Efficient Subgraph Matching by Postponing Cartesian Products

Lijun Chang

Lijun.Chang@unsw.edu.au
The University of New South Wales, Australia

Joint work with Fei Bi, Xuemin Lin, Lu Qin, Wenjie Zhang
Outline

- Introduction & Existing Works
- Challenges of Subgraph Matching
- Our Approach
  - Core-First Decomposition based Framework
  - Compact Path Index (CPI) based Matching
- Experiments
- Conclusion
Introduction

- **Subgraph Matching**
  Given a query $q$ and a large data graph $G$, the problem is to extract all subgraph isomorphic embeddings of $q$ in $G$. 
Introduction

- **Subgraph Matching**
  Given a query $q$ and a large data graph $G$, the problem is to extract all subgraph isomorphic embeddings of $q$ in $G$. 

![Diagram](a) Query $q$  
![Diagram](b) Data graph $G$
Introduction

**Subgraph Matching**

Given a query $q$ and a large data graph $G$, the problem is to extract all subgraph isomorphic embeddings of $q$ in $G$.
Introduction

- **Subgraph Matching**
  
  Given a query $q$ and a large data graph $G$, the problem is to extract all subgraph isomorphic embeddings of $q$ in $G$. 

(a) Query $q$

(b) Data graph $G$
Introduction

➤ Applications

- Protein interaction network analysis
- Social network analysis
- Chemical compound search
Hardness

- Subgraph Isomorphism Testing is \textbf{NP-complete}
  - Decide whether there is a subgraph of $G$ that is isomorphic to $q$

- Enumerating all subgraph isomorphic embeddings is \textbf{NP-hard}

- Many techniques have been developed for efficient enumeration in practice
Existing Work

- **Ullmann’s algorithm [J.ACM’76]**
  - Iteratively maps query vertices one by one to data vertices, following the **input order** of query vertices.
  - **Cartesian Products** between vertices’ candidates.

- **VF2 [IEEE Trans’04] and QuickSI [VLDB’08]**

- **Turbo_{ISO} [SIGMOD’13]**

- **Boost_{ISO} [VLDB’15]**
Existing Work

- Ullmann’s algorithm [J.ACM’76]
- VF2 [IEEE Trans’04] and QuickSI [VLDB’08]
  - Independently propose to enforce connectivity of the matching order to reduce Cartesian products caused by disconnected query vertices.
  - QuickSI further removes false-positive candidates by first processing infrequent query vertices and edges.
- TurboISO [SIGMOD’13]
- BoostISO [VLDB’15]
Existing Work

- Ullmann’s algorithm [J.ACM’76]

- VF2 [IEEE Trans’04] and QuickSI [VLDB’08]

- Turbo\textsubscript{ISO} [SIGMOD’13]
  - Compress a query graph by merging together similar vertices (i.e., with the same neighborhoods)
    - Reduce Cartesian product caused by similar query vertices
    - Build a data structure online to facilitate the search process.

- Boost\textsubscript{ISO} [VLDB’15]
Existing Work

- Ullmann’s algorithm [J.ACM’76]
- VF2 [IEEE Trans’04] and QuickSI [VLDB’08]
- TurboISO [SIGMOD’13]

- BoostISO [VLDB’15, Ren and Wang]
  - Compress a data graph $G$ by merging together similar vertices in $G$.
  - Develop query-dependent relationship between vertices in $G$.
    - dynamically reduces duplicate computations.
  - Can be applied to accelerate all previous techniques as well as ours

It is still challenging for matching large query graphs.
Challenges of Subgraph Matching

Challenge I: Redundant Cartesian Products by Dissimilar Vertices.

10^5-100 partial mappings are redundant.

Cartesian products: 100 X 1000 = 10^5

No similar vertices in q or G.

Matching order of QuickSI and Turbo_ISO: (u_1, u_2, u_3, u_4, u_5, u_6).

Match dense subgraph first: (u_1, u_2, u_5, u_3, u_4, u_6)
Challenges of Subgraph Matching

Our Solution: Postpone Cartesian products.

- Decompose $q$ into a dense subgraph and a forest, and process the dense subgraph first.

- The dense subgraph has more edge-connectivity information.

- We are the first to exploit this feature.
Challenges of Subgraph Matching

Challenge II: Exponential number of embeddings of query paths in a data graph.

- Turbo\textsubscript{ISO} builds a data structure that materializes all embeddings of query paths in a data graph
  1. for generating matching order based on estimation of #candidates.
  2. for enumerating subgraph isomorphic embeddings.

- Effective only when the number of embeddings is small

- Worst-case space complexity: $O(|V(G)|^{v(q) - 1})$. 

Challenges of Subgraph Matching

Our Solution: We propose a polynomial-size data structure to avoid enumerating all embeddings of a query path in the data graph.
Our Approach

- CFL-Match
  - A Core-First Decomposition based Framework
  - Compact Path-Index (CPI) based Matching
Core-First Decomposition

- Core-Forest Decomposition
  Compute the **minimal connected** subgraph containing **all non-tree edges** of \( q \) regarding any spanning tree.

- Forest-Leaf Decomposition
  Compute the set of **leaf vertices** by rooting each tree at its connection vertex.
Framework

- A Core-First Decomposition based Framework
  1) Core-First (Core-Forest-Leaf) Decomposition

(a) Query q
Framework

> A Core-First Decomposition based Framework

1) Core-First (Core-Forest-Leaf) Decomposition
2) Mapping Extraction
   i. Core-Match
   ii. Forest-Match
   iii. Leaf-Match

• Categorize leaf nodes according to labels
• Perform combination instead of enumeration among different labels.
Compact Path-Index based Matching

- **Auxiliary Data Structure: Compact Path-Index (CPI)**
  - Compactly stores candidate embeddings of query spanning trees.
  - Prunes invalid candidates
  - Serves for computing an effective matching order.
    - Estimate #matches for each root-to-leaf query path based on CPI
    - Add query paths to the matching order in increasing order w.r.t. #matches

- **CPI Structure**
Compact Path-Index based Matching

 Auxiliary Data Structure: Compact Path-Index (CPI)

- Compactly stores candidate embeddings of query spanning trees.
- Prunes invalid candidates
- Serves for computing an effective matching order.
  - Estimate #matches for each root-to-leaf query path based on CPI
  - Add query paths to the matching order in increasing order w.r.t. #matches

 CPI Structure

(a) Query $q$

(b) CPI
CPI-based Matching

- CPI Structure
  - **Candidate set**: each query node $u$ has a candidate set $u.C$.
  - **Edge set**: there is an edge between $v \in u.C$ and $v' \in u'.C$ for adjacent query nodes $u$ and $u'$ in CPI if and only if $(v, v')$ exists in $G$.

- Traverse CPI to find mappings for query vertices

(c) CPI

$G$ is probed only for non-tree edge validation

$G = (u_0, u_1, u_4, u_3, u_2, u_5, u_6, u_7, u_8, u_9, u_{10})$
Minimizing the CPI

- **Benefits of minimizing the CPI**
  - Less memory consumption
  - Fast embedding enumeration

- **Soundness of CPI**
  
  For every query node $u$ in CPI, if there is an embedding of $q$ in $G$ that maps $u$ to $v$, then $v$ must be in $u.C$.

  **Theorem**
  
  Given a sound CPI, all embeddings of $q$ in $G$ can be computed by **traversing only the CPI** while $G$ is only probed for non-tree edge checkings.

- It is NP-hard to build a minimum sound CPI.
CPI Construction

$v_9$ is pruned from $u_3.C \leftarrow$ edge $(u_3, u_4)$;
$v_1$ is pruned from $u_1.C \leftarrow$ edge $(u_1, u_3)$;
$v_8$ is pruned from $u_2.C \leftarrow$ edge $(u_1, u_2)$;
$v_{17}$ is pruned from $u_5.C \leftarrow$ edge $(u_2, u_5)$;
$v_{27}$ is pruned from $u_9.C \leftarrow$ edge $(u_5, u_9)$.
Build a small CPI

- General Idea
  - A heuristic approach:
    1) $u.C$ is initialized to contain all vertices in $G$ with the same label as $u$
    2) A data vertex $v$ is pruned from $u.C$, if $\exists u' \in N_q(u)$, such that $\nexists v' \in N_G(v) \land v' \in u'.C$.

- A two-phase CPI construction process:
  - Top-down construction, bottom-up refinement
  - Exploit the pruning power of both directions of every query edge.
  - Construct CPI of $O(|E(G)| \times |V(q)|)$ size in $O(|E(G)| \times |E(q)|)$ time
Experiment

- All algorithms are implemented in C++ and run on a machine with 3.2G CPU and 8G RAM.

- **Datasets**
  - **Real Graphs**
  - **Synthetic Graphs**
    - Randomly generate graphs with 100k vertices with average degree 8 and 50 distinct labels.

- **Query Graphs**
  - Randomly generate by random walk
  - Two Categories:
    - S: sparse (average degree ≤ 3).
    - N: non-sparse (average degree > 3).

|       | |V| | |E| | |∑| |Degree |
|-------|---|---|---|---|---|---|---|---|
| HPRD  | 9460 | 37081 | 307 | 7.8 |
| Yeast | 3112 | 12519 | 71  | 8.1 |
| Human | 4674 | 86282 | 44  | 36.9 |
Comparing with Existing Techniques

CFL-Match: our proposed algorithm

Varying the size of query graph $|V(q)|$
Effectiveness of Our New Framework

- Match: subgraph matching algorithm with CPI but no query decomposition.
- CF-Match: only core-forest decomposition with CPI.
- CFL-Match: our best algorithm.

Evaluating our framework
Scalability Testing

(a) Synthetic (vary $|V(G)|$)

(b) Synthetic (vary $d(G)$)

(c) Synthetic (vary $|\Sigma|$)

(d) Index Size (vary $|\Sigma|$)
Conclusion

- A core-first framework for subgraph matching by postponing Cartesian products
- A new polynomial-size path-based auxiliary data structure CPI, and efficient and effective technique for constructing a small CPI
- Efficient algorithms for subgraph matching based on the core-first framework and the CPI
- Extensive empirical studies on real and synthetic graphs
Thank you!

Questions?

Lijun.Chang@unsw.edu.au