Graph Analytics with Neo4J

Team 4: Emily Strong and Raksha Kaverappa

GitHub Repository: <https://github.com/erstrong/INFO-7390-ADS-Fall-17-TeamNo.4/tree/master/Team%20Presentation>

**Introduction**

Social network graphs represent the relationships and interactions between people or things. Analysis of these graphs has a variety of applications from social networking sites to fraud detection and national security. The graphs can be clustered with KMeans methods, and methods of analysis include examining the centrality of a node, the density of connections, and the shortest path between a pair of nodes, as well as many other graph algorithms.

Relationships can be directed or undirected, the nodes can be of different types, and both the nodes and the relationships can have different properties. For example:

**Owns**

**Since: 2015**

For our presentation we worked with the graph database Neo4J. It is graph-data native, with directional relationships that uses the Cypher language. Cypther is similar to SQL and uses ASCII symbols to represent relationships. For example:

CREATE (bob:Person {name:'Bob', age:30, color:'blue'})

CREATE (ruff:Dog {name:'Ruff', age: 2, like:'running})

CREATE (bob)-[:OWNS]->(ruff)

**Working with Neo4J GUI**

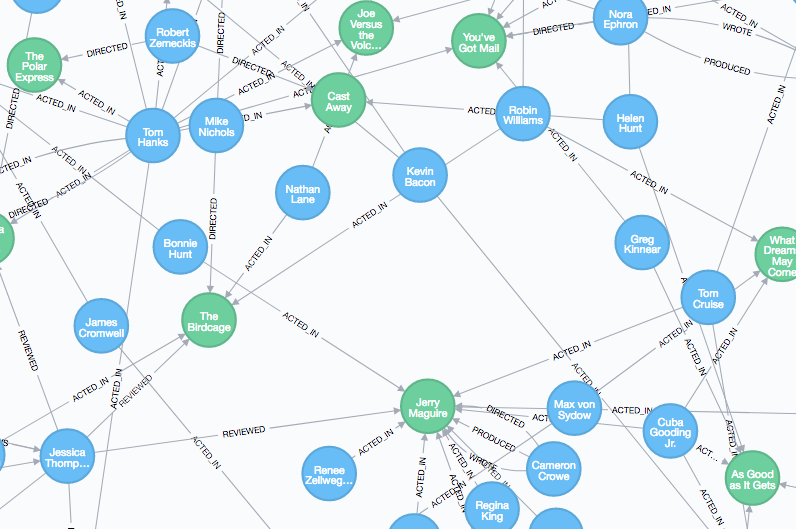
For our demo of using the Neo4J GUI we used the built-in tutorial set for movies and the Six Degrees to Kevin Bacon game. To generate the set:

1. Download the Community Edition from <https://neo4j.com/download/community-edition/> or use the online sandbox environment: <https://neo4j.com/sandbox-v2/?ref=product>. (Note: the second part of our demo requires having a local install)
   1. Once Neo4J is downloaded, follow the install prompts.
   2. Open the Neo4J application.
   3. When you launch the application for the first time you will have to sign in. The default password is neo4j and you will be asked to change it after you sign in.
   4. When the application is ready it will provide you with a local URL to access the interface
2. Run **:play movie-graph**
3. Click on the next arrow to get to the Create section of the tutorial
4. Click on the **play** icon next to the series of CREATE statements. This will add those statements to the command line at the top of the page.
5. Click on the command line **Play**
6. To view the entire graph database type in the below prompt and hit Play:

MATCH (n)-[r]->(m) RETURN n,r,m

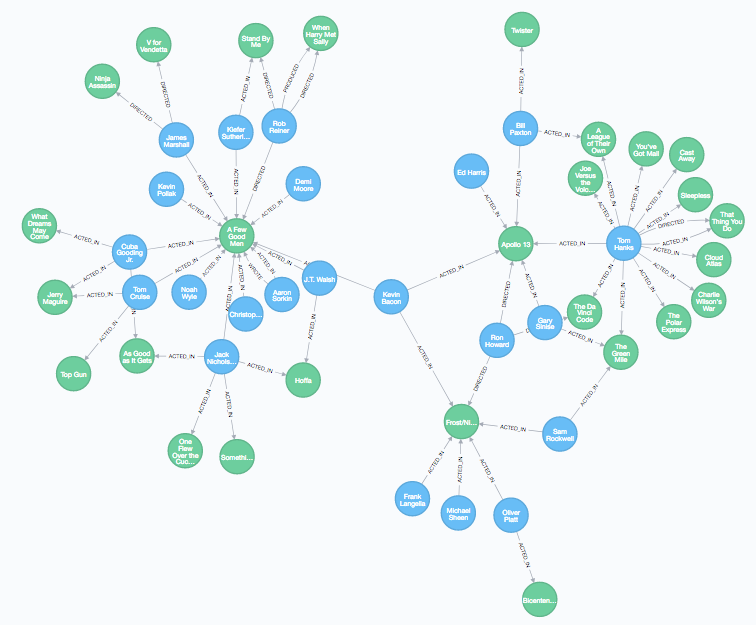
**The entire movie data set**

Zooming in, we can see that the data set involves people, various relationships (directed, acted in, produced, wrote, etc), and movies. It is centered around Kevin Bacon so that it can be used to find a solution to Six Degrees of Kevin Bacon:



The built-in tutorial provides an example of how to do that as well as a number of other querying methods in Neo4J. In our demo we performed a simplified Three Degrees to Kevin Bacon in the GUI and in Python. The query is:

MATCH (bacon:Person {name:"Kevin Bacon"})-[\*1..3]-(hollywood) RETURN DISTINCT bacon,hollywood



This returns a graph of all nodes with a shortest path of length 3 to Kevin Bacon.

**Working with Neo4J in Python**

We used the py2neo and bottle libraries to interact with our Neo4J database in Jupyter Notebook.

We connected to our local graph database as a graph object. Once the graph object is created, we extracted the entire data set using a get\_graph method that uses bottle's run method to execute a query and cleans up the formatting of the output. The returned object is a database cursor.

Example output for *Apollo 13*:

{'links': [{'source': 1, 'target': 0},

{'source': 2, 'target': 0},

{'source': 3, 'target': 0},

{'source': 4, 'target': 0},

{'source': 5, 'target': 0}],

'nodes': [{'label': 'movie', 'title': 'Apollo 13'},

{'label': 'actor', 'title': 'Tom Hanks'},

{'label': 'actor', 'title': 'Kevin Bacon'},

{'label': 'actor', 'title': 'Ed Harris'},

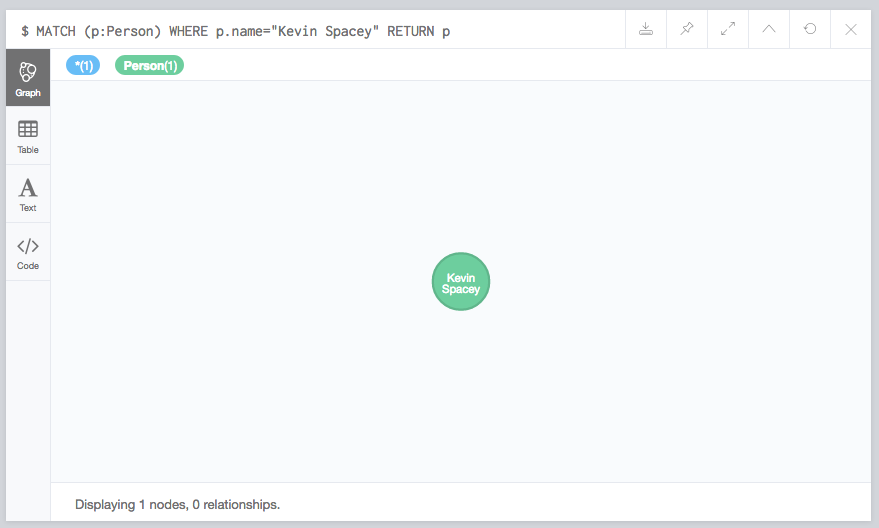
{'label': 'actor', 'title': 'Bill Paxton'},

{'label': 'actor', 'title': 'Gary Sinise'}]}

For our second example of interacting with the database, we ran our Three Degrees to Kevin Bacon query using the run method. We did not alter the format for this, and the nodes can be accessed as cursorname.data(). Since we did not perform any reformatting, it is in Cypher format:

{'hollywood': (ab1b9dc:Person {born:1958,name:"Kevin Bacon"})}

This could be reformatted into a data frame and analyses run on it in Python. Relationships have to be accessed separately as 'rel' objects in the cursor. There are also methods included in the py2neo library for performing simple graph analytics such as degree.

For our third example, we added a new node to the database from Jupyter Notebook representing Kevin Spacey. Returning to the Neo4J GUI we can see that the node was successfully created:  


However if we run Three Degrees of Kevin Spacey nothing is returned since a lone node is not a valid graph:

