Graph Database Indexing

Presenter: Wensheng TANG, James Kit Yung LAM
Present Date: April 19th, 2018
Computer Science and Engineering Department
The Hong Kong University of Science and Technology
Outline

• Introduction

• Why do Graph Indexing?

• Conventional Method:
  • Metrics Indexing

• Feature-based Indexing:
  • Path-based
  • Tree-based
  • Subgraph-based
Introduction:
Relational DB vs Graph DB

- Relational Databases
- Graph Databases
Introduction: Relational DB vs Graph DB

- Question: All of the **Friends** of any **User** who has **Liked** one of my **Posts**

```sql
SELECT friends_of_likers.*
FROM posts
JOIN likes ON (posts.post_id = likes.post_id)
JOIN users likers ON (likers.user_id = likes.user_id)
JOIN friends ON (likers.user_id = friends.user_id)
JOIN users friends_of_likers ON (friends_of_likers.user_id = friends.friend)
WHERE posts.author = :me
ORDER BY friends_of_likers.username ASC
```
Introduction: Relational DB vs Graph DB

• Relational Databases
• Graph Databases
Introduction:
Relational DB vs Graph DB

• Question: All of the **Friends** of any **User** who has **Liked** one of my **Posts**

MATCH (:User {id:{author}})
<-[:AUTHOR]- (:Post)
<-[:LIKES]- (:User)
<-[:FRIENDS]- (u:User)
RETURN (u)
Introduction: Query on a Graph DB

Querying a set of graphs

Given a set of graphs $D = \{G_1, \ldots, G_n\}$ and a query graph $Q$, find all graphs $D_Q = \{G | \text{Match}(Q, G) = 1, G \in D\}$, where $\text{Match}$ is a Boolean function.
Introduction: Application

- Application on Molecular Substructure Search

Question:
Each time we want to query a molecule substructure on a huge amount of molecule structures, do we need to traverse from nodes to nodes (edges to edges) from the very beginning?
Outline

• Introduction

• Why do Graph Indexing?

• Conventional Method:
  • Metrics Indexing

• Feature-based Indexing:
  • Path-based
  • Tree-based
  • Subgraph-based
Problem: Index on Graph DB

- Motivation: Graph search need to be fast!
- Challenge on Graph Search:
  - Structure search is hard (\textbf{NP}-complete)
  - Require to process large graphs/set of graphs
  - Require a lot of Disk I/Os
Problem: Index on Graph DB

• Make indices on relational DB is simple:
  • Usually, an index involves less than two entities.
  • Database engine supports
Graph Search w/o Index

Traverse each from nodes to nodes in each graph.

Query Response Time:

\[ T = \sum_{i=1}^{\mid D \mid} (T_{Qi} + T_{io}) \]

Number of all graph in set D

Disk I/O Time

Time of finding subgraph Q on each graph
Graph Search with Index: Basic Idea

If graph $G$ contains query graph $Q$, $G$ should contain any substructure of $Q$.

Index substructures of a query graph to prune graphs that do NOT contain these substructures (conservative strategy).
Graph Search with Index: Steps

1. **Database**
   \[ D = \{ G_1, \ldots, G_n \} \]
   
   - Each node is a feature
   
   - Index construction

2. **Query (Q)**
   
   - **(1) Search**
     - Features: \( f_1, f_2 \)
     - Candidate set: set of graphs containing all the features
     - \( C_Q = \bigcap_{f \in Q, f \in F} D_f \)
     - \( F \) is the set of features
   
   - **(2) Fetching**
     - Retrieve the candidate graphs from the disk
   
   - **(3) Verification**
     - Check if the candidates satisfy the query
   
   - Query processing
Graph Search with Index: Cost Analysis

• Query response time w/ index:
\[ T_{index} + |C_Q| \times (T_{io} + T_{isomorphism}) \]

• Query Response Time w/o index:
\[ T = \sum_{i=1}^{\left| D \right|} (T_{Qi} + T_{io}) \]

Remarks:
• \( |D| \gg |C_Q| \), the cost is greater obviously!
• Index methods should make \( |C_Q| \) as small as possible.
Outline

• Introduction
• Why do Graph Indexing?
• Conventional Method:
  • Metrics Indexing
• Feature-based Indexing:
  • Path-based
  • Tree-based
  • Subgraph-based
Metrics Indexing

• Problem: is $h$ similar to $v$?

Index based on each node’s Euclidean Length of $k$-nearest neighbors.

Example: Using 5-nearest neighbors.

$NN_5(v) = \{u_1, u_2, u_3, u_4, u_5\}$

$d(v, u_i) = \|v - u_i\|_2$

$= \sqrt{(v - u_i)^2}$
Metrics Indexing

• Problem: is $h$ similar to $v$?

Find $u_i$ such that $d(h, v) + d(v, u_i) \leq r$

Clearly, the metric form $d(h, u_i) \leq d(h, v) + d(v, u_i)$ satisfies $\triangle$-inequalities. Obviously, $d(h, u_i) \leq r$

Answer: $\{u_1, u_2, u_3\}$

Expected Answer: $\{u_1, u_2, u_3, u_4, u_5\}$

Effective but has False Negatives!

Outline

• Introduction
• Why do Graph Indexing?
• Conventional Method:
  • Metrics Indexing
• Feature-based Indexing:
  • Path-based
  • Tree-based
  • Subgraph-based
Feature-based Indexing

Different approaches use different types of features:

1. **Paths (GraphGrep)**
   - easy to compute and to manipulate
   - generate many false positive candidates

2. **Trees (Tree+$\Delta$, GCoding, GString)**
   - Easier to manipulate than subgraphs, and more efficient
   - Generate more false positives than subgraphs

3. **Subgraphs (gIndex, C-Tree, GDIndex, FG-Index, Turbo$_{ISO}$)**
   - Generate fewer candidates
   - Complex structures, generate bigger indexes
Path-based Indexing

1. Enumerate paths of a specific length
   - 0-length: C, O, N, S
   - 1-length: C-C, C-O, C-N, N-N, S-O
   - 2-length: C-C-C, C-O-C, C-N-C, ...
   - 3-length: ...

2. Build an inverted index between paths and graphs
   \[ S_C = \{a, b, c\}, S_O = \{a, b, c\} \]
   \[ S_{C-C} = \{a, b, c\}, S_{C-N} = \{a, b, c\} \]
   \[ S_{C-N-C} = \{a, b\}, ... \]

[James et al. 2003, Shasha et al. 2002]
Path-based Indexing

• Query on

• Decompose the query graph Q into paths and compute intersection among candidates

$$ S_C = \{a, b, c\}, S_O = \{a, b, c\} $$
$$ S_{C-C} = \{a, b, c\}, S_{C-N} = \{a, b, c\} $$
$$ S_{C-N-C} = \{a, b\}, ... $$

• Intersection is \{a, b\}

• Verify if graphs in the set \{a, b\} really contain query graph (Subgraph Isomorphism, \textbf{NP-Complete})

[James et al. 2003, Shasha et al. 2002 ]
Feature-based Indexing

Different approaches use different types of features:

1. **Paths** (GraphGrep)
   - Easy to compute and to manipulate
   - Generate many false positive candidates

2. **Trees** (Tree+Δ, GCoding, GString)
   - Easier to manipulate than subgraphs, and more efficient
   - Generate more false positives than subgraphs

3. **Subgraphs** (gIndex, C-Tree, GDIndex, FG-Index, TurboISO)
   - Generate fewer candidates
   - Complex structures, generate bigger indexes
Tree-based Indexing

• Why trees?
  • Tree features are easier to compare
    • Subgraph isomorphism can be polynomial on ordered trees
  • Tree features are more expressive than paths
    • Paths generate more candidates than trees since they are less restrictive.
  • Most of the discovered frequent patterns are trees!!!
    • Frequent tree-features and graph-features share similar distributions and frequent tree-features have similar pruning power like graph-features
  • Tree mining can be done much more efficiently than graph mining on G
Tree-based Indexing: Tree + $\Delta \geq$ Graph

- **Index features**
  - Frequent **tree** features
  - a small number of discriminative **graph**-features that can prune graphs effectively, **on demand**, **without costly graph mining**

[Zhao, P. et al, PVLDB’2007]
Tree-based Indexing: Tree + $\Delta \geq$ Graph

- Discriminative Graph Features $\Delta$:

  - Pruning Power: $\text{power}(g) = \frac{|D|-|D_g|}{|D|}$
    (Greater if it can prune more candidates)
  
  - If $\text{power}(T(g)) \approx \text{power}(g)$, there is no need to index the graph-feature $g$, because its subtrees $T(g)$ jointly have the similar pruning power
  
  - if $\text{power}(g) \gg \text{power}(T(g))$, it will be necessary to select $g$ as an index feature because $g$ is more discriminative than $T(g)$, in terms of pruning more candidates

(Example next page)

[Zhao, P. et al, PVLDB’2007 ]
Tree-based Indexing: Tree + $\Delta \geq$ Graph

- Pruning Power:

$$power(Fig.\,7a) = \frac{|D| - |D_{Fig.\,7a}|}{|D|} = \frac{3 - 2}{3} = \frac{1}{3}$$

$$power(Fig.\,7b) = \frac{|D| - |D_{Fig.\,7b}|}{|D|} = \frac{3 - 1}{3} = \frac{2}{3}$$

- Note: all sub-trees for Fig.7(a) are sub-tree of C-C-C-C-C-

$$power(T(Fig.\,7a)) = power(T(C - C - C - C - C)) = 0$$

- $power(Fig.\,7a) \gg power(T(Fig.\,7a))$

[Zhao, P. et al, PVLDB’2007]
Tree-based Indexing: Tree + Δ ≥ Graph

- Evaluation

Figure 9: Index Construction on The Real Dataset
(a) Feature Size  (b) Index Size  (c) Construction Time  (d) Average Time

Figure 11: Filtering Cost
(a) N=1000  (b) N=2000  (c) N=4000  (d) N=8000  (e) N=10000

[Zhao, P. et al, PVLDB’2007 ]
Feature-based Indexing

Different approaches use different types of features:

1. **Paths (GraphGrep)**
   - easy to compute and to manipulate
   - generate *many* false positive candidates

2. **Trees (Tree+Δ, GCoding, GString)**
   - Easier to manipulate than subgraphs, and more efficient
   - Generate *more* false positives than subgraphs

3. **Subgraphs (gIndex, C-Tree, GDIndex, FG-Index, Turbo_{ISO})**
   - Generate *fewer* candidates
   - Complex structures, generate *bigger* indexes
Path-based approach Weakness

• Path-based approach has week points
  • Path is too simple: structural information is lost
  • There are too many paths: the set of paths in a graph database usually is huge

• Solution
  • Use graph structure instead of path as the basic index feature

Sample Database

Query

Paths in Query Graph

Cannot Filter Any Graphs In Database
Subgraph-based Indexing

• “Can we use a graph structure instead of a path as the basic index feature?”
  • Indexes only “frequent subgraphs”
  • Creates a smaller index
  • Improves query times

**Advantage**: Generate fewer candidates

**Disadvantage**: Complex structures, generate bigger indexes
Subgraph-based Indexing

- gIndex, C-Tree, GDIndex, FG-Index, TurboISO
- Exact subgraph matching
  - Find graphs in DB which have all components of the query graph
- Similarity subgraph matching
  - Find graphs in DB which have some components of the query graph
  - Similarity measure is needed
- Super graph matching
  - Find graphs in DB which are contained in the query graph
subgraph-based approach: \texttt{gIndex} [Yan et al., SIGMOD’04]

- Find and Index only \textbf{frequent structures} in graph DB
  - subgraphs that appear often in DB
- Prune redundant frequent structures to maintain a small set of \textbf{discriminative structures}
  - Create smaller index
- Create an \textbf{inverted index} between discriminative frequent structures and graphs in the database
How to define frequent structures

- **support**\( (g) \)
  - The number of graphs in DB, where \( g \) is a subgraph

- **minSup**
  - Minimum support threshold
  - Index a fragment, \( g \) only if \( \text{support}(g) \geq \text{minSup} \)

- **Size-increasing support**
  - Frequent fragments are increasing as the size of a fragment increases
  - Low \( \text{minSup} \) for small fragments, high \( \text{minSup} \) for large fragment

![Exponential curve graph](image)
Example

Size=1

Size=2

Size=3

Size=4
Discriminative structures in glIndex

- **Redundant fragment**
  - The indexed graphs by a fragment are also indexed by its subgraphs
  - don’t need to include redundant fragments
  
  \[ D_x \approx \bigcap_{f \in F} D_f \]

- **Discriminative fragment**
  - Fragments which are not redundant

\[ D_x \ll \bigcap_{f \in F} D_f \]

- **Examples**
  - Size=2
    - \( f_1 = \{A, B\} \)
    - \( D_{f_1} = \{g_1, g_2, g_3\} \)
    - \( f_2 = \{A, B, B\} \)
    - \( D_{f_2} = \{g_2, g_3, g_4\} \)
  
  - Size=3
    - \( f_3 = \{A, A, B\} \)
    - \( D_{f_3} = \{g_2, g_3\} = D_{f_1} \cap D_{f_2} \)
Discriminative structures

• Mine useful structures
  • Given a set of features $f_1, f_2, \ldots, f_n$ and a new structure $x$, measure the probability of reconstructing $x$ having already indexed $f_1, f_2, \ldots, f_n$
  • $P( x \mid f_1, f_2, \ldots, f_n ), f_i \sqsubseteq x$

• Advantage of indexing $x$
  • When $P$ is small enough, $x$ is a discriminative feature and should be included in the index

$$\gamma_x = \frac{1}{P(x \mid f_1, f_2, \ldots, f_n)}$$

• $\gamma_x$ is called discriminative ratio of $x$.

• A feature $x$ is discriminative is $\gamma_x \geq \gamma_{\text{min}}$

A feature $x$ is discriminative if $\gamma_x \geq \gamma_{\text{min}}$
GIndex - Construction

• First generates all frequent fragments while taking out redundant ones
• Translates fragments into sequences and holds them in a prefix tree
  • Graph Sequentialization
    • DFS coding
    • Translate a graph into a unique edge sequence

• gIndex Tree
  – Prefix tree which consists of the edge sequences of discriminative fragments
  – Record all size-n discriminative fragments in level n
  – Black nodes \( \rightarrow \) discriminative fragments
  – Have ID lists: the ids of graphs containing fi
  – White nodes \( \rightarrow \) redundant fragments; for Apriori pruning
GIndex - Searching

• Searching process
  • Given a query q, enumerate all q’s fragments (size \( \leq \) maxSize)
  • Locate the fragments in gIndex tree
  • Intersect the id lists associated with the fragments

• Apriori pruning
  • Generating every fragment is inefficient
  • If a fragment is not in gIndexTree, we need not check its super-graphs any more
  • Redundant fragments need to be recorded for Apriori pruning
Experimental Result

• The index size of glIndex is more than 10 times smaller than that of GraphGrep;

• glIndex outperforms GraphGrep by 3 to 10 times in various query loads;
Experimental Result

- Data is from an AIDS Antiviral Screen Dataset
Experimental Result

**Figure 9: Low Support Queries**

**Figure 10: High Support Queries**
Thank You!

Q & A