Graph Database System Neo4j

Hannes Voigt
Some Background
Adjacency Matrix

**Undirected Graph Without Labels**

**Directed Graph With Edge Labels**

[Diagram showing undirected and directed graphs with adjacency matrices]
Adjacency Lists

**Compression of Adjacency Matrix**
- Compression scheme:
  0 1 2 3
  0 \( a \) \( c \)
  1 \( b \) \( e \)
  2 \( d \) \( i \) \( f \)
  3 \( h \) \( g \)

**Adjacency List**
- Source vertex together outgoing edges
  - Without edge labels
    0 -> (0,2)
    1 -> (1,3)
    2 -> (0,1,2)
    3 -> (1,2)
  - With edge labels
    0 -> ([0,a],[2,c])
    1 -> ([1,b],[3,e])
    2 -> ([0,d],[1,i],[2,f])
    3 -> ([1,h],[2,g])
  - With edge properties
    0 -> ([0,a](weight=4)],[2,c](weight=3))
    1 -> ([1,b](weight=3)],[3,e](weight=2))
    2 -> ([0,d](weight=5)],[1,i](weight=2))
    3 -> ([1,h](weight=9)],[2,g](weight=7))

Compressed Sparse Row
- Source-oriented
- Without edge labels
- With edge labels

Compressed Sparse Column
- Target-oriented
- Without edge labels
- With edge labels

Coordinate list
- \((0,0,a)\)
- \((0,2,c)\)
- \((1,1,b)\)
- \((2,2,f)\)
- \((3,1,h)\)
- \((3,2,g)\)
**Basic Terminology**

**Walk**
- Sequence of edges connecting vertices
  - \( w_v = v_1e_1v_2 \ldots v_{n-1}e_{n-1}v_n \) with \( e_i = (v_i, v_{i+1}) \in E, 1 \leq i < n \)
  - \( n \) is the length of the walk

**Path**
- Walk connecting distinct vertices
  - \( p_v = v_1e_1v_2 \ldots v_{n-1}e_{n-1}v_n \) with \( \forall v_i, v_j: i \neq j \rightarrow v_i \neq v_j \) and \( e_i = (v_i, v_{i+1}) \in E, 1 \leq i < n \)
  - Length is number of edges or sum of edge weights

**Cycle**
- Walk with \( v_1 = v_n \) is a cycle
- Graph is acyclic iff \( \forall v \in V: w_{v,v} \in G \)

**Degree (or Valency)**
- In/out degree of a vertex: Number of incoming/outgoing edges of that vertex
  - \( \deg_{\text{out}}(v) = |\{(v, u) \in E\}| \)
  - \( \deg_{\text{in}}(v) = |\{(u, v) \in E\}| \)
  - \( \deg(v) = \deg_{\text{out}}(v) + \deg_{\text{in}}(v) \)

**Distance**
- Distance between two vertices in a graph is number of edges in a shortest path connecting them
  - \( d(v, u) = \min_{p_v,u \in G} |p_v,u| \)

**Diameter**
- Maximum eccentricity of any vertex in the graph
  - \( d(G) = \max_{v \in V} \epsilon(v) \)
  - \( d(G) = 4 \)
Finding Shortest Paths

**UNWEIGHTED SHORTEST PATHS**
- Length of path its number of edges
- Restriction to simple paths (w/o cycles)

**Two main ways of path search**
- Depth-first search (DFS)
- Breadth-first search (BFS)

**Depth-first search (DFS)**
- Search tree is deepened as much as possible on each child before going to the next sibling
- Lower space complexity
- Has to examine whole graph to find shortest path between two nodes

**Breadth-first search (BFS)**
- Search tree is broadened as much as possible on each depth before going to the next depth
- Potential large space required
- Find shortest path between two nodes first (before finding a longer one)
**Bidirectional BFS**

**IDEA: SEARCH FROM START AND END VERTEX**

- Alternatingly explore vertices on both sides
  - Optimization: explore vertices on the side with smaller frontier
- Algorithm stops when both BFS meet
  - When discovering a new vertex, each BFS check if that vertex is in frontier of other side

Forward (598 vertices explored)  
Backward (860 vertices explored)  
Bidirectional (448 vertices explored)  

Centrality Measures

**QUESTION:** WHO ARE THE KEY PLAYERS IN A GRAPH

- Most social contacts (vaccination schedules)
- Most influential thinkers/papers (reading lists)
- Most important website (web search)
- Most important distributors (supply network)
- etc.

- Can we measure that?

**YES! WITH CENTRALITY MEASURES!**

- Centrality measures identify the most important vertices within a graph

[http://brendangriﬃen.com/blog/gow-inﬂuential-thinkers]
Centrality Measures

Various Centrality Measures Have Been Defined

- **Betweenness centrality (A)**
  - Number of shortest paths between all other vertices that pass through that vertex

- **Closeness centrality (B)**
  - Reciprocal of the sum of distances to all other vertices
  - Harmonic centrality (E) uses the sum of reciprocal of distances instead

- **Eigenvector centrality/Eigencentrality (C)**
  - Score of a vertex contributes to score of neighboring vertices
  - Page rank is variant of eigenvector centrality

- **Katz centrality (F)**
  - Number of all vertices that can be connected through a path
  - Contributions of distant nodes are penalized
  - Degree centrality (D) only considers direct neighbors

[https://commons.wikimedia.org/wiki/File:6_centrality_measures.png]
Neo4j
Neo4j Terminology

Match

MATCH-CLAUSE
- Primary way of getting data from a Neo4j database
- Allows you to specify the patterns
- Named pattern element, e.g. (p:Person), will be bound to the match instance
- Query can have multiple MATCH-clauses

RETURN-CLAUSE
- Projects to the result set
- Allows projection to nodes, edges, and properties

MATCH (p:Person)-[:Likes]->(f:Person)
RETURN p.name, f.sex

MATCH (p:Person)-[:Likes]->(:Person)-[:Likes]->(fof:Person)
RETURN p.name, fof.name
Pattern Syntax


**VERTEX PATTERN**

- ()
- (matrix)
- (:Movie)
- (matrix:Movie:Action)
- (matrix:Movie {title: "The Matrix"})

unidentified vertex
vertex identified by variable *matrix*
unidentified vertex with label *Movie*
vertex with labels *Movie* and *Action* identified by variable *matrix*
+ property *title* equal the string “The Matrix”
+ property *released* equal the integer 1997

**EDGE PATTERN**

- -->
- -[role]->
- -[:ACTED_IN]->
- -[role:ACTED_IN]->
- -[role:ACTED_IN {roles: ["Neo"]}]->

unidentified edge
edge identified by variable *role*
unidentified edge with label *ACTED_IN*
edge with label *ACTED_IN* identified by variable *role*
+ property *roles* contains the string “Neo”
Pattern Syntax


**PATH PATTERNS**
- String of alternating vertex pattern and edge pattern
- Starting and ending with a vertex pattern
  - (a)-->(b)<--(c)<--(d)<--(a)<--(e)
  - (keanu:Person:Actor {name: "Keanu Reeves"})-[role:ACTED_IN {roles: ["Neo"]}]-> (matrix:Movie {title: "The Matrix"})

**GRAPH PATTERNS**
- One or multiple path patterns
- Path patterns should have at least one shared variable
- Without shared variable graph pattern is disconnected
  - Results in a cross-product of the results for connected sub patterns
  - Quadrating blow up in result size and computational complexity
- (a)-->(b)<--(c)<--(d)<--(a)<--(e), (e)<--(b)<--(d), (a)<--(a)
**Return**

RETURN


![](http://neo4j.com/docs/developer-manual/current/#cypher-expressions)

**RETURN-CLAUSE**

- Defines what to include in the query result set
- Comparable with relational projection
- Only once per query
- Allows to return nodes, edges, properties, or any expressions
- Column can be rename using AS <new name>

**EXAMPLE**

```
MATCH (n)
RETURN n, "node " + id(n) +" is " +
    CASE
        WHEN n.title IS NOT NULL THEN "a Movie"
        WHEN EXISTS(n.name) THEN "a Person"
        ELSE "something unknown"
    END AS about
```

Returned 174 rows in 46 ms.
Optional Match & Where


**OPTIONAL MATCH-CLAUSE**
- Matches patterns against your graph database, just like MATCH
- Matches the complete pattern or not
- If no matches are found, OPTIONAL MATCH will use NULLs as bindings
- Like relational outer join
- Example: MATCH (a:Movie)
  OPTIONAL MATCH (a)<-[[:WROTE]]-(x)
  RETURN a.title, x.name

**WHERE**
- After an (OPTIONAL) MATCH, it adds constraints to the (optional) match
- WHERE becomes part of the pattern
- After a WITH, it just filters the result
- Syntax: WHERE <expression>
- Example: MATCH (n) WHERE n.name = 'Peter' XOR (n.age < 30 AND n.name = 'Tobias')
  OR NOT (n.name =~ 'Tob.*' OR n.name CONTAINS 'ete')
  RETURN n
Matching Paths

VARIABLE LENGTH PATH PATTERNS

- Repetitive edge types can be expressed by specifying a length with lower and upper bounds
- Example: \((a)-[:x*2]-(b)\) is equal to \((a)-[:x]-(b)\)
- More examples:
  - \((a)-[*3..5]-(b)\)
  - \((a)-[*3..]-(b)\)
  - \((a)-[*..5]-(b)\)
  - \((a)-[*]-(b)\)
- Complete example: MATCH (me)-[:KNOWS*1..2]-(remote_friend)
  WHERE me.name = "Filipa" RETURN remote_friend.name

- Matches unique paths (relationship uniqueness), not unique reachable nodes!!
- Particularly the unbounded [*] easily matches larger numbers of paths -> exponential blowup!!

PATH VARIABLES

- Assign matched paths to variable or further processing
- Example: \(p = ((a)-[*3..5]-(b))\)
Matching Shortest Paths

SHORTEST PATHS

- Path between two nodes with minimum number of edges
- Apply the shortestPath/allShortestPath function to a path pattern to match single/all shortest paths
- Additional filter predicates can be given with WHERE clause
  - Universal (NONE/ALL) predicates can be evaluated during shortest path search
  - Other predication can be evaluated only after shortest path has been discovered
- Fast evaluation algorithm
  - Bidirectional BFS
  - Standard for paths without additional predicates and path with universal predicates
- Slow evaluation algorithm
  - DFS
  - Fallback for paths with non-universal predicates
- Example (fast evaluation):
  ```
  MATCH (m { name: "Martin Sheen" }), (o { name: "Oliver Stone" }), p = shortestPath((m)-[*..15]-(o))
  WHERE NONE(r IN rels(p) WHERE type(r) = "FATHER")
  RETURN p
  ```
- Example (fast evaluation):
  ```
  MATCH (m { name: "Martin Sheen" }), (o { name: "Oliver Stone" }), p = shortestPath((m)-[*..15]-(o))
  WHERE length(p) > 1
  RETURN p
  ```

Max. 15 hops
Aggregation

In RETURN-clause

- Implicit group by
  - Expressions without an aggregation function will grouping keys
  - Expressions with an aggregation function will produce aggregates
- DISTINCT within the aggregation function removes duplicates in a group before the aggregation
- Aggregation function: COUNT, SUM, AVG, MIN, MAX, STDEV, STDEVP, PERCENTILEDISC, PERCENTILECONT, and COLLECT – collects all the values into a list

In WITH-clause

- Like a process pipe
- Chains query parts together, piping the results from one to be used as starting points in the next
- Like RETURN, WITH defines – including aggregation – the output before it is passed on
- Allows to
  - Filter on aggregates
  - Aggregation of aggregates
  - Limit search space based on order of properties or aggregates

Aggregation

**IN RETURN-CLAUSE**

- Implicit group by
  - Expressions without an aggregation function will be group keys
  - Expressions with an aggregation function will produce aggregates
- DISTINCT within the aggregation function removes duplicates in a group before the aggregation
- Aggregation function: COUNT, SUM, AVG, MIN, MAX, STDEV, STDEVP, PERCENTILEDISC, PERCENTILECONT, and COLLECT – collects all the values into a list
- Example: MATCH (me:Person {name: 'Ann'})-->(friend:Person)-->(friend_of_friend:Person) RETURN me.name, count(DISTINCT friend_of_friend), count(friend_of_friend)

**IN WITH-CLAUSE**

- See next slide

**Result**

<table>
<thead>
<tr>
<th>me</th>
<th>COUNT DISTINCT</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ann</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Query Composition


**WITH-CLAUSE**

- Like a process pipe
- Chains query parts together, piping the results from one to be used as starting points in the next
- Like RETURN, WITH defines – including aggregation – the output before it is passed on

- Filter on aggregates
  Example: Soccer team on average younger than 25
  ```sql
  MATCH (p)-[:PLAYS]->(t) WITH t, AVG(p.age) AS a WHERE a < 25 RETURN t
  ```

- Aggregation of aggregates
  Example: Average age of the youngest player in each team
  ```sql
  MATCH (p)-[:PLAYS]->(t) WITH t, MIN(p.age) AS a RETURN AVG(a)
  ```

- Limit search space based on order of properties or aggregates
  Example: Friends of five best friends
  ```sql
  MATCH (p)-[f:FRIENDS]->(p2) WITH f,p2 ORDER BY f.rating DESC LIMIT 5
  MATCH (p2)-[f:FRIENDS]->(p3) RETURN DISTINCT p3
  ```
Exercise
Exercise

PREPARATION

- Download and install neo4j community edition:
  - For installation follow standard download procedure: [http://neo4j.com/download/](http://neo4j.com/download/)
  - For portable usage without installation download archive (tar/zip): [https://neo4j.com/download/other-releases/](https://neo4j.com/download/other-releases/)
  and follow OS-specific installing instructions at download page

- Import the movie database > :play movie graph > 2\(^{nd}\) page > click on code > execute

- Try out query: MATCH (n) WITH COUNT(n) AS numVertices
  MATCH (a)-[e]->(b)
  RETURN numVertices, COUNT(e) AS numEdges

- Try out query: MATCH (n) RETURN n

<table>
<thead>
<tr>
<th>Expected Result</th>
<th>numVertices</th>
<th>numEdges</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>171</td>
<td>253</td>
</tr>
</tbody>
</table>

ADD DATA

- Add movie, actor (three main characters), director as vertices and ACTED_IN/DIRECTED edges for the movie *The Bridges of Madison County* [http://www.imdb.com/title/tt0112579/](http://www.imdb.com/title/tt0112579/)
- Do not insert vertices that already exist in the database!!!
Exercise

**Simple Patterns**

- Find all actors that directed a movie they also acted in and return actor and movie nodes
- Find all reviewer pairs, one following the other and both reviewing the same movie, and return entire subgraphs
- Find all reviewer pairs, one following the other, and return the two reviewers and a movie the may have reviewed both
- Restrict previous query so that the name of the followed reviewer is not 12 characters long
  - Try a different position for the where clause. Explain why this gives a different result.
- Find all actors that acted in a movie together after 2010 and return the actor names and movie node
- By extending the previous query, find all movies that the cast of the movies found before also acted in

**Matching Semantics of Neo4j**

- Which matching semantics does Neo4j implement? Homomorphism, Isomorphism, Induced subgraph isomorphism?
- Remove duplicates for pattern (x)- (y)
- Match pattern (a1)-[:REVIEWED] ->(m)<[:REVIEWED]-(a2) as induced subgraph
- Find all actor pairs that acted in multiple movies together
- Find all pairs of actor-movie subgraphs with equal roles (on ACTS_IN edges), return actors names, roles, and movie titles
Exercise

**Paths**

- Match all reviewers and the one they are following directly or via another a third reviewer
- Count the number of paths of at most length 4 starting from *Clint Eastwood* ignoring edge direction
- Count the number of paths of at most length 10 starting from *Clint Eastwood* ignoring edge direction
- Count the number of paths of at most length 11 starting from *Clint Eastwood* ignoring edge direction
- Count the number of nodes reachable in at most 4 hops starting from *Clint Eastwood* ignoring edge direction
- Count the number of nodes reachable in at most 10 hops starting from *Clint Eastwood* ignoring edge direction
- Count the number of nodes reachable in at most 11 hops starting from *Clint Eastwood* ignoring edge direction
Exercise

**Young and Old Movies**

- Determine the average age of the Apollo 13 cast at the time of the movie’s release
- Find the movies with the top-10 oldest cast at the time of the movie’s release
  - Return movie and average age rounded to two decimal ordered by descending age
- Find average age of youngest actors in movie casts at time of release
- Find ACTED_IN subgraph of the movie with the youngest cast at the time of the movie’s release
- Determine the movie with youngest and movie with oldest cast and their age difference rounded to two decimal points

**Adjacency List and Distributions**

- Return the whole graph a simple adjacency list of vertex ids ordered by decreasing vertex degree
- Return out degree distribution ordered by ascending degree
- Return degree distribution ordered by ascending degree
- Return edge types with number of instances order by decreasing instances number
Exercise

- Determine the Bacon number of Clint Eastwood
- Count for each Bacon number the number of actor
  - Return degree and number of actors ordered by ascending degree

- Find actors with top 10 Katz centrality along ACTED_IN edges
  - Distance penalty is reciprocal of path length (e.g. 3-hop neighbor gets a penalty of 1/3)
  - Return actor vertex and Katz centrality
Exercise

Hints

- Use your preferred search engine
- Try out! Explorer! Have fun!!!