COL761: Graph Indexing via Discriminative Subgraph Selection

Team: PyMining [2025SIY7623, 2025SIY7622, 2022TT11351]

February 4, 2026

1 Introduction

Graph indexing is a critical bottleneck in managing large-scale molecular databases where exhaustive subgraph isomorphism testing is computationally prohibitive[cite: 201]. This report details a feature-based indexing approach designed for Task 3, leveraging frequent subgraph mining to build a discriminative filter.

2 Methodology and Implementation

The current implementation follows a **naive feature-selection logic**. While functional, it is recognized that this approach is not the most optimized for speed or absolute accuracy in complex scenarios. The pipeline relies heavily on the robustness of the **Gaston** engine to extract a meaningful initial set of frequent subgraphs[cite: 138, 139].

2.1 Feature Extraction Engine: Why Gaston?

Based on empirical analysis from Task 2, Gaston was selected as the primary engine[cite: 124]. In comparative tests against gSpan and FSG, Gaston demonstrated superior runtime efficiency while identifying a consistent set of common subgraphs[cite: 125]. Its speed is vital for the "Identify" phase of the indexing pipeline[cite: 257].

3 Discriminative Subgraph Selection

We employ a frequency-based distribution strategy to select $k = 50$ fragments[cite: 231].

3.1 Canonical Representation

To handle graph isomorphism and ensure node-order invariance, all mined subgraphs are converted into **canonical labels**. The graphs are treated as undirected to focus on structural connectivity patterns.

3.2 Fragment Distribution (2-Edge vs. 3-Edge)

Drawing inspiration from the path-based approaches in **GraphGrep**, we target fragments of size 2 and 3 edges. This provides a balance between structural detail and computational cost.

3.3 Frequency-Based Pruning

The 50-dimensional binary feature vector is constructed by sampling from the frequency distribution:

- **2-Edge Fragments (25 total):** The 12 most frequent and 13 least frequent (within the frequent set) labels.
- **3-Edge Fragments (25 total):** The 12 most frequent and 13 least frequent (within the frequent set) labels.

4 Inspiration and Theoretical Basis

The logic for this indexing scheme is rooted in pre-neural network graph literature:

- **GraphGrep:** Inspired the use of small structural fragments (paths/edges) as basic filtering units to prune the search space efficiently.
- **gIndex:** Informed the strategy of using *discriminative* frequent structures rather than just frequent ones. While our current implementation is naive, it aims to emulate gIndex's principle that frequent but redundant subgraphs add little pruning power compared to rare frequent fragments.

5 Limitations and Future Scope

The current naive implementation can be improved in several ways:

- **Dynamic Support Filtering:** Future iterations could implement more rigorous pruning of the canonical labels based on specific support thresholds to eliminate redundant features.
- **Complexity Scaling:** Moving beyond 3-edge fragments to larger, more complex discriminative structures as suggested in the gIndex paper.

6 References and Acknowledgments

- **Literature:**
  - Shasha, D., et al. "Algorithmics and applications of tree and graph searching" (GraphGrep).
  - Yan, X., et al. "Graph Indexing: A Frequent Structure-based Approach" (gIndex).
- **Tools:** Large Language Models, including **ChatGPT** and **Gemini**, were utilized as collaborative tools for debugging Python logic, optimizing script performance, and structuring this report.

7 Indexing and Filtering

For each graph $g_i \in D$, we construct a binary vector $v_i \in \{0,1\}^{50}$[cite: 232]. The candidate set for a query q is defined as:

$$C_q = \{g_i \in D \mid v_q \subseteq v_i\}$$

where $v_q \subseteq v_i$ denotes the component-wise inequality $v_q[j] \leq v_i[j]$[cite: 236].