`)

Output: A canonical string that encodes the graph's topology and labeled structure. Two graphs with the same hash string are treated as equivalent (isomorphic with respect to labels).

Preprocessing and Notes

- To ensure that node labels do not affect the hash (only structure and types), each subgraph is relabeled to use integer node indices before hashing:

  ```
  H_relabel = networkx.convert_node_labels_to_integers(H)
  ```

- To maximize performance, type labels are stored as integers (or strings) rather than complex objects (e.g., Enums).
- The hash values are used to deduplicate subgraphs during enumeration:

  ```
  if wl_hash not in seen_hashes:    ...
  ```

Complexity Analysis

The problem of enumerating all non-trivial, fully connected, unique subgraphs of a labeled graph $G = (V, E, \tau_V, \tau_E)$ involves several computational steps, each with its own complexity considerations.

Subgraph Enumeration

The number of labeled subgraphs is exponential:

$$O(2^{|V|} \cdot 2^{|E|})$$

and each subgraph requires isomorphism checking, so efficient hashing or pruning is critical for tractability on large graphs.

Overall Complexity of Weisfeiler-Lehman Hash Computation

For each valid subgraph $H \subseteq G$, a WL hash is computed. The runtime for WL hashing is:

$$O(k(|V_H| + |E_H|))$$

where k is the number of refinement iterations (typically a small constant), and V_H, E_H are the vertex and edge sets of the subgraph. Since each subgraph is typically small, this step is efficient and can be considered nearly linear per subgraph.

After computing the hash string for each subgraph, we store it in a hash set to check for uniqueness. This provides:

$$O(1)$$

expected time per subgraph for lookup and insertion (amortized), assuming good hash distribution and a suitable hash function (e.g., Weisfeiler-Lehman).

Let N denote the number of candidate subgraphs generated and tested. The overall complexity is:

$$O(N \cdot (|V_H| + |E_H|)) \quad \text{(WL hash per subgraph)}$$

where $|V_H|$ and $|E_H|$ are typically small constants for molecular graphs or motifs.

In the worst case (e.g., for dense graphs with many small motifs), the algorithm remains exponential in $|V|$, but the combination of pruning (via degree filtering) and hashing makes it tractable for moderate-sized sparse graphs.