# Neo4j tutorial

Jan Aerts

2016-04-20

## Preamble

This post is part of a collection to help students in the “Managing Large Omics Datasets” course at KU Leuven get a feel for how to work with graph databases. The database used as well as all tools are available as docker files (for more information, see my earlier post on Hadoop).

To get up and running, run this docker command: `docker run -d -p 7474:7474 jandot/neo4j-i0u19a`. Check if everything is working as it should by going to http://192.168.99.100:7474. That should open a webinterface to the database. Note that the URL might be different. If you’re working on linux, use `localhost` instead of `192.168.99.100`; otherwise, find the IP to use with `docker ip default`.

## The data

In this exercise, we will have a look at the gene and disease networks. Gene-gene interactions are downloaded from BioGRID; gene-disease interactions from DisGeNET. Gene-gene interaction data includes the 2 genes in question, as well as the proof (both details and number of proof); gene-disease associations include the gene and disease, a score and an association type.

Example data for gene-gene interactions:

```
#gene1,nr_proofs,proof,gene2
MAP2K4,2,"Two-hybrid;Reconstituted Complex",FLNC
MYPN,1,"Two-hybrid",ACTN2
ACVR1,1,"Two-hybrid",FNTA
```

Example data for gene-disease associations:

```
#gene,score,proofs,disease
ATP7B,0.97,AlteredExpression; GeneticVariation,umls:C0019202
MC4R,0.94,Biomarker; GeneticVariation,umls:C0028754
IRS1,0.91,Biomarker; GeneticVariation,umls:C0011860
```

## Setup of the tutorial

This tutorial consists of 2 parts. First, we'll do some exercises using the neo4j webinterface. Here we'll focus on the cypher query language. Next, we'll use the python API to neo4j to investigate the network a bit further.

### 1. Cypher

The cypher query language uses ascii-art to define what you're looking for. Nodes are denoted with parentheses (`()`; looks like a circle); relationships are denoted with an arrow (`-->`). So to find 2 nodes that are linked, you could use `()-->()`. Several good tutorials are available for cypher (e.g. this one and this one), as well as a cypher reference card.

```
START <lookup> MATCH <pattern> RETURN <expr>;
```

### 2. Python API

For some good background and tutorials on using neo4j, see the following links:

- Using neo4j from jupyter
- On py2neo
- On connecting to neo4j using py2neo

## Instructions on installation

- Install neo4j
- Edit the server configuration: comment out `dbms.security.auth_enabled=true`
- Install python-api
- sudo pip install py2neo
- sudo pip install neo4jrestclient
- Load data into neo4j
- Start neo4j
- Change password using neoauth

Example script:

```
from py2neo import Graph, Node, Relationship, authenticate
authenticate("localhost:7474","neo4j","neo4j")

secure_graph = Graph("http://localhost:7474/db/data/")
alice = Node("Person", name="Alice")
```

```
bob = Node("Person", name="Bob")
alice_knows_bob = Relationship(alice, "KNOWS", bob)
secure_graph.create(alice_knows_bob)
```

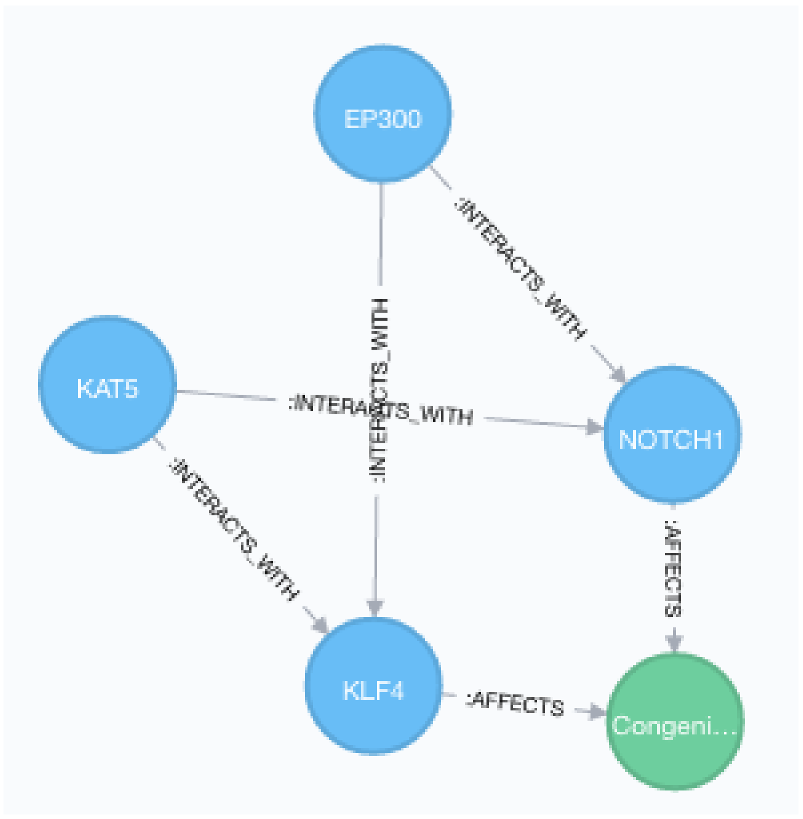

Figure 1: bi-fan

---

import csv import numpy import matplotlib.pyplot as plt from py2neo import Graph, Node, Relationship, authenticate %matplotlib inline

with open('regulatory_network.txt','r') as tsv: AoA = [line.strip().split('') for line in tsv]

first_column = map(lambda x:x[0].lower(), AoA) second_column = map(lambda x:x[1].lower(), AoA) both_columns = first_column + second_column gene_names = numpy.unique(both_columns)

authenticate("localhost:7474","neo4j","neo4j") secure_graph = Graph("http://localhost:7474/db/data/") secure_graph.delete_all nodes = {} for gene in gene_names: nodes[gene] = Node("Gene",name=gene)

for gene in gene_names: secure_graph.create(nodes[gene])

links = map(lambda x:[x[0].lower(),x[1].lower()],AoA) for link in links: source = nodes[link[0]] target = nodes[link[1]] secure_graph.create(Relationship(source, "AFFECTS", target))

len(secure_graph.cypher.execute("MATCH (a)-[:AFFECTS]->(b)-[:AFFECTS]->(c) RETURN b"))

## Get degree for each node

links_per_node = secure_graph.cypher.execute("MATCH (a)-[:AFFECTS]->(b) RETURN [a.name, COUNT(b)]") links_per_node = map(lambda lpn: lpn[0], links_per_node) # links_per_node

## What is maximal degree?

## sorted(links_per_node, key = lambda lpn: lpn[1])[-1] => [u'fis',233]

max_degree = sorted(links_per_node, key = lambda lpn: lpn[1])[-1][1] max_degree

x_values = range(0,(max_degree+1)) x_values y_values = [] for x in x_values: y_value = 0 for lpr in links_per_node: if lpr[1] == x: y_value += 1 y_values.append(y_value) y_values

plt.plot(x_values, y_values) plt.title("Degree distribution") plt.xlabel("Node degree") plt.ylabel("Number of nodes")