A Brief Introduction to Graph Databases

Olaf Hartig
olaf.hartig@liu.se
Theoretical Foundations
Foundations of Graph Databases

- Topic of research since at least 30 years
- Typical questions of interest:
  - Expressiveness
  - Complexity of evaluation
  - Containment problem
Expressiveness

• Let $L_1$ and $L_2$ be query languages
• $L_1$ is at least as expressive as $L_2$ if for every query in $L_2$, there exists a semantically equivalent query in $L_1$
• $L_1$ is strictly more expressive than $L_2$ if $L_1$ is at least as expressive as $L_2$ and there exists a query in $L_1$ for which there does not exist a semantically equivalent query in $L_2$
Complexity of Evaluation

• Let $L$ be a query language
• $L$-EVAL: Given a graph database $G$, a query $Q$ in $L$, and a result element $\mu$ of the right type for $L$, does $\mu$ belong to the result of $Q$ over $G$?
  - Combined complexity
Complexity of Evaluation (cont.)

- EVAL(Q) for a fixed query Q in L: Given a graph database G and a result element μ of the right type for L, does μ belong to the result of Q over G?
- Let C be a complexity class
- If for every query Q in L, the problem EVAL(Q) is in C, then L-EVAL is in C in data complexity
- L-EVAL is C-hard in data complexity if there exists a query Q in L such that EVAL(Q) is C-hard
- L-EVAL is C-complete in data complexity if a) it is in C and b) it is C-hard in data complexity
Containment Problem

- Let $L$ be a query language
- $L$-CONT: Given two queries $Q$ and $Q'$ in $L$, is the result of $Q$ over $G$ a subset of the result of $Q'$ over $G$ for every graph database $G$?
Data Model

• Prevalent data model: directed, edge-labeled graph
  – Given a finite alphabet $\Sigma$, a graph database over $\Sigma$ is a pair $(V,E)$ where $V$ is a finite set of node ids and $E$ is a subset of $V \times \Sigma \times V$

• A *path* is a sequence $\rho = v_0 a_0 v_1 a_1 \ldots v_{k-1} a_{k-1} v_k$ such that $(v_{i-1}, a_{i-1}, v_i)$ in $E$ for each $i$ in $\{1, \ldots, k\}$

• The *label* of $\rho$ is the string $a_0 a_1 \ldots a_{k-1}$
  – Label of the empty path $v$ is the empty string

• A path is *simple* if it does not go through the same node twice
Types of Queries

- Conjunctive queries (subgraph matching)
  \[ \text{ans}(z_1, \ldots, z_n) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, a_i, y_i) \]

- Regular path queries (RPQs)
  \[ \text{ans}(x, y) \leftarrow (x, r, y) \]

- Conjuctive RPQs (CRPQs)
  \[ \text{ans}(z_1, \ldots, z_n) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, r_i, y_i) \]

- RPQs with inverse (2RPQ) and C2RPQ
- RPQs with label variables (RPQVs)
- Unions of C2RPQs
- RPQs with nested regular expression
- Extended CRPQs
Graph Data Systems and their Data Models
Categories of Graph Data Systems

- **Triple stores**
  - Typically, pattern matching queries and inferencing
  - Data model: RDF

- **Graph databases**
  - Typically, navigational queries
  - Prevalent data model: property graphs

- **Graph processing systems**
  - Typically, complex graph analysis tasks
  - Prevalent data model: generic graphs

- **Graph dataflow systems**
  - Typically, complex graph analysis tasks in combination with general dataflow tasks
  - Prevalent data model: generic graphs
Examples of Graph DB Systems

• System that focus on graph databases
  – Neo4j
  – Sparksee
  – Titan
  – InfiniteGraph

• Multi-model NoSQL stores with support for graphs:
  – OrientDB
  – ArangoDB

• Triple stores with TinkerPop support
  – Blazegraph
  – Stardog
  – IBM System G
Property Graph

1. person
   name: marko
   age: 29
   knows
   weight: 0.5
   7
   2. person
   name: vadas
   age: 27
   created
   9
   weight: 0.4

3. software
   created
   9
   weight: 0.4
   11
   4. person
   name: josh
   age: 32
   created
   11
   weight: 1.0

5. software
   name: ripple
   lang: java
   created
   10
   6. person
   name: peter
   age: 35
   created
   12
   weight: 0.2
Property Graph (cont'd)

- Directed multigraph
  - multiple edges between the same pair of nodes
- Any node and any edge may have a label
- Additionally, any node and any edge may have an arbitrary set of key-value pairs ("properties")
Gremlin Graph Traversal Language

• Part of the Apache TinkerPop framework
• Powerful domain-specific language (DSL) for which embeddings in various programming languages exist
• Expressions specify a concatenation of traversal steps
Gremlin Example

g.V().has('name','marko').out('knows').values('name')

Result:
==>vadas
==>josh
Gremlin Example

g.V().has('name','marko').out('knows').values('name').path()

Result:
==->[v[1],v[2],vadas]
==->[v[1],v[4],josh]
Cypher

- Declarative graph database query language
- Proprietary (used by Neo4j)
- The OpenCypher project aims to deliver an open specification

- Example
  - Recall our initial Gremlin example:
    
    ```
    g.V().has('name','marko').out('knows').values('name')
    ```
  - In Cypher we could express this query as follows:
    
    ```
    MATCH ( {name:'marko'} )-[:knows]->( x )
    RETURN x.name
    ```
Categories of Graph Data Systems

- **Triple stores**
  - Typically, pattern matching queries and inferencing
  - Data model: RDF

- **Graph databases**
  - Typically, navigational queries
  - Prevalent data model: property graphs

- **Graph processing systems**
  - Typically, complex graph analysis tasks
  - Prevalent data model: generic graphs

- **Graph dataflow systems**
  - Typically, complex graph analysis tasks in combination with general dataflow tasks
  - Prevalent data model: generic graphs
Complex Graph Analysis Tasks???

- Tasks that require an *iterative processing* of the *entire graph* or large portions thereof
- Examples:
  - Centrality analysis (e.g., PageRank)
  - Clustering, connected components
  - Graph coloring
  - Diameter finding
  - All-pairs shortest path
  - Graph pattern mining (e.g., frequent subgraphs, community detection)
  - Machine learning (e.g., belief propagation, Gaussian non-negative matrix factorization)
Generic Graphs

• Data model
  - Directed multigraphs
  - Arbitrary user-defined data structure can be used as value of a vertex or an edge (e.g., a Java object)

• Example (Flink Gelly API):

```
// create new vertexes with a Long ID and a String value
Vertex<Long, String> v1 = new Vertex<Long, String>(1L, "foo");
Vertex<Long, String> v2 = new Vertex<Long, String>(2L, "bar");
Edge<Long, Double> e = new Edge<Long, Double>(1L, 2L, 0.5);
```

• Advantage: give users maximum flexibility
• Drawback: systems cannot provide built-in operators related to vertex data or edge data
Graph Processing Systems

Pregel Family
- Pregel
- Giraph
- Giraph++
- Mizan
- GPS
- Pregelix
- Pregel+

GraphLab Family
- GraphLab
- PowerGraph
- GraphChi (centralized)

Other Systems
- Trinity
- TurboGraph (centralized)
- Signal/Collect
Vertex-Centric Abstraction

- Many such algorithms iteratively propagate data along the graph structure by transforming intermediate vertex and edge values
  - These transformations are defined in terms of functions on the values of adjacent vertexes and edges
  - Hence, such algorithms can be expressed by specifying a function that can be applied to any vertex separately

- “Think like a vertex”
Vertex-Centric Abstraction (cont'd)

- Vertex compute function consists of three steps:
  1. Read all incoming messages from neighbors
  2. Update the value of the vertex
  3. Send messages to neighbors
- Additionally, function may “vote to halt” if a local convergence criterion is met

- Overall execution can be parallelized
  - Terminates when all vertexes have halted and no messages in transit
Categories of Graph Data Systems

- **Triple stores**
  - Typically, pattern matching queries and inferencing
  - Data model: RDF

- **Graph databases**
  - Typically, navigational queries
  - Prevalent data model: property graphs

- **Graph processing systems**
  - Typically, complex graph analysis tasks
  - Prevalent data model: generic graphs

- **Graph dataflow systems**
  - Typically, complex graph analysis tasks in combination with general dataflow tasks
  - Prevalent data model: generic graphs
Acknowledgements:

- Some of the slides about graph processing systems are from a slideset of Sherif Sakr. Thanks Sherif!

Image sources:

- Example Property Graph [http://tinkerpop.apache.org/docs/current/tutorials/getting-started/](http://tinkerpop.apache.org/docs/current/tutorials/getting-started/)
- Smiley [https://commons.wikimedia.org/wiki/File:Face-smile.svg](https://commons.wikimedia.org/wiki/File:Face-smile.svg)
- Frowny [https://commons.wikimedia.org/wiki/File:Face-sad.svg](https://commons.wikimedia.org/wiki/File:Face-sad.svg)