Graph Indexing

Aleksandr Tkachenko
Why indexing?

Consider a classical graph query problem:

Given a graph database \( D = \{g_1, g_2, \ldots, g_n\} \) and a graph query \( q \), find all the graphs in which \( q \) is a subgraph.

- Sequential scan is costly.
- Need to check subgraph isomorphisms.
Indexing graphs: path-based indexing

• Principle:
  • Index all paths in a graph database up to $maxL$
  • Use index to find every graph that contains all the paths (up to $maxL$ edges) in query graph $q$.

• Advantages:
  – Paths are easier to manipulate than trees and graphs,
  – the index space is predefined

• Disadvantages:
  – Structural information is lost
  – there are too many paths

• Example

A Sample Database

A Sample Query
“Can we use graph structure instead of path as the basic index feature?”

• Yes:

• By mining **frequent** structures
  – **Frequent structures**: those with support > threshold.
  – **Problem**: If only frequent structures are indexed, how to answer those queries which only have infrequent ones?
  – **Solution**: replace the uniform support constraint with a size-increasing support function with
    • very low support thresholds for small structures and
    • high thresholds for large ones

• By selecting **discriminative** structures
Framework for Graph Query Processing

• Index construction
  – Evaluate and select indexing substructures of graphs in a database.

• Query processing
  – Search:
    • identify all the features in a query graph, q
    • identify matching candidate set, so that each candidate contains all q’s features
  – Fetching:
    • Retrieve the graphs in the candidate answer set from disks.
  – Verification:
    • check the graphs in the candidate set whether they satisfy the query.
Cost Model

\[ T_{\text{search}} + |C_q| \times (T_{\text{io}} + T_{\text{iso\_test}}) \]
Good indexing

• The indexed fragments should have high frequency.
• The index needs to have the “downward-complete” property.
• The indexed fragments should be discriminative.
Discriminative fragments

**Observation:** among similar fragments with the same support, it is often sufficient to index only the smallest common fragment.

![Sample Database](image)

![Sample Query](image)

**Discriminative fragment** – matches less graphs than all its subgraphs.
Gindex algorithm

• (1) discriminative fragment selection
• (2) index construction
• (3) search
• (4) verification
Discriminative fragment selection

**Algorithm 1 Feature Selection**

Input: Graph database $D$, Discriminative ratio $\gamma_{min}$, Size-increasing support function $\psi(l)$, and Maximum fragment size $maxL$.

Output: Feature set $F$.

1: let $F = \{f_\emptyset\}$, $D_{f_\emptyset} = D$, and $l = 0$;
2: while $l \leq maxL$ do
3:     for each fragment $x$, whose size is $l$ do
4:         if $x$ is frequent and discriminative\(^4\) then
5:             $F = F \cup \{x\}$;
6:         $l = l + 1$;
7: return $F$;
Index construction

• Goal: enable efficient storage and retrieval of discovered discriminative fragments.
• Approach: translate fragments into sequences and hold them in a prefix tree.
  – use canonical labeling (DFS coding)

\[
\langle (0, 1, X, a, X), (1, 2, X, a, Z), (2, 0, Z, b, X), (1, 3, X, b, Y) \rangle
\]
Index construction

Using such labeling, each fragment can be mapped to an edge sequence

In tree all fragments reference matching graphs in the DB
Search

Given a query q:

• Extract all its fragments up to a maximum size,
• Locate them in the index,
• Intersect the id lists associated with these fragments

<table>
<thead>
<tr>
<th>Algorithm 2 Search</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Graph database $D$, Feature set $F$, Query $q$, and Maximum fragment size $maxL$.</td>
</tr>
<tr>
<td><strong>Output:</strong> Candidate answer set $C_q$.</td>
</tr>
</tbody>
</table>

1: let $C_q = D$;
2: for each fragment $x \subseteq q$ and $size(x) \leq maxL$ do
3:    if $x \in F$ then
4:        $C_q = C_q \cap D_x$;
5: return $C_q$;

• **Apriori pruning:** if a fragment is not in the prefix tree, we need not check its super-graphs any more.