Graph Indexing for Subgraph Isomorphism Queries

Assignment 3 - Part 3

2024EEY7601, 2023CS10438, 2023CS51067

February 3, 2026

Abstract

This report presents a graph indexing system designed to efficiently filter candidate graphs for subgraph isomorphism queries. The implementation employs a two-stage cascaded filtering approach: a global feature-based filter using binary histograms (node labels, edge patterns, and degrees) followed by a neighborhood consistency check inspired by the INCOMPLETELAD algorithm. Tested on the Mutagenicity and NCI-H23 datasets, the system ensures zero false negatives ($C_q \supseteq R_q$) while reducing the search space by approximately 45–60% without the high computational cost of full verification.

1 Introduction

Graph indexing addresses the challenge of querying large graph databases where exhaustive subgraph isomorphism testing is computationally prohibitive. Given a database $\mathcal{D} = \{g_1, \ldots, g_n\}$ and a query q, the objective is to retrieve a candidate set C_q such that $\{g \in \mathcal{D} \mid q \sqsubseteq g\} \subseteq C_q$. The primary goal is to maximize pruning power (minimize $|C_q|$) while maintaining index construction and query processing efficiency.

The system was evaluated on two molecular graph datasets:

- **Mutagenicity**: 4,337 compounds, 14 atom labels.
- **NCI-H23**: 40,353 compounds, 65 atom labels.

Both datasets have undirected labeled graphs, requiring careful handling of edge duplication common in molecular data formats.

2 Approach

2.1 Design Philosophy

The proposed solution uses a **cascaded filtering** architecture. Rather than relying on a single complex check, the system applies progressively expensive filters:

1. **Global Histogram Filter:** Fast, bitwise elimination using structural invariants.
2. **Neighborhood Consistency Filter:** Local structural verification to remove false positives from Stage 1.

Per the assignment requirements, the final output is the candidate set C_q, effectively prioritizing soundness and speed over the perfect precision of a full VF2 verification.

2.2 Stage 1: Feature-Based Global Filtering

2.2.1 Feature Selection

Instead of mining complex k-node subgraphs (which is computationally expensive for large databases), I employed a hybrid feature set. A graph g is mapped to a binary feature vector v_g indicating the presence (1) or absence (0) of:

1. **Node Labels:** Presence of specific atom types (e.g., C, O, N).
2. **Frequent Edge Patterns:** The top frequent labeled edges (2-node subgraphs) mined with a minimum support of 10%.
3. **Degree Distribution:** Presence of nodes with degrees $0, 1, \ldots, d_{max}$.

Why Edge Patterns? Mining frequent edges is $O(|E|)$ compared to the exponential cost of mining larger subgraphs. In molecular graphs, edge labels (bond types) combined with endpoint labels (e.g., C-Double-O) are highly discriminative while remaining cheap to index.

2.2.2 Vectorization & Filtering

The feature vector dimension d is $|L| + |P| + (d_{max} + 1)$. The filtering condition is based on the necessary condition for isomorphism: if $q \sqsubseteq g$, then any structural feature present in q must exist in g.

$$v_q[i] \leq v_g[i] \quad \forall i \in [0, d-1]$$

This allows for vectorized comparison using NumPy, processing thousands of graphs in milliseconds.

2.3 Stage 2: Neighborhood Consistency

Survivors of Stage 1 undergo a local check inspired by INCOMPLETELAD [6]. For $q \sqsubseteq g$ to hold, every node $u \in V(q)$ must map to a distinct node $v \in V(g)$ such that:

- Labels match: $\ell(u) = \ell(v)$
- Neighborhoods are compatible: $N_q(u) \subseteq N_g(v)$ (in terms of labels)

This check runs in polynomial time, significantly faster than the factorial complexity of full subgraph matching, yet it prunes graphs that are globally similar but locally incompatible.

3 Implementation

3.1 System Architecture

The solution is implemented as a modular pipeline in Python 3.10, orchestrated by Bash scripts.

- `identify.sh`: Mines discriminative features from the database. It handles edge deduplication (canonical ordering of source/dest) to ensure undirected edges are counted correctly.
- `convert.sh`: Generates feature vectors. It outputs `.npy` files for efficient I/O.
- `generate_candidates.sh`: Performs the actual querying. It loads the vectors, applies the Stage 1 mask, and then iterates through survivors to run the Stage 2 neighborhood check using `networkx`.

3.2 Key Algorithms

Feature Mining: To handle the "double edge" representation in the input files (where $u \to v$ and $v \to u$ are listed separately), the miner stores edges as canonical tuples '(min(u,v), edge-label, max(u,v))' before counting frequencies.

Filtering Logic:

```
for q_idx, q_vec in enumerate(q_matrix):
    # Stage 1: Vectorized Global Filter
    # Returns indices where database vector contains all features of query
        vector
    survivors_mask = np.all(q_vec <= db_matrix, axis=1)
    candidates = np.where(survivors_mask)[0]

    final_candidates = []
    Q = q_nx[q_idx]

    # Stage 2: Structural Checks
    for db_idx in candidates:
        G = db_nx[db_idx]
        # Basic edge count check
        if G.number_of_edges() < Q.number_of_edges():
            continue
        # Neighborhood consistency
        if check_neighborhood_consistency(G, Q):
            final_candidates.append(db_idx + 1)

    write_output(q_idx, final_candidates)
```

Listing 1: Cascaded Filtering Logic

4 Experimental Results

The implementation was tested on the provided Baadal VM environment using the standard datasets.

4.1 Dataset Profiles

Based on the execution logs, the feature extraction module identified:

- **Mutagenicity**: 14 labels, 13 frequent edge patterns (minsup=0.1), Max Degree 8.
- **NCI-H23**: 65 labels, 10 frequent edge patterns (minsup=0.1), Max Degree 10.

4.2 Filtering Performance

The system's pruning effectiveness was measured over 50 random queries per dataset. Table 1 summarizes the final candidate set sizes.

Table 1: Candidate Set Sizes (Lower is Better)

Dataset	Total DB Size	Min $\lvert C_q \rvert$	Max $\lvert C_q \rvert$	Avg $\lvert C_q \rvert$
Mutagenicity	4,337	764	2,355	**1,858.9**
NCI-H23	40,353	5,751	38,626	**22,729.4**

Table 2 breaks down the pruning contribution of each stage. Stage 1 (global features) provides the initial bulk reduction, while Stage 2 (local structure) refines the set further.

Table 2: Filtering Effectiveness (Pruning Rates)

Stage	**Mutagenicity**	**NCI-H23**
Stage 1 (Global)	44% pruned	27% pruned
Stage 2 (Local)	+13% pruned	+17% pruned
Overall Pruning	**57%**	**44%**

4.3 Computational Efficiency

The cascaded approach offers a significant speed advantage over naive verification. Stage 1 operates in $<$ 1ms per query using vectorized NumPy operations. Stage 2, while more complex, only runs on the surviving subset, averaging 10–100ms depending on candidate set size. Total query time averages $\approx$ 50ms.

Table 3: Approach Comparison vs. Baselines

Approach	**Time/Query**	**Avg** $\lvert C_q\rvert/\lvert\mathcal{D}\rvert$	**Est.** s_q
Naïve VF2	10-60s	100%	1.0
Histogram only	$<$ 1ms	56-73%	0.1-0.3
Two-Stage (Ours)	$\sim$ **50ms**	**43-56%**	**0.4-0.7**

4.3.1 Analysis

- **Mutagenicity**: The system filtered out approximately 57% of the database. The smaller label set makes binary features less discriminative, but the neighborhood check effectively compensates.
- **NCI-H23**: The average candidate set size is large (56% of the DB). This highlights the trade-off of using *binary* features; many large molecules satisfy the "at least one" condition for most features. However, strict soundness ($C_q \supseteq R_q$) is maintained.

5 Challenges and Solutions

5.1 The "Degree Doubling" Bug

Initial experiments showed poor pruning because degree features were not matching. **Root Cause:** The input format lists undirected edges as two directed edges (u, v and v, u). Naively iterating the file doubled every node's degree (e.g., a Carbon with 4 bonds appeared as degree 8). **Fix:** Implemented a set-based deduplication using canonical edge keys '(min(u,v), max(u,v))' before calculating degrees.

6 Conclusion

The implemented graph indexing system successfully demonstrates the efficacy of cascaded filtering. Experimental results confirm that combining rapid global feature checks with local neighborhood verification dramatically reduces the search space—achieving **57% pruning** on Mutagenicity and **44%** on NCI-H23.

The two-stage design proves critical to this performance: the constant-time global filter eliminates the bulk of non-candidates (27–44%) in under 1ms, while the neighborhood check provides a necessary second layer of refinement (+13–17%) to remove locally inconsistent graphs. With an average query time of $\sim$50ms, the system avoids the prohibitive overhead of naive VF2 search while maintaining strict soundness ($C_q \supseteq R_q$).

While the binary feature representation ensures speed and simplicity, the results on NCI-H23 suggest that including *count-based* features (e.g., "contains $\geq$ 3 Carbon rings") would significantly improve pruning power for larger, denser datasets. However, the current system successfully balances trade-offs to deliver a robust, reproducible, and efficient indexing pipeline.

References

[1] Kotthoff, Lars, Ciaran McCreesh, and Christine Solnon. "Portfolios of subgraph isomorphism algorithms." *Learning and Intelligent Optimization* (LION 10). Springer, Cham, 2016. 107-122.

[2] Morris, C., et al. "TUDataset: A collection of benchmark datasets for learning with graphs." *arXiv preprint arXiv:2007.08663* (2020).

[3] Kazius, J., et al. "Derivation and validation of toxicophores for mutagenicity prediction." *J. Med. Chem.* 48.1 (2005).

[4] Wale, N., et al. "Comparison of descriptor spaces for chemical compound retrieval." *Knowl. Inf. Syst.* 14.3 (2008).

[5] McCreesh, C., and Prosser, P. "The shape of the search tree for the maximum clique problem." *ACM Trans. Parallel Comput.* 2.1 (2015).

[6] Solnon, Christine. "Alldifferent-based filtering for subgraph isomorphism." *Artificial Intelligence* 174.12-13 (2010): 850-864.

[7] Cordella, L. P., et al. "A (sub) graph isomorphism algorithm for matching large graphs." *IEEE TPAMI* 26.10 (2004).

[8] Yan, X., and Han, J. "gSpan: Graph-based substructure pattern mining." *IEEE ICDM* (2002).