COL761/COL7361/AIL7026
Data Mining
Assignment 1 Report

Task 3: Graph Indexing

---

1 Graph Indexing for Subgraph Search

1.1 Problem Definition

Let $D = \{g_1, g_2, \ldots, g_n\}$ be a database of labeled graphs and let q be a query graph. The task is to return all database graphs that contain q as a subgraph under subgraph isomorphism.

Formally, the true answer set is

$$R_q = \{\, g_i \in D \mid q \subseteq g_i \,\}.$$

Since exact subgraph isomorphism testing is NP-complete, performing verification against every database graph is computationally infeasible for large datasets. Instead, we first generate a *candidate set* C_q using inexpensive filters.

The only strict correctness requirement is:

$$R_q \subseteq C_q,$$

i.e., **zero false negatives**. Missing even a single true match invalidates the output for that query.

At the same time, the evaluation metric rewards smaller candidate sets. Hence, the goal is:

> Prune as aggressively as possible using only conditions that are **necessary** for subgraph isomorphism.

1.2 Overall Approach

Each graph G is represented by a binary feature vector

$$v(G) \in \{0, 1\}^F,$$

where each component corresponds to the presence of a necessary structural property or fragment.

The key monotonicity property we enforce is:

$$q \subseteq g \;\Rightarrow\; v(q) \leq v(g) \quad \text{(component-wise)}.$$

Candidate generation then reduces to a simple containment test:

$$C_q = \{\, i \mid v(g_i) \geq v(q) \text{ component-wise} \,\}.$$

This filtering step may produce *false positives* (extra candidates), which are acceptable, but never false negatives.

In practice, the containment check is implemented efficiently using vectorized NumPy operations: `np.all(db_matrix >= q_vec, axis=1)`.

1.3 Feature Design

All features are binary lower-bound constraints. Each feature corresponds to a property that must also hold in any supergraph. Thus, all features are monotone under subgraph embedding.

We use a union of several complementary feature groups.

1.3.1 Size Constraints

- `NV` $\geq k$: at least k vertices
- `NE` $\geq k$: at least k edges

Justification: Subgraph embeddings are injective, so $|V(q)| \leq |V(g)|$ and $|E(q)| \leq |E(g)|$.

1.3.2 Node-Label Multiplicity

- `Label` $L \geq k$: at least k nodes with label L

Justification: Node labels must be preserved by isomorphism, so counts cannot decrease.

1.3.3 Edge-Type Multiplicity

- `(uLabel, edgeLabel, vLabel)` $\geq k$

Justification: Edges in the query map to distinct edges with identical labels in the database graph; therefore, typed-edge counts are also lower bounded.

1.3.4 Degree Constraints

- `Deg`$(L) \geq k$: a node with label L has degree at least k

Justification: Embedding cannot reduce the degree of mapped nodes.

1.3.5 Cycle Presence

- `Cycle` ≥ 1

Justification: If q contains a cycle, all cycle edges must also appear in g.

1.3.6 Hashed Local Structural Fragments (Main Pruning Power)

To obtain stronger pruning, we include small local patterns:

- length-2 wedges (H2)
- length-3 labeled paths (H3)
- labeled star neighborhoods (HS)

For any fragment signature S,

$$\text{count}_g(S) \geq \text{count}_q(S).$$

Binary features of the form "$S \geq t$" are generated.

Hashing. Since the number of distinct signatures can be large, we hash them into a fixed number of buckets using a stable MD5 hash. Collisions only merge signatures and, therefore, may add false positives but *cannot remove true matches*. Hence, correctness is preserved.

1.4 Correctness Guarantee (Zero False Negatives)

Claim. The candidate generator produces no false negatives.

Proof. Let $g \in R_q$. Then $q \subseteq g$. Every feature is monotone under subgraph embedding, therefore $v(q) \leq v(g)$ component-wise. Hence g passes the containment test and is included in C_q.

1.5 Effectiveness of Pruning

The evaluation metric rewards smaller candidate sets:

$$s_q = \frac{|R_q|}{|C_q|}.$$

Adding more independent monotone constraints increases the probability that non-matching graphs violate at least one feature, thus shrinking C_q.

In practice, the local hashed fragments provide the strongest discrimination because they capture structural information beyond simple label or size counts.

1.6 Implementation Details

- Duplicate database graphs are removed while preserving original order.
- Candidate indices are written in increasing 1-based order.
- Feature computation and filtering are fully vectorized using NumPy.

1.7 Execution Pipeline

```
bash q1_1.sh \
    "./apriori/apriori/src/apriori" \
    "./fpgrowth/fpgrowth/src/fpgrowth" \
    "generated_transactions.dat" \
    "./output_task1_2"
```