Graphs-at-a-time: Query Language and Access Methods for Graph Databases

Huahai He*    Ambuj K. Singh
Department of Computer Science
University of California, Santa Barbara

* Now at Google Inc.
Introduction

- Graphs occur in various domains
  - Chemical compounds, protein structures and interactions, social networks, semantic web, database schemas, etc

- Graphs-at-a-time queries
  - Chemical substructures
  - Protein structural motifs
  - Protein complexes
  - Co-authorship graphs from DBLP

- Motivation 1: need a language to query and manipulate graphs with arbitrary attributes and structures
Why not Relational Model?

Motivation 2: native access methods that exploit graph structural information

query

Database graph

```
SELECT V1.vid, V2.vid, V3.vid
FROM V AS V1, V AS V2, V AS V3,
     E AS E1, E AS E2, E AS E3
WHERE V1.label = 'A' AND V2.label = 'B' AND V3.label = 'C'
    AND V1.vid = E1.vid1 AND V1.vid = E3.vid1
    AND V2.vid = E1.vid2 AND V2.vid = E2.vid1
    AND V3.vid = E2.vid2 AND V3.vid = E3.vid2
    AND V1.vid <> V2.vid AND V1.vid <> V3.vid
    AND V2.vid <> V3.vid;
```

Graph structural information is lost 😞

Table V (vid, label)
Table E (vid1, vid2)
Outline

- Formal language for graphs
- Graph query language
- Access methods
- Experimental study
- Related work
- Conclusion
Formal Language for Graphs

- Goals:
  - Notion for manipulating graph structures
  - Basis of graph query language

- Simple graph motifs

```plaintext
graph G_1 {
    node v_1, v_2, v_3;
    edge e_1 (v_1, v_2);
    edge e_2 (v_2, v_3);
    edge e_3 (v_3, v_1);
}
```
Complex Graph Motifs

- Concatenation by edges

```graph G2 {
  graph G1 as X;
  graph G1 as Y;
  edge e4 (X.v1, Y.v1);
  edge e5 (X.v3, Y.v2);
}
```

- Concatenation by unification

```graph G3 {
  graph G1 as X;
  graph G1 as Y;
  unify X.v1, Y.v1;
  unify X.v3, Y.v2;
}
```
Complex Graph Motifs (2)

Disjunction

```
graph G_4 {
  node v_1, v_2;
  edge e_1 (v_1, v_2);
  {
    node v_3;
    edge e_2 (v_1, v_3);
    edge e_3 (v_2, v_3);
  } |
  {
    node v_3, v_4;
    edge e_2 (v_1, v_3);
    edge e_3 (v_2, v_4);
    edge e_4 (v_3, v_4);
  };
}
```
Complex Graph Motifs (3)

- Repetition

```
graph Path {
    graph Path;
    node v_1;
    edge e_1 (v_1, Path.v_1);
    export Path.v_2 as v_2;
}

graph Cycle {
    graph Path;
    edge e_1 (Path.v_1, Path.v_2);
}
```

Path and cycle

```
graph G_5 {
    graph G_5;
    graph G_1;
    export G_5.v_0 as v_0;
    edge e_1 (v_0, G_1.v_1);
}
```

Repetition of G_1
Outline

- Formal language for graphs
- **Graph query language (GraphQL)**
  - Graph pattern and matching
  - Graph Algebra
  - FLWR expressions
- Access methods
- Experimental study
- Related work
- Conclusion
Graph Query Language

- **Graph**
  - = Simple graph motif + annotated attributes

- **Graph pattern**
  - = Graph motif + predicate on attributes

- **Graph pattern matching**
  - = Subgraph isomorphism + predicate evaluation

```
graph G <inproceedings> {
  node v_1 <title="Title1", year=2006>;
  node v_2 <author name="A">;
  node v_3 <author name="B">;
}

graph P {
  node v_1;
  node v_2;
  } where v_1.name="A"
  and v_2.year>2000;
```

(a) Graph          (b) Graph pattern          (c) Matching
Graph Algebra

- Relational algebra (RA)
  - Tuples are the basic unit
  - Algebraic operators:
    - Selection, Cartesian product and join, Projection, Set operators

- Graph algebra
  - Graphs are the basic unit
  - Selection: graph pattern matching
  - Composition: compose new graph structures and attributes

- Expressive power
  - RA ⊆ GraphQL
  - GraphQL ⊆ Datalog
  - Nonrecursive GraphQL ≡ RA
Selection $\sigma$

- **Input:**
  - Graph pattern $P$
  - A collection of graphs $C$

- **Output:**
  - A collection of matched graphs

$$\sigma_P(C) = \{ \phi_P(G) \mid G \in C \}$$
Cartesian Product and Join

- **Input:**
  - Two collections of graphs $C$ and $D$

- **Output:**

\[ C \times D = \{ \text{graph} \{ \text{graph} \ G_1, G_2; \} \mid G_1 \in C, \ G_2 \in D \} \]

- **Join:**
  - Product + Selection

\[ C \bowtie_P D = \sigma_P (C \times D) \]
Composition

- Graph template and instantiation

\[
T_P = \text{graph} \{
\text{node } v_1 <\text{label}=P.v_1.\text{name}>;
\text{node } v_2 <\text{label}=P.v_2.\text{title}>;
\text{edge } e_1(v_1, v_2);
\}\ 
\]

\[
T_P(G) = \text{graph} \{
\text{node } v_1 <\text{label}="A">;
\text{node } v_2 <\text{label}="Title1">;
\text{edge } e_1(v_1, v_2);
\}\ 
\]

(a) Graph template  (b) Instantiation

- Composition

\[
\omega_{T_P}(C) = \{ T_P(G) \mid G \in C \} \quad \text{(primitive composition)}
\]

\[
\omega_{T_{P_1}, P_2}(C_1, C_2) = \omega_{T_P}(C_1 \times C_2),
\]

where \( P = \text{graph} \{ \text{graph } P_1, P_2; \} \).
Example (FLWR Expressions)

This query generates a co-authorship graph from DBLP

\[
C = \sigma_J (\omega_{\tau_{P,C}} (\sigma_P ("DBLP"), \{C\}))
\]
Outline

- Formal language for graphs
- Graph query language
- **Access methods**
  - Graph pattern matching
  - Local pruning and retrieval of feasible mates
  - Joint reduction of search space
  - Optimization of search order
- Experimental study
- Related work
- Conclusion
Graph Pattern Matching

- **Basic algorithm**
  
  
  for A in \{A_1,A_2\}
  
  for B in \{B_1,B_2\}
  
  for C in \{C_1,C_2\}
  
  \[...\]

- **Feasible mates:**
  
  \{A_1,A_2\} of A

- **Search space:**
  
  \{A_1,A_2\}X\{B_1,B_2\}X\{C_1,C_2\}

- **Search order:**
  
  A→B→C
1. Local Pruning and Retrieval of Feasible Mates

![Diagram of graphs P and G with nodes A, B, C, A_1, B_1, C_1, A_2, B_2, C_2]

<table>
<thead>
<tr>
<th>Nodes of G</th>
<th>Neighborhood subgraphs of radius 1</th>
<th>Profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_1</td>
<td>A_1</td>
<td>ABC</td>
</tr>
<tr>
<td>B_1</td>
<td>B_1</td>
<td></td>
</tr>
<tr>
<td>C_2</td>
<td>C_2</td>
<td></td>
</tr>
<tr>
<td>A_2</td>
<td>A_2</td>
<td>AB</td>
</tr>
<tr>
<td>B_2</td>
<td>B_2</td>
<td></td>
</tr>
<tr>
<td>C_2</td>
<td>C_2</td>
<td></td>
</tr>
<tr>
<td>B_1</td>
<td>B_1</td>
<td>ABCC</td>
</tr>
<tr>
<td>A_1</td>
<td>A_1</td>
<td></td>
</tr>
<tr>
<td>C_1</td>
<td>C_1</td>
<td></td>
</tr>
<tr>
<td>B_2</td>
<td>B_2</td>
<td>ABC</td>
</tr>
<tr>
<td>C_1</td>
<td>C_1</td>
<td>BC</td>
</tr>
<tr>
<td>C_2</td>
<td>C_2</td>
<td>ABBC</td>
</tr>
</tbody>
</table>

**Search space**
- Retrieve by nodes: \(\{A_1, A_2\} \times \{B_1, B_2\} \times \{C_1, C_2\}\)
- Retrieve by neighborhood subgraphs: \(\{A_1\} \times \{B_1\} \times \{C_2\}\)
- Retrieve by profiles of neighborhood subgraphs: \(\{A_1\} \times \{B_1, B_2\} \times \{C_2\}\)
2. Joint Reduction of Search Space

- Refinement procedure of pseudo subgraph isomorphism (previous work in C-tree)
3. Optimization of Search Order

- **Cost model**

  ![Diagram showing search space and cost calculations for different orders: (A \times B) \times C and (A \times C) \times B]

  **Input search space:**
  \{A_1\} \times \{B_1,B_2\} \times \{C_2\}

  **Cost calculations:**

  - Case (a): \( (A \times B) \times C \)
    - Cost(J1) = 1 \times 2 = 2
    - Size(J1) = 2\gamma
    - Cost(J2) = 2\gamma
    - Cost(J1+J2) = 2 + 2\gamma

  - Case (b): \( (A \times C) \times B \)
    - Cost(J1) = 1 \times 1 = 1
    - Size(J1) = \gamma
    - Cost(J2) = 2\gamma
    - Cost(J1+J2) = 1 + 2\gamma

  \( (A \times B) \times C \) is worse than \( (A \times C) \times B \)
Outline

- Formal language for graphs
- Graph query language
- Access methods
- Experimental study
- Related work
- Conclusion
Experimental Study

- **Graphs**
  - Yeast network (3112 nodes, 12519 edges, 183 GO terms)
  - Synthetic graphs (n=10K~320K, m=5n, 100 labels)

- **Queries**
  - Cliques (yeast network)
  - Paths (yeast network)
  - General graphs (synthetic)

- **Metrics**
  - Reduction of search space
  - Running time
  - Compare with SQL-based implementation
Clique Queries

- **Reduction of search space**

Reduction ratio:

\[
\gamma(\Phi, \Phi_0) = \frac{|\Phi(u_1)| \times \cdots \times |\Phi(u_k)|}{|\Phi_0(u_0)| \times \cdots \times |\Phi_0(u_k)|}
\]
Running time (low hits)

(a) Individual steps

(b) Total query processing

Optimized: Retrieve by profiles + Refine + Optimize order + Search w/ opt. order
Baseline: Retrieve by nodes + Search w/o opt. order
Path Queries

(a) Low hits

(b) High hits
Path Queries (2)

(a) Individual steps  
(b) Total query processing
Synthetic Graphs

- Individual steps

(a) Search space

(b) Time for individual steps

n=10K, m=5n
100 labels, Zipf’s distribution

Search times are ignorable after refinement
Synthetic Graphs (2)

- Total running time

(a) Varying query sizes
n=10K

(b) Varying graph sizes
|query|=4

“SQL-based” does not scale for large queries!
Related work

- **Graph query languages**
  - | Language               | Basic unit | Query style   | Semistructured |
  - |-----------------------|------------|---------------|----------------|
  - | GraphQL               | graphs     | set-oriented  | yes            |
  - | SQL                   | tuples     | set-oriented  | no             |
  - | TAX                   | trees      | set-oriented  | yes            |
  - | GraphLog              | nodes/edges| logic pro.    | -              |
  - | OODB (GOOD, GraphDB, GOQL) | nodes/edges | navigational | no             |

- **Graph indexing**
  - GraphGrep, gIndex, C-tree, etc
  - Useful for accessing a large collection of small graphs

- **Reachability queries**
  - Useful for recursive graph patterns
Take-Home Message

- A graph algebra extended from the relational algebra where graphs are the basic unit

- Access methods for the selection operator over large graphs that exploit local and global graph structural information
Questions?