Q3: Discriminative Subgraph Identification

Names: Parth Singh and Anand Sasikumar
Entry Nos.: 2022ME11438 2022CS11087

Approach

Goal

We need a small set of subgraph "fragments" (features) that help filter database graphs for a given query graph. If a query graph is a subgraph of a database graph, then every fragment present in the query must also be present in that database graph. Using this monotonic property, we prune candidates using feature containment.

Feature definition

Instead of running expensive subgraph isomorphism, we represent each graph as a set of small, easy-to-extract fragments. We extract the following fragments from every graph:

- **Edge features:** a single edge represented using the labels of its two endpoints and the edge label.
- **Path length-2 features (3 nodes, 2 edges):** a chain of two edges sharing a middle node, represented using endpoint labels, edge labels, and center node label (canonicalized so direction does not matter).
- **Path length-3 features (4 nodes, 3 edges):** a longer chain that captures more structural context than length-2 paths, also canonicalized to remove direction ambiguity.
- **Triangle features (3-cycle):** a 3-node cycle with node labels and edge labels, canonicalized by checking permutations and choosing a stable minimum representation.

These fragments are "subgraph-like" and fast to compute from the adjacency list, which makes them suitable for indexing.

Mining fragments and scoring

To avoid fragments that are too rare (weak filters) or too common (appear in almost every graph), we do:

1. Read all database graphs.
2. Extract the fragment set for each graph.
3. Count the **support** of each fragment: number of database graphs containing it.
4. Compute a discriminativeness score using only the support fraction:

$$p = \frac{\text{support}}{N}, \qquad \text{score}(f) = p(1-p)$$

This score is highest near $p = 0.5$, so it prefers fragments that appear in "some but not all" graphs.

Selecting discriminative subgraphs

We sort fragments by the score $p(1-p)$ (and break ties by support and a stable ordering), and select the top K fragments as our final "discriminative subgraphs". In our final version we used $K = 200$ because it improves pruning on large datasets (e.g., NCI-H23) while remaining efficient.

The selected fragments are saved to the output path provided as `<path_discriminative_subgraphs>`.

Why this works for candidate filtering

For every database graph g, we create a binary feature vector $v_g \in \{0,1\}^K$:

$$v_g[j] = 1 \text{ if fragment } f_j \text{ is present in } g, \text{ else } 0.$$

For every query graph q, we create v_q in the same feature space.

Filtering rule:

$$g \text{ is a candidate for } q \implies v_q \leq v_g \quad \text{(component-wise)}.$$

This uses the property: if $q \subseteq g$, then every fragment found in q must also be present in g.

This produces a candidate set that is much smaller than the full database, reducing the cost of later verification.

Practical considerations

- We treat graphs as undirected during feature extraction (molecular graphs are typically undirected).
- We canonicalize fragment encodings so the same structure maps to the same feature key.
- We keep fragment sizes small (edges/paths/triangles) so extraction is fast even for large databases.

Outcome

This approach produces a compact and effective set of discriminative fragments and achieves strong pruning (small candidate sets) while keeping computation simple and within the assignment constraints.