## **Alex Marshall**

## **HU Ext: E63 Big Data Analytics**

## **Neo4j in Linguistics**

## **Graphing Cycles of Sentence Similarity with neo4j in 2016 Presidential Campaign Speeches**

## 

**Problem Statement:**

I will use Neo4j to graph texts with sentence nodes connected by both temporal order and Bag-of-Words cosine similarity edges to draw insight on their structure. I will specifically compare recent campaign speeches from Hillary Clinton and Donald Trump. My hypothesis is that I will find relatively more short, higher similarity cycles in Donald Trump speeches and long, lower similarity cycles in Hillary Clinton speeches. This should correspond with the more repetitive and unscripted feel of Mr. Trump speeches and the more structured and focused feel of Ms. Clinton’s.

**Technology:**

Neo4j is a graph database optimized for connected data. It can traverse paths of connectedness much faster than RDBMSs and scales with little impact on performance. Natural language is one form of data that is usefully modeled with many connecting nodes. I will use Python with py2neo to split text into sentences and write to Neo4j where it can be analyzed and visualized. Specifically, I’m interested in modelling cycles between sentences of similar meaning.

**Overview of steps:**

1. Install and configure Neo4j, Anaconda and Python packages
2. Retrieve and prepare data by tokenizing into sentences and vectorizing those sentences
3. Write sentences to neo4j database and create temporal and cosine similarity connections between them.
4. Collect metrics on data in neo4j
5. Visualize parts of the speech graph

**Data**

Clinton Trump Corpus: <http://www.thegrammarlab.com/?nor-portfolio=corpus-of-presidential-speeches-cops-and-a-clintontrump-corpus>

The dataset consists of plain text documents of speeches delivered at campaign events by Hillary Clinton and Donald Trump. After unzipping, there are 37 Hillary Clinton speeches totaling 737 KB and 83 Donald Trump Speeches totaling 2.6 MB.

**Hardware:**

2014 MacBook Air running OS X

**Software:**

* Neo4j 3.0.5 ([neo4j.com/download/](https://neo4j.com/download/))
* Anaconda 4.3.30 ([www.anaconda.com/download/)](http://www.anaconda.com/download/))
  + Contains Python 3.6.3 and Jupyter Notebook 5.2.2
* Python Packages:
  + Scikit-learn ([scikit-learn.org/](http://scikit-learn.org/))
  + NLTK ([www.nltk.org/](http://www.nltk.org/))
  + Py2Neo ([py2neo.org/v3/](https://py2neo.org/v3/))
  + Numpy ([www.numpy.org/](http://www.numpy.org/))
  + Pandas ([pandas.pydata.org/](https://pandas.pydata.org/))
  + Scipy ([www.scipy.org/](http://www.scipy.org/))
  + Matplotlib (<https://matplotlib.org/>)

**Lessons Learned:**

* Donald Trump speeches contain many more similar sentences than Hillary Clinton’s
* This method is great at measuring repetition but not yet effective in finding themes or other structural patterns
* Multiple similar sentences, especially said in close proximity (ie “thank you”) add lots of noise to results

**Video Presentations:**

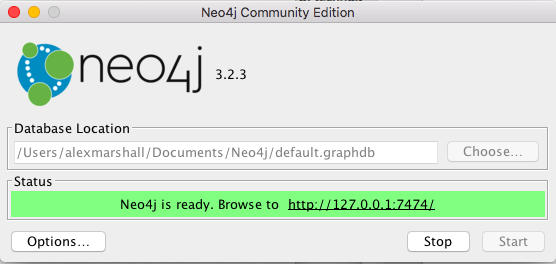
Short: <https://youtu.be/ZhZqxxkMmu0>

Long: <https://youtu.be/8xwWGrvepig>

**Detailed Steps & Demonstration:**

Installtion & Configuration

It’s easy to get started with neo4j. Go to <https://neo4j.com/download/?ref=home>, select your operating system and download the executable file. Then install, choose a database location and start a local server.



We will use Python along with neo4j. I worked within Jupyter notebook and used packages that are included in the Anaconda distribution of Python. [www.anaconda.com/download/](http://www.anaconda.com/download/)

After getting set up with neo4j and Anaconda, open Jupyter notebook and import the following packages: Nltk, Numpy, Pandas, Sklearn, Scipy Matplotlib, Py2neo.

My notebook looks like this:

*import nltk*

*from nltk import sent\_tokenize*

*import numpy as np*

*import pandas as pd*

*from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer*

*from sklearn.metrics.pairwise import cosine\_similarity*

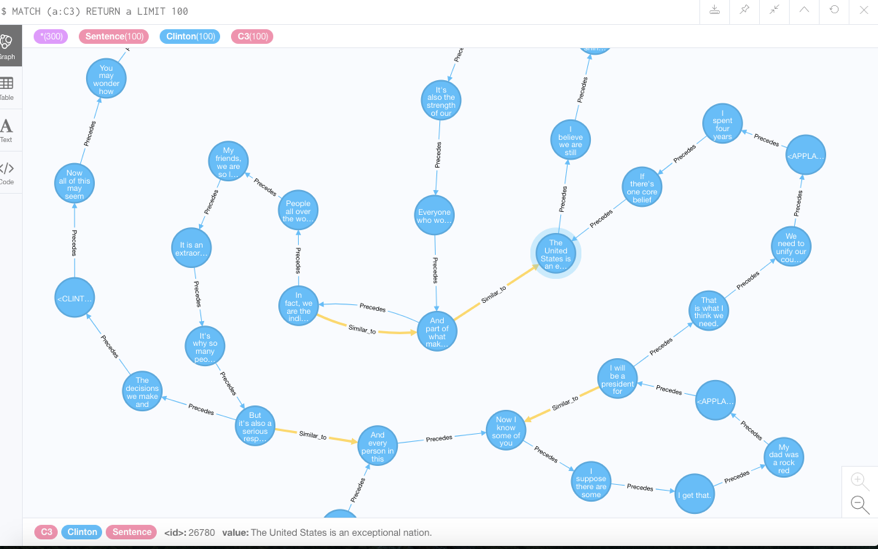
*from scipy.special import comb*

*import matplotlib.pyplot as plt*

*from py2neo import Node, Relationship, Graph*

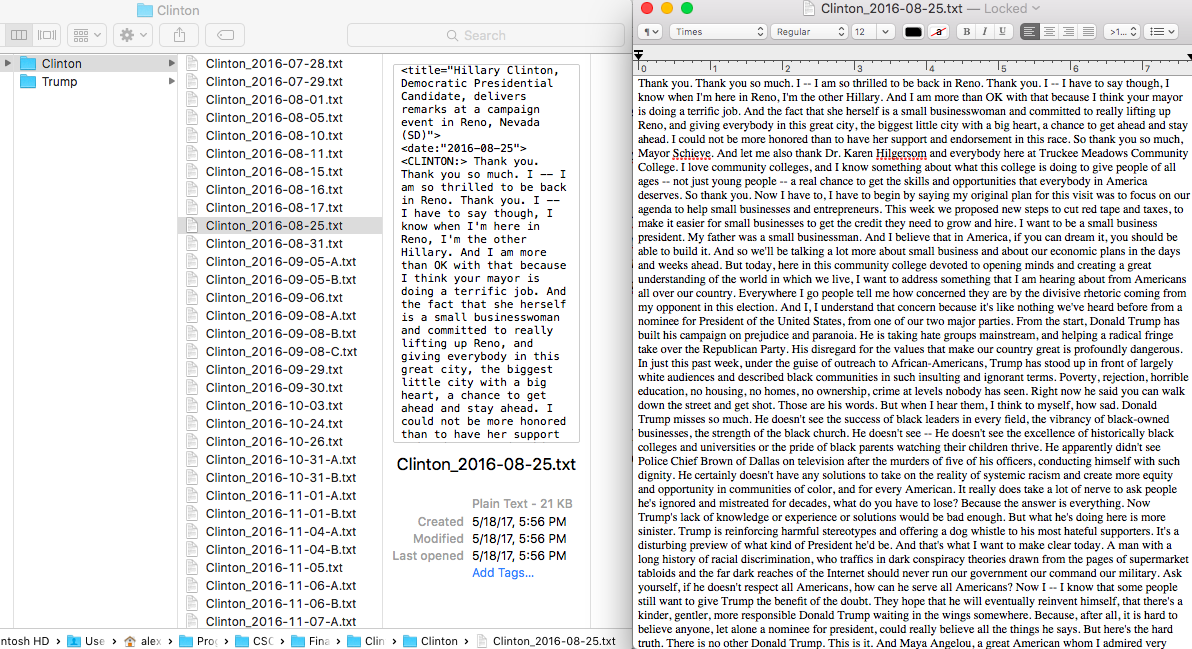
Data Prep

Our goal is to create a graph in neo4j that looks something like this:



Each node is a sentence and the nodes are connected by “Precedes” edges as well as “Similar\_to” edges. Out data prep needs to split the text into sentences as well as turn the sentences into vectors which can be used to find cosine similarities between nodes.

First, download the Clinton-Trump Corpus from [http://www.thegrammarlab.com/](http://www.thegrammarlab.com/?nor-portfolio=corpus-of-presidential-speeches-cops-and-a-clintontrump-corpus). The corpus is a collection plain text speeches from the two candidates.



Our goal is to work with this data at the sentence level, so we want to import the files and tokenize by sentence. Because the first sentence is often just metadata, we will skip it. Let’s write a function to load and tokenize each file. We use NLTK’s *sent\_tokenize* function.

*#function to load and tokenize data.*

*def Load\_Token(file\_path):*

*file=open(file\_path, 'r', encoding='utf-8')*

*data = file.read()*

*file.close()*

*sentences=sent\_tokenize(data)*

*return sentences[1:]*

We’ll stick to 5 speeches from each candidate. I created a *Tpaths* and *Cpaths* list which points to the files we want to bring in. I picked the first and last speeches for both candidates (party nomination acceptance and victory/concession speeches) and three in between. Now we can run our function on each one:

*#Load data*

*Folder="/Users/alexmarshall/programming.projects/CSCI E-63 (15499) course/FinalProj/Clinton-Trump Corpus/"*

*Tpaths=['Trump/Trump\_2016-07-22.txt','Trump/Trump\_2016-08-05.txt','Trump/Trump\_2016-09-07-B.txt'*

*,'Trump/Trump\_2016-10-28.txt','Trump/Trump\_2016-11-09.txt']*

*Cpaths=['Clinton/Clinton\_2016-07-28.txt','Clinton/Clinton\_2016-08-01.txt','Clinton/Clinton\_2016-08-31.txt'*

*,'Clinton/Clinton\_2016-09-05-B.txt','Clinton/Clinton\_2016-11-09.txt']*

*Tsent1=Load\_Token(Folder+Tpaths[0])*

*Tsent2=Load\_Token(Folder+Tpaths[1])*

*Tsent3=Load\_Token(Folder+Tpaths[2])*

*Tsent4=Load\_Token(Folder+Tpaths[3])*

*Tsent5=Load\_Token(Folder+Tpaths[4])*

*Csent1=Load\_Token(Folder+Cpaths[0])*

*Csent2=Load\_Token(Folder+Cpaths[1])*

*Csent3=Load\_Token(Folder+Cpaths[2])*

*Csent4=Load\_Token(Folder+Cpaths[3])*

*Csent5=Load\_Token(Folder+Cpaths[4])*

And check the results:

*print(Tsent1[50:54])*

*print("")*

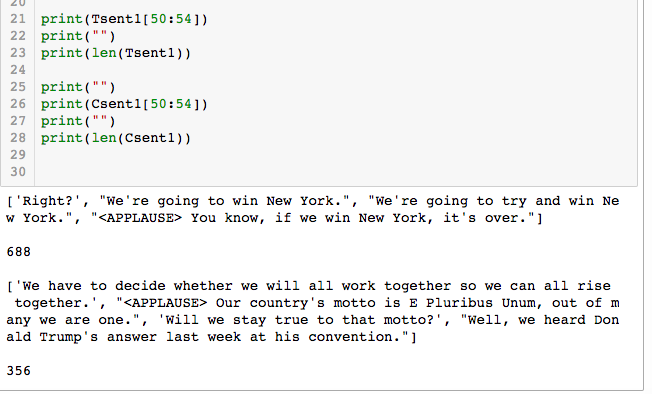
*print(len(Tsent1))*

*print("")*

*print(Csent1[50:54])*

*print("")*

*print(len(Csent1))*



The next thing we want to do is vectorize the sentences. There are some words which we do not want included if they’re common or don’t contain much information. We can take scikit-learn’s built in stop word list and add some of our own:

*#Add stop words to scikit learn's*

*stop\_wrds = text.ENGLISH\_STOP\_WORDS*

*stop\_wrds=stop\_wrds.union(['<applause>','(applause)','<CHEERING AND APPLAUSE>','<laughter>'\*

*,'<audience>','<clinton>','<trump>','\n'])*

Now we can add these stop words as an argument when we vectorize. We’ll use scikit-learn’s *CountVectorizer* and *TfidTransformer* to create the vectors. *CountVectorizer* will create a vocabulary and transform a corpus into one –hot encoded vectors. *TfidTransformer* will apply term frequency-inverse document frequency to weight each word within the vector based on how common it is within the whole text.

First we’ll define a *CountVectorizer* for the Clinton and Trump documents and a *TfidTransformer* which we’ll use for both. Then fit the Trump and Clinton documents separately. Finally we can transform the individual speeches.

*#define vectorizer and TfidfTransformer*

*Tvectorizer= CountVectorizer(stop\_words=stop\_wrds)*

*Cvectorizer = CountVectorizer(stop\_words=stop\_wrds)*

*tfidfTransf = TfidfTransformer()*

*#fit vocab*

*Tcorpus = Tsent1+Tsent2+Tsent3+Tsent4+Tsent5*

*Ccorpus = Csent1+Csent2+Csent3+Csent4+Csent5*

*Tvecs = Tvectorizer.fit\_transform(Tcorpus)*

*Cvecs = Cvectorizer.fit\_transform(Ccorpus)*

*#transform indiv Trump Speeches*

*Tvec1,Tvec2,Tvec3, Tvec4, Tvec5 =tfidfTransf.fit\_transform(Tvectorizer.transform(Tsent1))\*

*,tfidfTransf.fit\_transform(Tvectorizer.transform(Tsent2))\*

*,tfidfTransf.fit\_transform(Tvectorizer.transform(Tsent3))\*

*,tfidfTransf.fit\_transform(Tvectorizer.transform(Tsent4))\*

*,tfidfTransf.fit\_transform(Tvectorizer.transform(Tsent5))*

*#transform indiv Clinton Speeches*

*Cvec1,Cvec2,Cvec3,Cvec4,Cvec5 =tfidfTransf.fit\_transform(Cvectorizer.transform(Csent1))\*

*,tfidfTransf.fit\_transform(Cvectorizer.transform(Csent2))\*

*,tfidfTransf.fit\_transform(Cvectorizer.transform(Csent3))\*

*,tfidfTransf.fit\_transform(Cvectorizer.transform(Csent4))\*

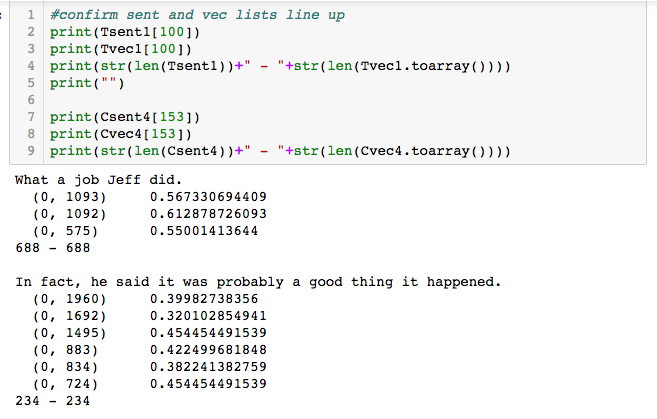
*,tfidfTransf.fit\_transform(Cvectorizer.transform(Csent5))*

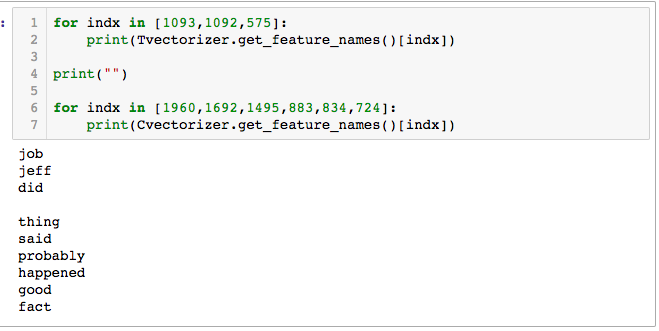
Now we can check the results. The output is a sparse matrix with each row representing a sentence and each column a word. Because we one-hot encoded, the arrays are too sparse to fit on a screen, but we can sum them to see that something is there. If we did not use TFID the sum would be equal to the number of words in the sentence, however TFID weights and normalizes so that each vector has a Euclidean distance of 1.



we can also see the words associated with each position with the *get\_feature\_names()* method, and the position of each word with the *vocabulary\_.get()* method.







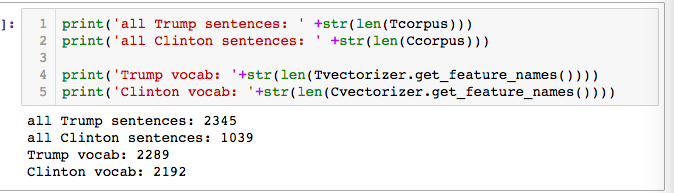
This looks good! We’re ready to write to Neo4j. One last thing before moving on -let’s get an idea of the magnitudes of the Clinton and Trump texts and vocabularies.

*print('all Trump sentences: ' +str(len(Tcorpus)))*

*print('all Clinton sentences: ' +str(len(Ccorpus)))*

*print('Trump vocab: '+str(len(Tvectorizer.get\_feature\_names())))*

*print('Clinton vocab: '+str(len(Cvectorizer.get\_feature\_names())))*



Keep in mind that even though the vocabulary and number of speeches are similar, The trump speeches as a whole have more than double the sentences.

Import to Neo4j

Now that our data is ready, we need to write it to Neo4j graphs. We will interact with Neo4j through the UI as well as through the API with Python and py2neo. We already imported what we need from py2neo so let’s define our graph, clear the database. My DB resides at localhost 7474, most likely yours does too.

*from py2neo import Node, Relationship, Graph*

*graph = Graph("http://neo4j:neo4j@localhost:7474")*

*#clear graph*

*graph.delete\_all()*

Now let’s write a function that allows us to write nodes from our text

*#Function to write nodes and precedes edges to neo4j*

*def Neo\_Write(Sentences,Author, Work):*

*for indx in range((len(Sentences)-1)):*

*relat=Relationship(Node("Sentence", Author, Work, value=Sentences[indx]),"Precedes"\*

*,Node("Sentence", Author, Work, value=Sentences[indx+1]))*

*graph.merge(relat)*

This function will take a vector of sentences, an author label and a work label and create a directional graph in neo4j with each sentence as a node pointing to the proceding sentence. We’ll call this function for each speech and label by the speaker and individual speech.

*Neo\_Write(Tsent1,'Trump','T1')*

*Neo\_Write(Tsent2,'Trump','T2')*

*Neo\_Write(Tsent3,'Trump','T3')*

*Neo\_Write(Tsent4,'Trump','T4')*

*Neo\_Write(Tsent5,'Trump','T5')*

*Neo\_Write(Csent1,'Clinton','C1')*

*Neo\_Write(Csent2,'Clinton','C2')*

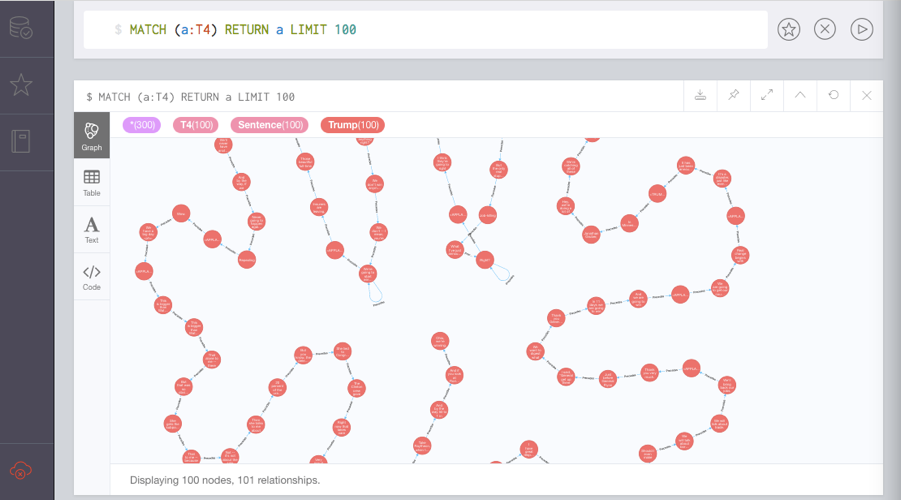
*Neo\_Write(Csent3,'Clinton','C3')*

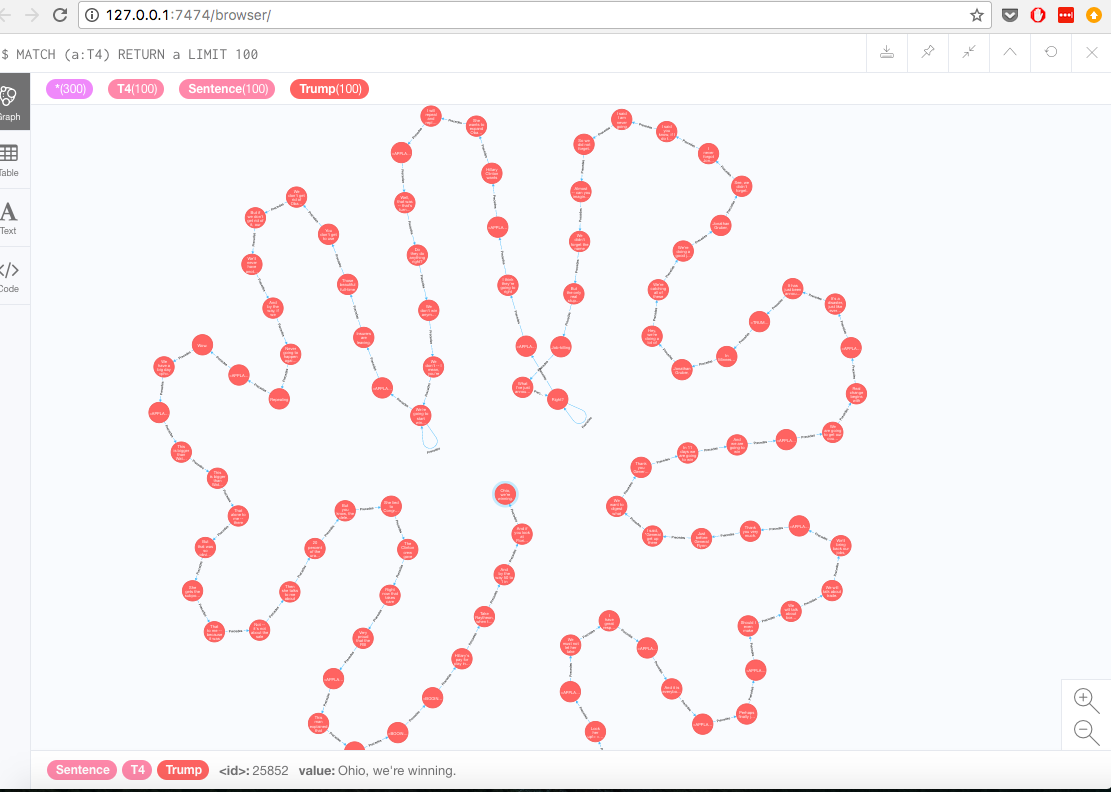
*Neo\_Write(Csent4,'Clinton','C4')*

*Neo\_Write(Csent5,'Clinton','C5')*

And let’s open Neo4j and enter some Cypher code to check the result

*MATCH (a:T4) RETURN a LIMIT 100*





This is a great start! Since we want to analyze cycles in text where sentences are similar to each other, we need to create another type of edge based on similarity – this is where the vectorization comes in. We will calculate the cosine similarity between each vector using *cosine\_similarity* from scikit-learn. This function takes our vectors and outputs a matrix showing the cosine similarity between each vector. We’ll create our own function to loop through this matrix and create new edges in our graph for similar vectors. I set a threshold of 0.4, so that cosine similarity between nodes must be greater than 0.4 for an edge to be drawn. You can play around with this threshold and compare sentences at different thresholds. I found that 0.4 produced decently similar sentences while keeping the number of connections within a reasonable range for both Trump and Clinton’s speeches.

*#function to calculate and add Cosine similarity relationship between nodes within a work*

*def Neo\_Sim(sents, vecs, Author, Work):*

*CosSimMat=cosine\_similarity(vecs, dense\_output=True)*

*for indxA in range(len(sents)-1):*

*for indxB in range(indxA+1,len(sents)):*

*CosSim=CosSimMat[indxA,indxB]*

*if CosSim>0.4: #set based on what seemed to capture simlar sentences best*

*relat=Relationship(Node("Sentence", Author, Work, value=sents[indxB]),\*

*"Similar\_to",\*

*Node("Sentence", Author, Work, value=sents[indxA]),\*

*similarity = CosSim)*

*graph.merge(relat)*

Now we can run this function for each speech and see the results.

*Neo\_Sim(Tsent1,Tvec1,'Trump','T1')*

*Neo\_Sim(Tsent2,Tvec2,'Trump','T2')*

*Neo\_Sim(Tsent3,Tvec3,'Trump','T3')*

*Neo\_Sim(Tsent4,Tvec4,'Trump','T4')*

*Neo\_Sim(Tsent5,Tvec5,'Trump','T5')*

*Neo\_Sim(Csent1,Cvec1,'Clinton','C1')*

*Neo\_Sim(Csent2,Cvec2,'Clinton','C2')*

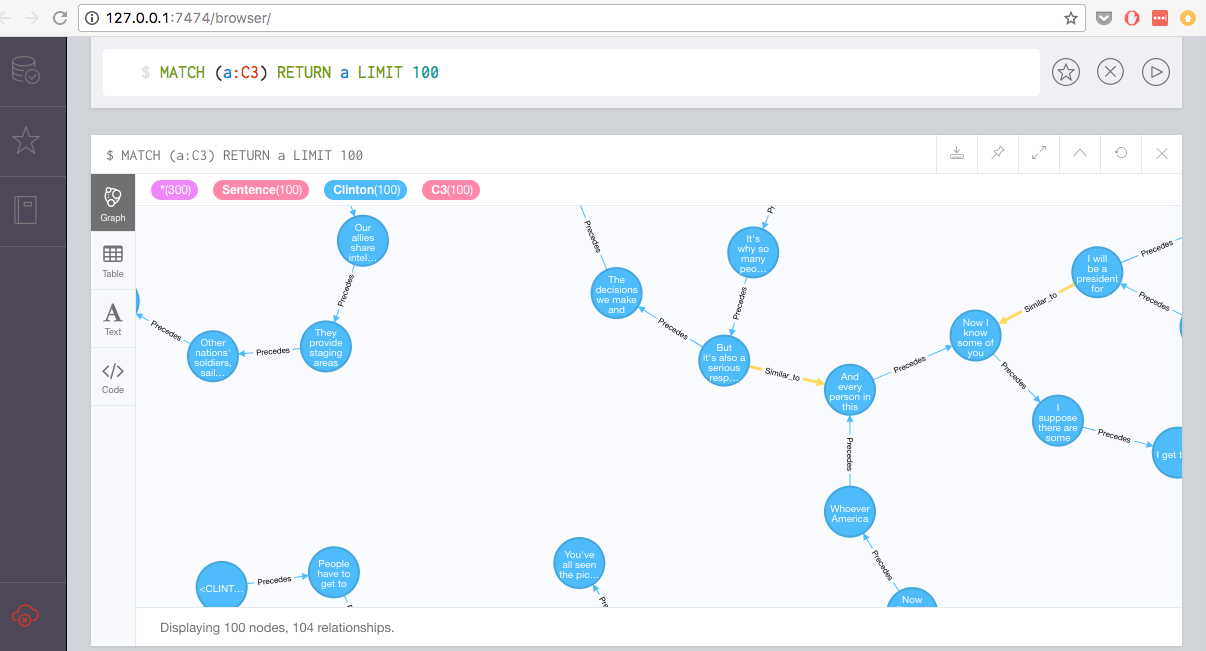
*Neo\_Sim(Csent3,Cvec3,'Clinton','C3')*

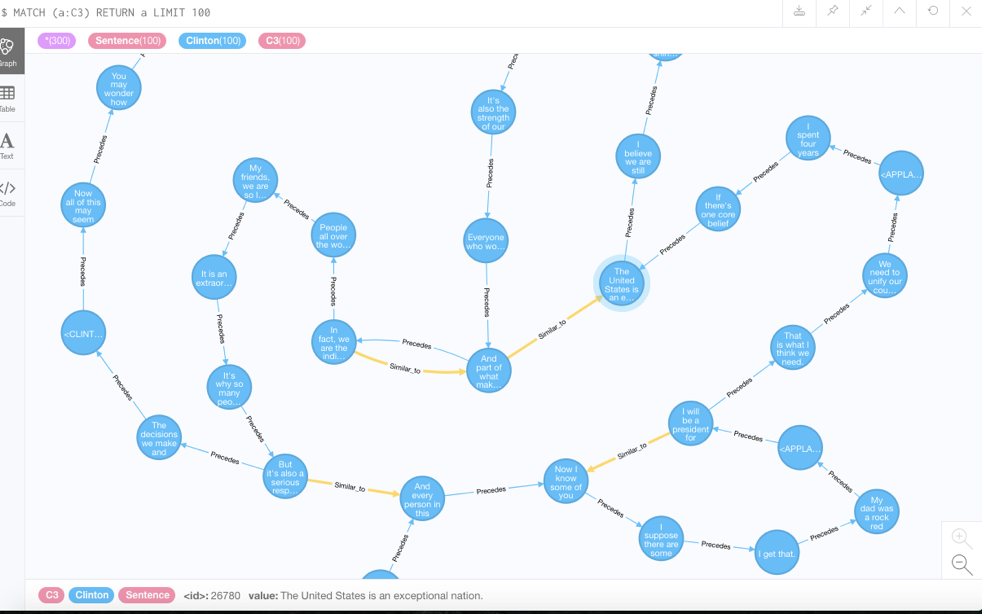
*Neo\_Sim(Csent4,Cvec4,'Clinton','C4')*

*Neo\_Sim(Csent5,Cvec5,'Clinton','C5')*

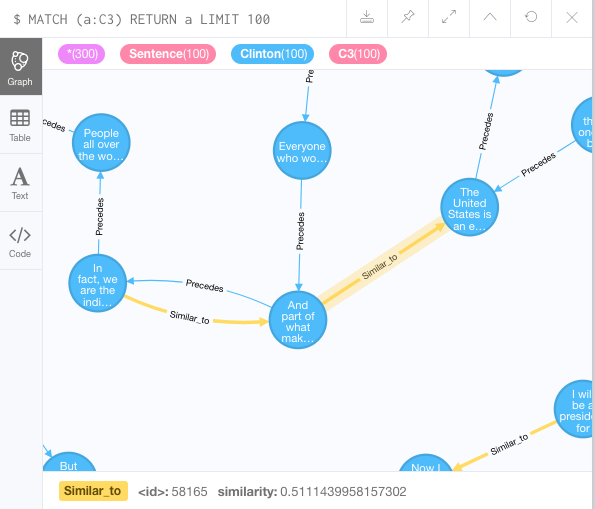
In Neo4j UI:

*MATCH (a:C3) RETURN a LIMIT 100*





The cosine similarity value is also attached as a property of the “Similar\_to” edges



That completes the set up! Now we can explore the results and see if we can find anything interesting.

Analysis

Let’s create a function that queries neo4j and writes the results to a data frame. To start we’ll calculate some summary statistics – number of “Precedes” edges, number of nodes, number of “Similar\_to” edges, number of nodes with at least one similar to edge pointing away, average cosine similarity of “Similar\_to” edges and the standard deviation of “Similar\_to” edges. Keep in mind that the minimum similarity of a “Similar\_to” edge is 0.4. I’m also interested in seeing the sentences with the most “Similar\_to” edges coming off of them. In the same function we’ll create another dataframe listing the top 10 for each speech.

*#metrics*

*#function that writes metrics to data frame*

*def metricsDF(lbl1,lbl2,lbl3,lbl4,lbl5,lbl6):*

*#1 summary stats*

*numP=graph.data("MATCH (a:"+lbl1+")-[p:Precedes]->(b) RETURN count(p) as numP")*

*numNode=graph.data("MATCH (a:"+lbl1+") RETURN count(a) as numNodes")*

*numSE=graph.data("MATCH (a:"+lbl1+")-[s:Similar\_to]->(b) RETURN count(s) as numSE")*

*numCnct= graph.data("Match (a:"+lbl1+")-[:Similar\_to]->(b) RETURN count(DISTINCT a) as NSE")*

*simStat=graph.data("MATCH (a:"+lbl1+")-[s:Similar\_to]->(b) RETURN avg(s.similarity) as avgSim, stDev(s.similarity) as stdevSim")*

*dict={lbl1 : [numP[0]['numP'],numNode[0]['numNodes'],numSE[0]['numSE'],numCnct[0]['NSE'],simStat[0]['avgSim'], simStat[0]['stdevSim']]}*

*for label in [lbl2,lbl3,lbl4,lbl5,lbl6]:*

*numP=graph.data("MATCH (a:"+label+")-[p:Precedes]->(b) RETURN count(p) as numP")*

*numNode=graph.data("MATCH (a:"+label+") RETURN count(a) as numNodes")*

*numSE=graph.data("MATCH (a:"+label+")-[s:Similar\_to]->(b) RETURN count(s) as numSE")*

*numCnct= graph.data("Match (a:"+label+")-[:Similar\_to]->(b) RETURN count(DISTINCT a) as NSE")*

*simStat=graph.data("MATCH (a:"+label+")-[s:Similar\_to]->(b) RETURN avg(s.similarity) as avgSim, stDev(s.similarity) as stdevSim")*

*dict.update({label : [numP[0]['numP'],numNode[0]['numNodes'],numSE[0]['numSE'],numCnct[0]['NSE'],simStat[0]['avgSim'], simStat[0]['stdevSim']]})*

*DF1= pd.DataFrame(dict,index=['precEdges','nodes','simEdges','nodesWithSimEdge','avgSim','stdevSim'])*

*#2 nodes with most 'similar\_to' edges*

*DF2=pd.DataFrame()*

*for label in [lbl1,lbl2,lbl3,lbl4,lbl5]:*

*tmpDF=pd.DataFrame(graph.data("MATCH (a:"+label+")-[s:Similar\_to]->(b) RETURN b, count(s) order by count(s) Desc LIMIT 10"))*

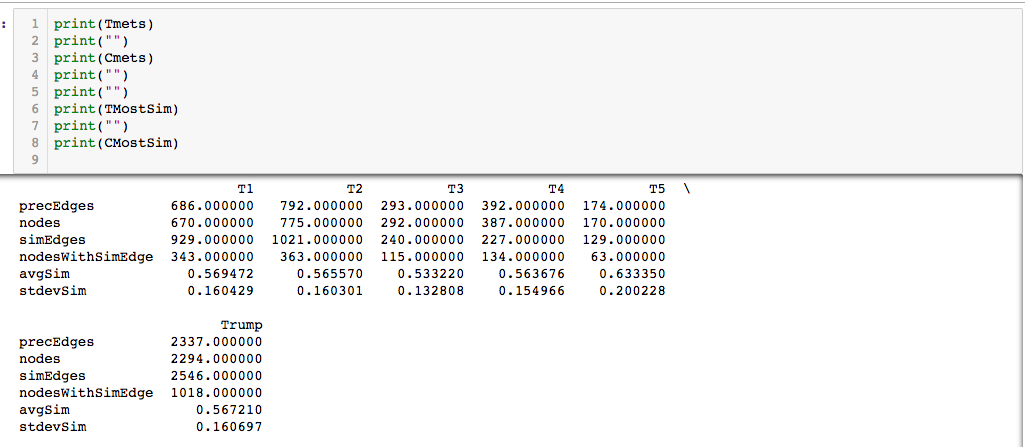
*DF2=pd.concat([DF2,tmpDF],ignore\_index=True)*

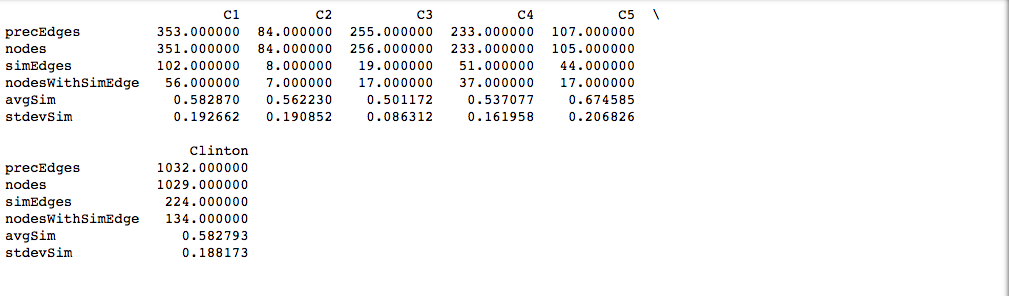
*return DF1,DF2*

*Tmets,TMostSim=metricsDF('T1','T2','T3','T4','T5','Trump')*

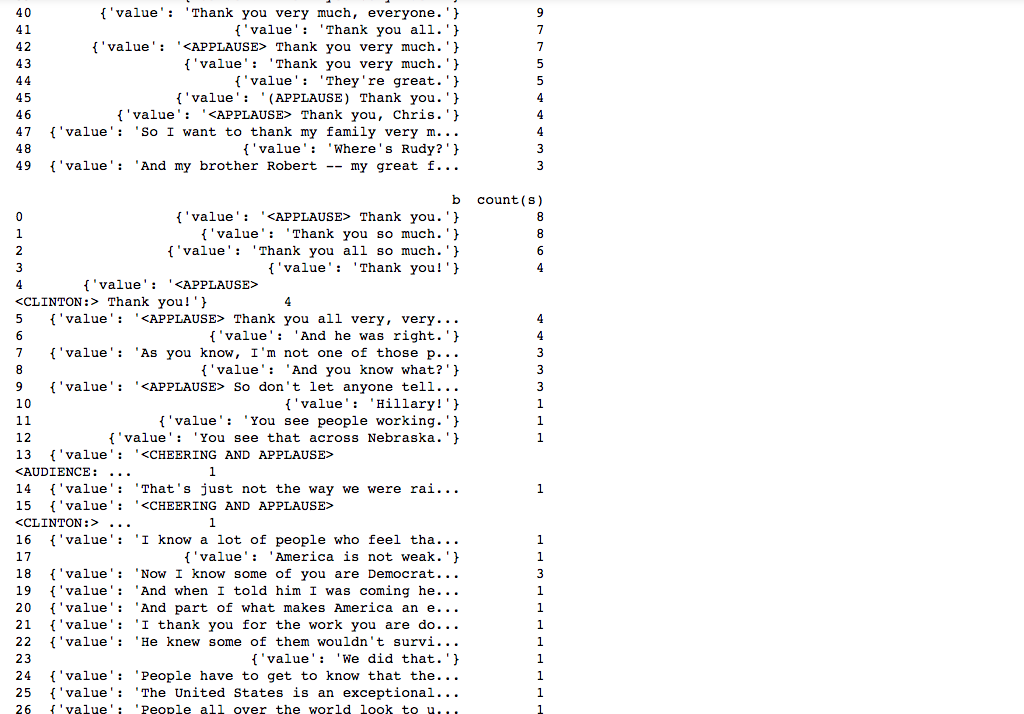
*Cmets,CMostSim=metricsDF('C1','C2','C3','C4','C5','Clinton')*

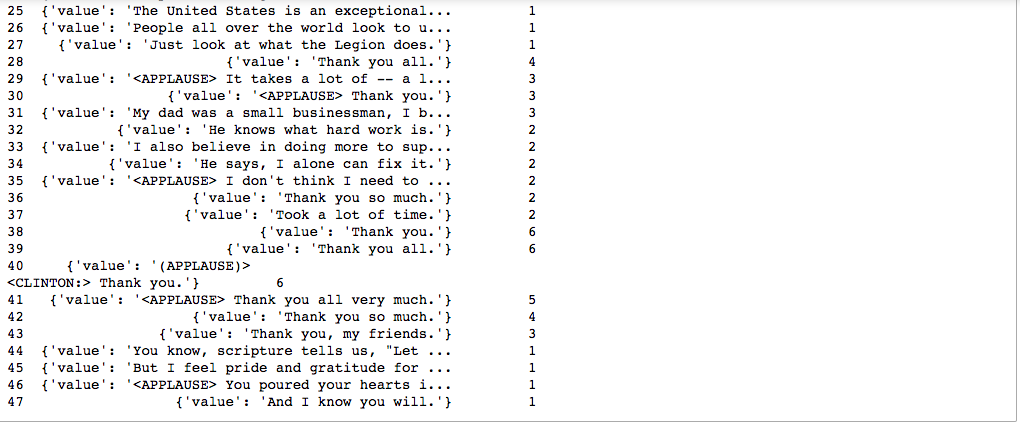
Results below:











The most salient observation here is how many more “Similar\_to” connections trump speeches have than Clinton’s. Trump also has more sentences in general. A good way to compare is using the “nodesWithSimEdge” metric and relating that to the number of nodes. This is a good measure for how often sentences are similar to each other in a work. Trump has around double number of sentences but ten times the number of sentences which meet the similarity threshold for at least one other sentence.  Trump’s speeches are fairly consistent with around half of the sentences meeting the threshold for another sentence above it. Clinton’s speeches have at most a fifth of the nodes connecting via a “Similar\_to” edge.

Examining the sentences that connect to other previous sentences most often in each speech, we see a lot of “I said”, “You know”, “Win” , “Thank you”, and people’s names in the Turmp speeches. In the Clinton speeches , there are much less sentences with multiple connections, many of them contain “Thank you”.

I was hoping this method would show repetitive sentences and themes, but it seems like we are getting only the repetitive sentences - shout outs to proper nouns, thank you’s, and crutch words dominate these nodes. My hypothesis was that we could separate “theme cycles” which would be longer and have a lower similarity from “repeat cycles”. It’s still possible to tease out these signatures, possibly by lowering the similarity threshold but it’s not obvious right now.

I’m also interested in seeing the frequency distribution for cycle lengths. Let’s create a new function to do this. It gets computationally expensive the longer the path we look for so I limited the number of “Precedes” nodes to 150.

*# funct whch creates freq dists for cycle length up to 151*

*def freqDF(lbl1,lbl2,lbl3,lbl4,lbl5):*

*RsltDF=pd.DataFrame(graph.data("MATCH cyc=(a:"+lbl1+")-[p:Precedes\*..150]->(b)-[s:Similar\_to]->(a) RETURN length(cyc) as cyc\_length, count(\*) as freq, avg(s.similarity) as avgSim Order by length(cyc)"))*

*RsltDF = RsltDF.rename({'freq':'freq\_'+lbl1,'avgSim':'avgSim\_'+lbl1},axis='columns')*

*RsltDF['freq\_Tot']=RsltDF.fillna(0)['freq\_'+lbl1]*

*for label in [lbl2,lbl3,lbl4,lbl5]:*

*DF2=pd.DataFrame(graph.data("MATCH cyc=(a:"+label+")-[p:Precedes\*..150]->(b)-[s:Similar\_to]->(a) RETURN length(cyc) as cyc\_length, count(\*) as freq, avg(s.similarity) as avgSim Order by length(cyc)"))*

*DF2=DF2.rename({'freq':'freq\_'+label,'avgSim':'avgSim\_'+label},axis='columns')*

*RsltDF=pd.merge(left=RsltDF,right=DF2,how='outer',left\_on='cyc\_length',right\_on='cyc\_length')*

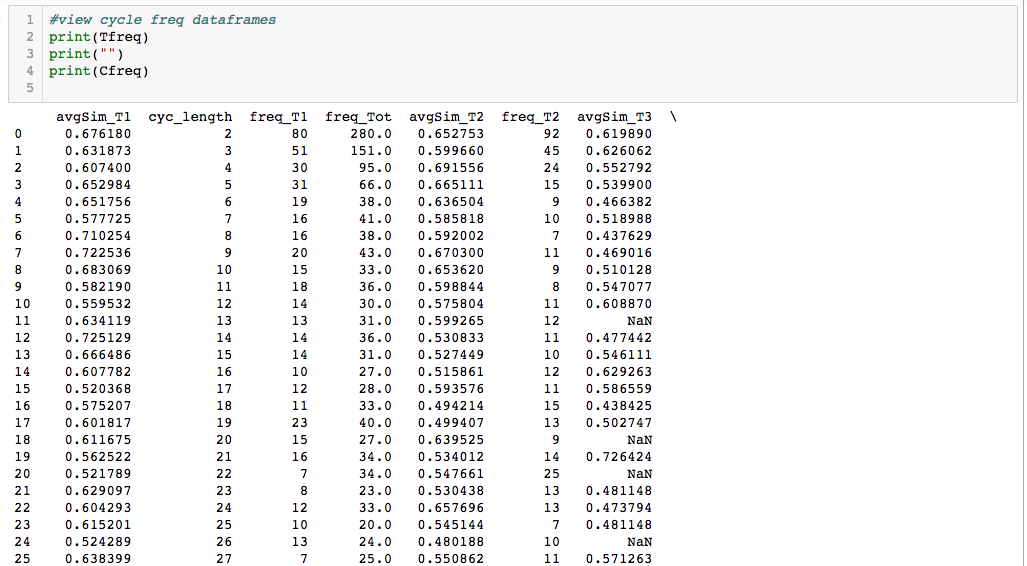
*RsltDF['freq\_Tot']=RsltDF['freq\_Tot']+RsltDF.fillna(0)['freq\_'+label]*

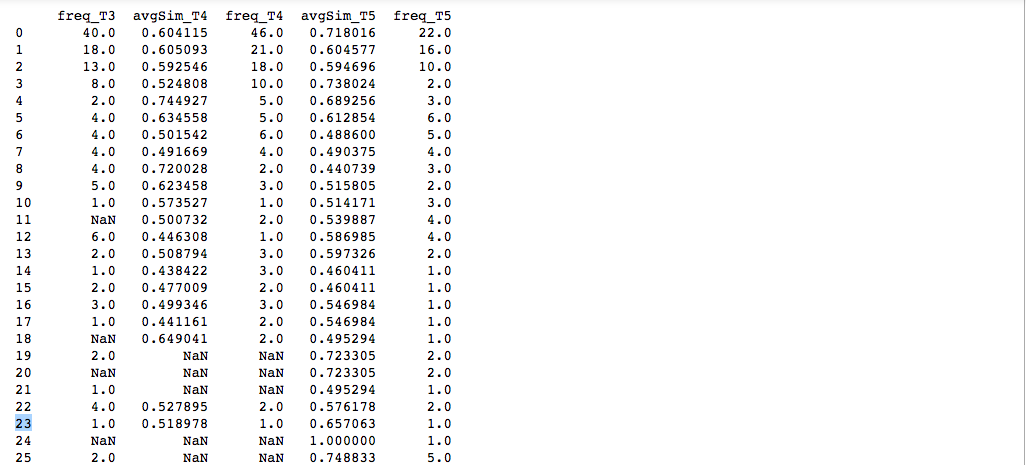
*return RsltDF*

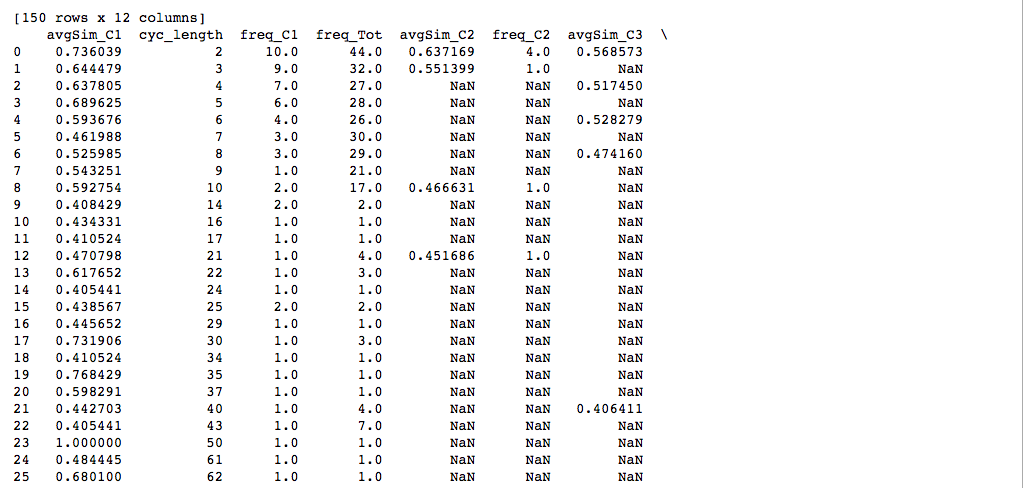
*Tfreq=freqDF('T1','T2','T3','T4','T5')*

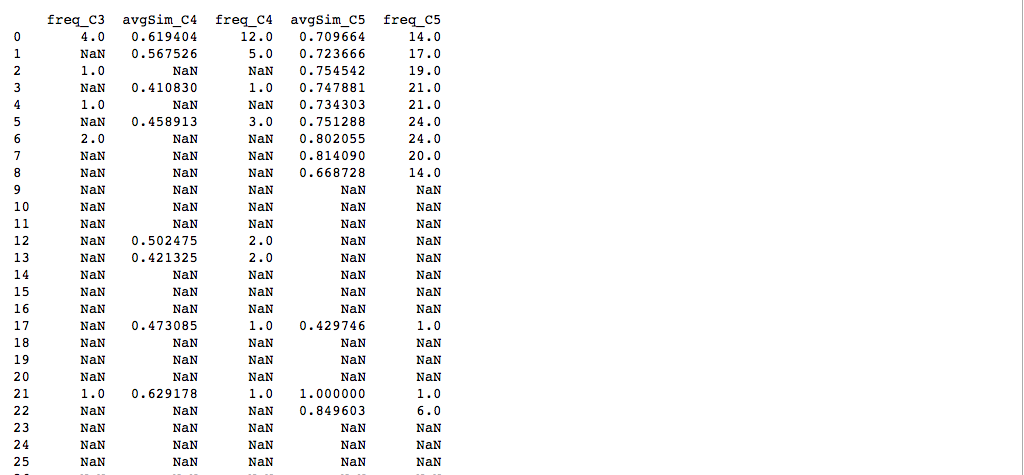
*Cfreq=freqDF('C1','C2','C3','C4','C5')*

I’ll just show a sample of the output, but we’ll graph the entire thing.









This will be easier to see as a graph. I want to make a chart for each speech and for the result as a whole. Showing the number of cycles for each cycle length as well as a line showing the average sentence cosine similarity for each cycle length. We’ll use MatPlotlib to create these graphs.

*#Trump cycle frequency charts*

*fig1=plt.figure()*

*ax1=fig1.add\_subplot(111)*

*line=ax1.plot(Tfreq['cyc\_length'],Tfreq['avgSim\_T1'],'r')*

*ax1.yaxis.tick\_right()*

*ax1.yaxis.set\_label\_position("right")*

*ax1.axes.set\_ylim([0,1])*

*plt.ylabel('avg similarity')*

*ax2=fig1.add\_subplot(111,sharex=ax1,frameon=False)*

*bar=ax2.bar(Tfreq['cyc\_length'],Tfreq['freq\_T1'])*

*plt.ylabel('Freq')*

*plt.xlabel('cycle length')*

*plt.title('T1 Cycle Freq Dist')*

*plt.show()*

*fig2=plt.figure()*

*ax1=fig2.add\_subplot(111)*

*line=ax1.plot(Tfreq['cyc\_length'],Tfreq['avgSim\_T2'],'r')*

*ax1.yaxis.tick\_right()*

*ax1.yaxis.set\_label\_position("right")*

*ax1.axes.set\_ylim([0,1])*

*plt.ylabel('avg similarity')*

*ax2=fig2.add\_subplot(111,sharex=ax1,frameon=False)*

*bar=ax2.bar(Tfreq['cyc\_length'],Tfreq['freq\_T2'])*

*plt.ylabel('Freq')*

*plt.xlabel('cycle length')*

*plt.title('T2 Cycle Freq Dist')*

*plt.show()*

*fig3=plt.figure()*

*ax1=fig3.add\_subplot(111)*

*line=ax1.plot(Tfreq['cyc\_length'],Tfreq['avgSim\_T3'],'r')*

*ax1.yaxis.tick\_right()*

*ax1.yaxis.set\_label\_position("right")*

*ax1.axes.set\_ylim([0,1])*

*plt.ylabel('avg similarity')*

*ax2=fig3.add\_subplot(111,sharex=ax1,frameon=False)*

*bar=ax2.bar(Tfreq['cyc\_length'],Tfreq['freq\_T3'])*

*plt.ylabel('Freq')*

*plt.xlabel('cycle length')*

*plt.title('T3 Cycle Freq Dist')*

*plt.show()*

*fig4=plt.figure()*

*ax1=fig4.add\_subplot(111)*

*line=ax1.plot(Tfreq['cyc\_length'],Tfreq['avgSim\_T4'],'r')*

*ax1.yaxis.tick\_right()*

*ax1.yaxis.set\_label\_position("right")*

*ax1.axes.set\_ylim([0,1])*

*plt.ylabel('avg similarity')*

*ax2=fig4.add\_subplot(111,sharex=ax1,frameon=False)*

*bar=ax2.bar(Tfreq['cyc\_length'],Tfreq['freq\_T4'])*

*plt.ylabel('Freq')*

*plt.xlabel('cycle length')*

*plt.title('T4 Cycle Freq Dist')*

*plt.show()*

*fig5=plt.figure()*

*ax1=fig5.add\_subplot(111)*

*line=ax1.plot(Tfreq['cyc\_length'],Tfreq['avgSim\_T5'],'r')*

*ax1.yaxis.tick\_right()*

*ax1.yaxis.set\_label\_position("right")*

*ax1.axes.set\_ylim([0,1])*

*plt.ylabel('avg similarity')*

*ax2=fig5.add\_subplot(111,sharex=ax1,frameon=False)*

*bar=ax2.bar(Tfreq['cyc\_length'],Tfreq['freq\_T5'])*

*plt.ylabel('Freq')*

*plt.xlabel('cycle length')*

*plt.title('T5 Cycle Freq Dist')*

*plt.show()*

*plt.bar(Tfreq['cyc\_length'],Tfreq['freq\_Tot'])*

*plt.xlabel('cycle length')*

*plt.ylabel('Freq')*

*plt.title('Trump total Cycle Freq Dist')*

*plt.show()*

*print(Tfreq.loc[23,'freq\_Tot'])*

*print(Tfreq.loc[144,'freq\_Tot'])*

##########################################

#Clinton cycle frequency charts

fig1=plt.figure()

ax1=fig1.add\_subplot(111)

line=ax1.plot(Cfreq['cyc\_length'],Cfreq['avgSim\_C1'],'r')

ax1.yaxis.tick\_right()

ax1.yaxis.set\_label\_position("right")

ax1.axes.set\_ylim([0,1])

plt.ylabel('avg similarity')

ax2=fig1.add\_subplot(111,sharex=ax1,frameon=False)

bar=ax2.bar(Cfreq['cyc\_length'],Cfreq['freq\_C1'])

plt.ylabel('Freq')

plt.xlabel('cycle length')

plt.title('C1 Cycle Freq Dist')

plt.show()

fig2=plt.figure()

ax1=fig2.add\_subplot(111)

line=ax1.plot(Cfreq['cyc\_length'],Cfreq['avgSim\_C2'],'r')

ax1.yaxis.tick\_right()

ax1.yaxis.set\_label\_position("right")

ax1.axes.set\_ylim([0,1])

plt.ylabel('avg similarity')

ax2=fig2.add\_subplot(111,sharex=ax1,frameon=False)

bar=ax2.bar(Cfreq['cyc\_length'],Cfreq['freq\_C2'])

plt.ylabel('Freq')

plt.xlabel('cycle length')

plt.title('C2 Cycle Freq Dist')

plt.show()

fig3=plt.figure()

ax1=fig3.add\_subplot(111)

line=ax1.plot(Cfreq['cyc\_length'],Cfreq['avgSim\_C3'],'r')

ax1.yaxis.tick\_right()

ax1.yaxis.set\_label\_position("right")

ax1.axes.set\_ylim([0,1])

plt.ylabel('avg similarity')

ax2=fig3.add\_subplot(111,sharex=ax1,frameon=False)

bar=ax2.bar(Cfreq['cyc\_length'],Cfreq['freq\_C3'])

plt.ylabel('Freq')

plt.xlabel('cycle length')

plt.title('C3 Cycle Freq Dist')

plt.show()

fig4=plt.figure()

ax1=fig4.add\_subplot(111)

line=ax1.plot(Cfreq['cyc\_length'],Cfreq['avgSim\_C4'],'r')

ax1.yaxis.tick\_right()

ax1.yaxis.set\_label\_position("right")

ax1.axes.set\_ylim([0,1])

plt.ylabel('avg similarity')

ax2=fig4.add\_subplot(111,sharex=ax1,frameon=False)

bar=ax2.bar(Cfreq['cyc\_length'],Cfreq['freq\_C4'])

plt.ylabel('Freq')

plt.xlabel('cycle length')

plt.title('C4 Cycle Freq Dist')

plt.show()

fig5=plt.figure()

ax1=fig5.add\_subplot(111)

line=ax1.plot(Cfreq['cyc\_length'],Cfreq['avgSim\_C5'],'r')

ax1.yaxis.tick\_right()

ax1.yaxis.set\_label\_position("right")

ax1.axes.set\_ylim([0,1])

plt.ylabel('avg similarity')

ax2=fig5.add\_subplot(111,sharex=ax1,frameon=False)

bar=ax2.bar(Cfreq['cyc\_length'],Cfreq['freq\_C5'])

plt.ylabel('Freq')

plt.xlabel('cycle length')

plt.title('C5 Cycle Freq Dist')

plt.show()

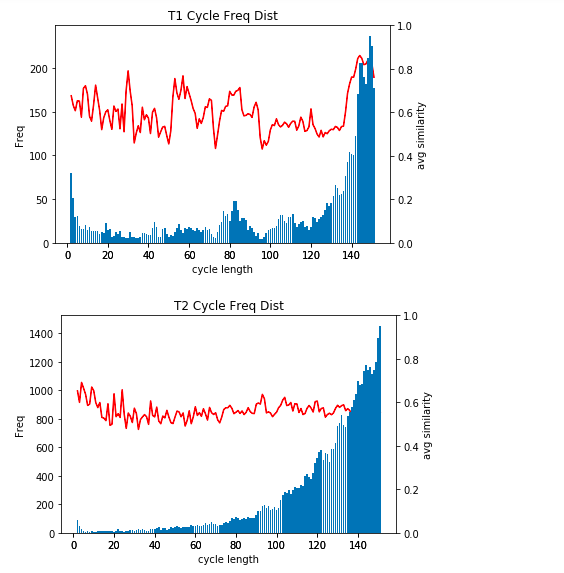
plt.bar(Cfreq['cyc\_length'],Cfreq['freq\_Tot'])

plt.xlabel('cycle length')

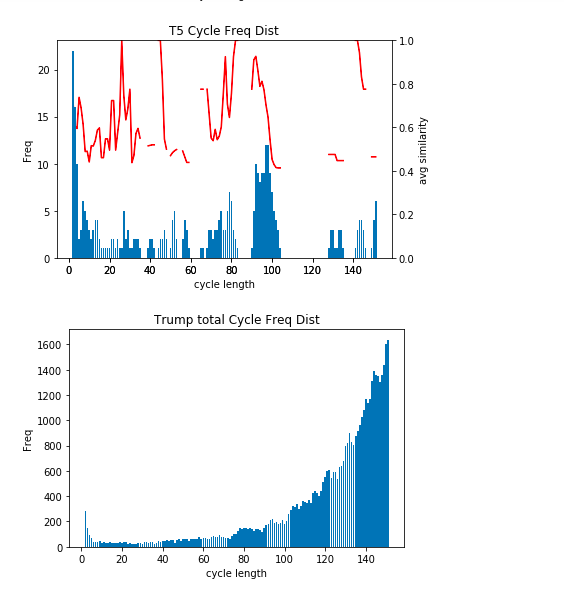
plt.ylabel('Freq')

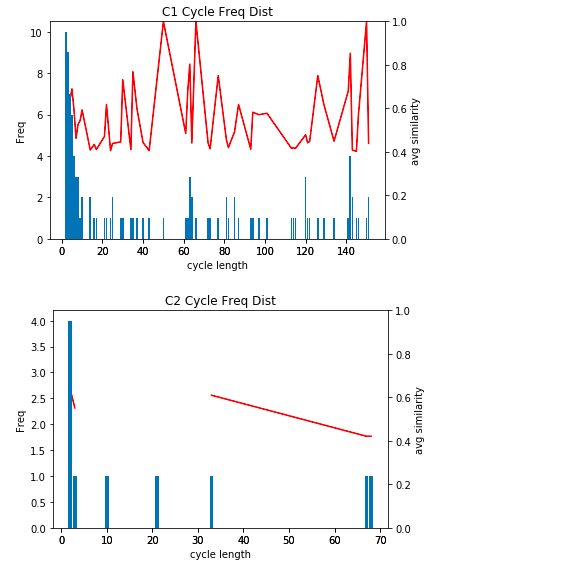
plt.title('Clinton total Cycle Freq Dist')

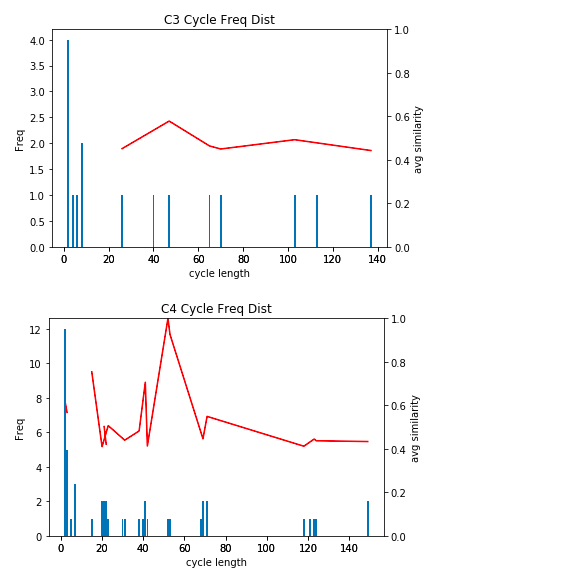
plt.show()

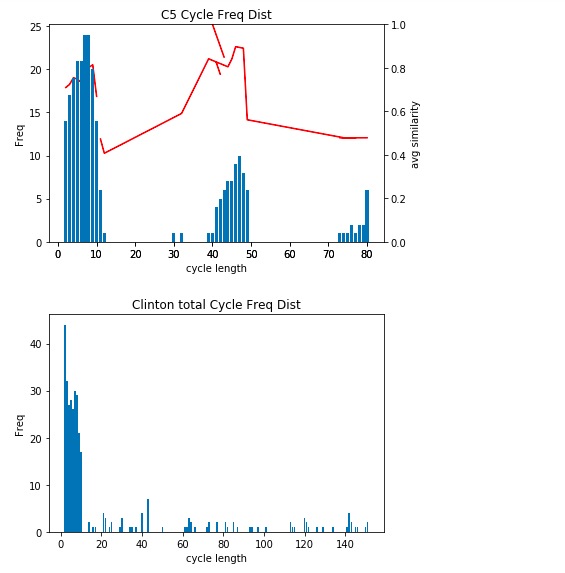










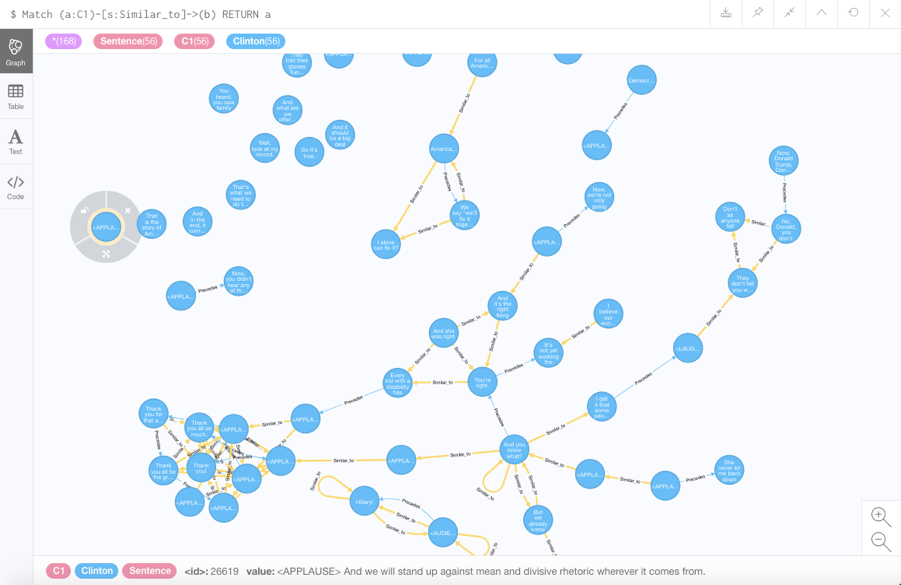


In general frequency is higher at lower cycle lengths but occasionally there are very large spikes at longer frequencies and for T2 the frequencies grow exponentially. I’m interested in the spike in average similarity and frequency around length 145 for T1, why the shape of T2 is so unexpected and the spike in T5 around length 100. The total chart for Trump speeches is largely driven by T2. The Clinton speech cycle length distributions are more similar with earlier spikes and not much after. They’re also much more sparse. C5 has the most cycles, with a spike around length 6 in frequency and average similarity and almost nothing until length 40-50 and then a few more cycles at the largest lengths. The total distribution is very concentrated at the earlier lengths.

As we’ve seen, one of the best features of Neo4j is the built in visualization tool in its user interface. We can dig into some of the phenomena we’ve seen or we’re interested in with these visualizations.

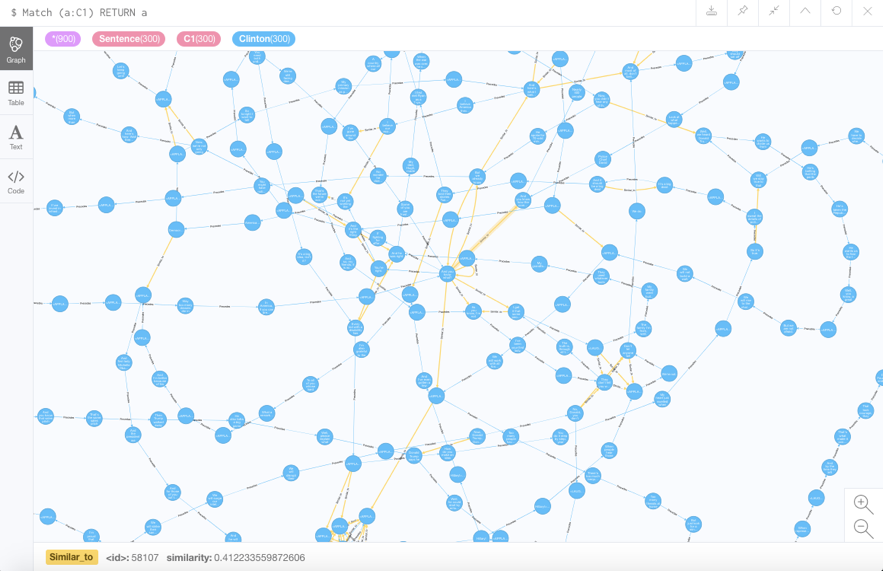
Clinton speeches tend to be shorter and have less “Similar\_to” edges than Trump ‘s. We can look at all the nodes with “Similar\_to” connections in one window:

*Match (a:C1)-[s:Similar\_to]->(b) RETURN a*



Or see an entire speech:

*Match (a:C1) RETURN a*



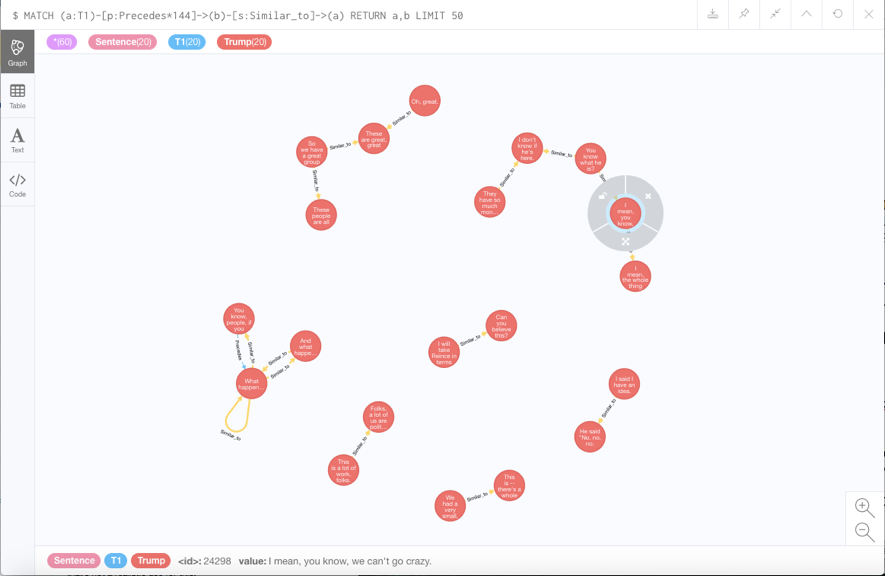
Of the Trump Speeches, T5 is the most similar to a Clinton speech and is more manageable to look at the entire thing.

*Match (a:T5)-[s:Similar\_to]->(b) RETURN a*

*Match (a:T5) RETURN a*

In T1, we noticed a spike in similarity and number of cycles of length 145. So we can look at the connecting nodes at this cycle length. The interesting thing here is that there are only a few connecting nodes that all these cycles are built around.

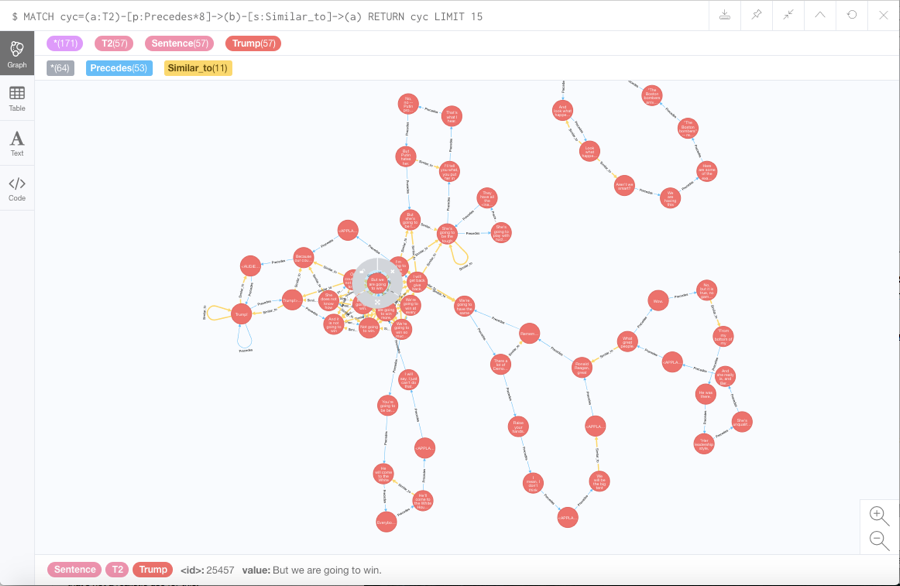
*MATCH (a:T1)-[p:Precedes\*144]->(b)-[s:Similar\_to]->(a) RETURN a,b LIMIT 50*

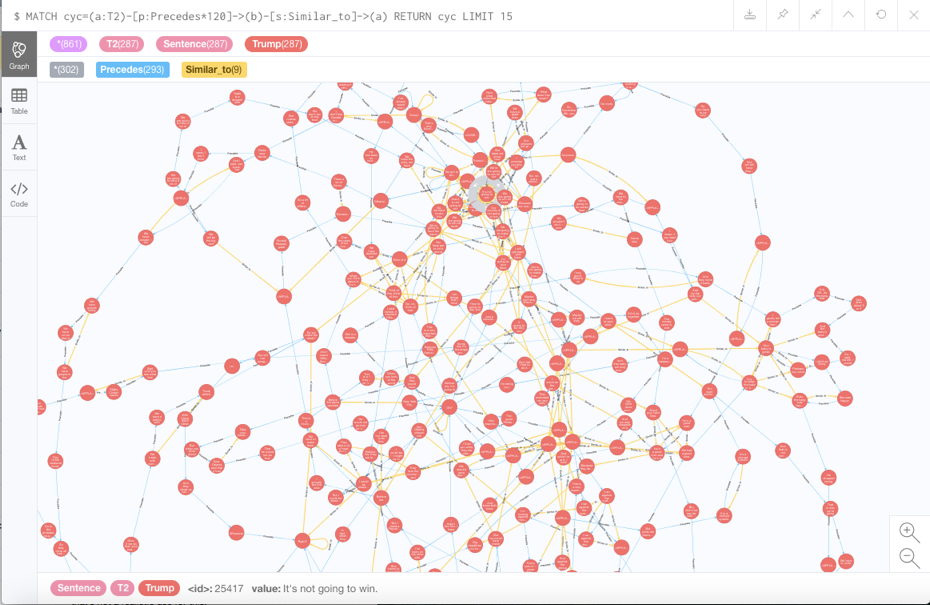


If we look at T2 which has the exponential growth in cycle frequencies, at an early and late point we see the reason immediately. There is a web of connected nodes with the word “win” with many of them close together but some far away in the text. There are a few others like that as well, look for one focused on the word “System”. This allows an explosion of ways to complete a cycle especially if the text contains sentences that are repeated verbatim.

*MATCH cyc=(a:T2)-[p:Precedes\*8]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*

*MATCH cyc=(a:T2)-[p:Precedes\*120]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*





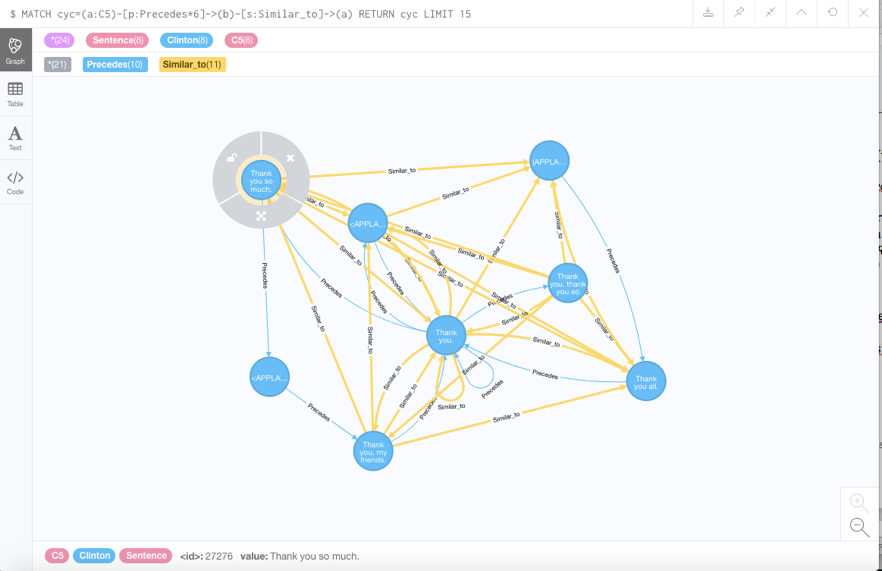
The spike around T5 seems to be related to “Thank you”s (this is the victory speech) as well as “Guiliani”.

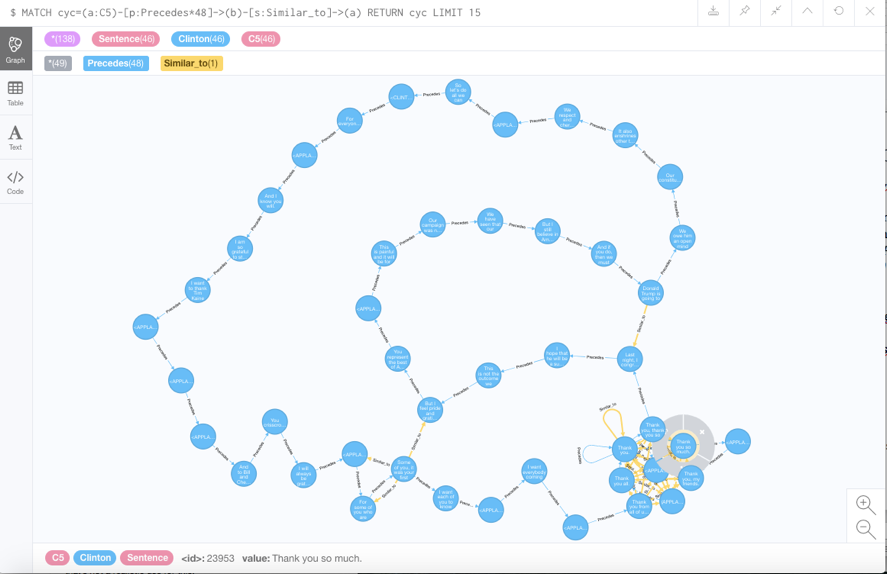
*MATCH cyc=(a:T5)-[p:Precedes\*100]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*

Spikes in Clinton speeches have similar behavior. There are a web of “thank you”s at the beginning of the speech which are be connected to again around 45 sentences later.

*MATCH cyc=(a:C5)-[p:Precedes\*6]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*

*MATCH cyc=(a:C5)-[p:Precedes\*48]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*

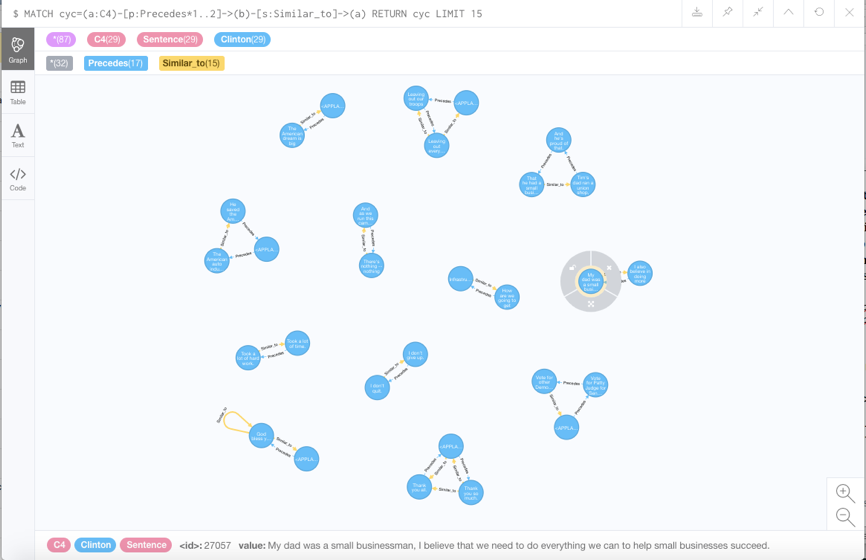


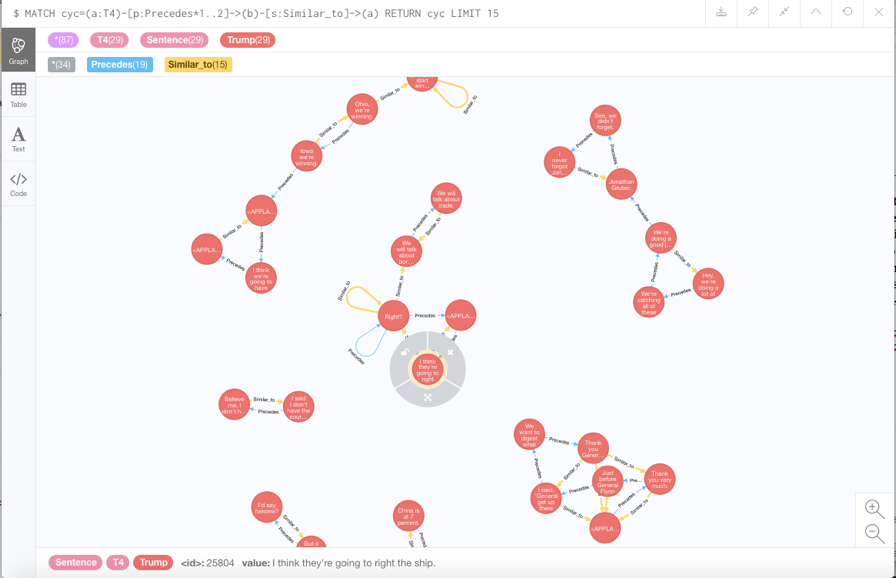


I’m also interested in seeing examples of immediate repetitions, proximate repetitions and determining if themes show up for longer cycles. For immediate repetitions, it does seem to me that Clinton’s next similar sentence more commonly reinforces the first while Trump’s more commonly emphasizes it. For example a Clinton couplet is: “I also believe in doing more to support small businesses :: My dad was a small businessman, I believe that we need to do everything we can to help small businesses succeed.” While one of Trump’s reads: “The FBI—I think they’re going to right the ship, folks. :: I think they’re going to right the ship”.

*MATCH cyc=(a:C4)-[p:Precedes\*1..2]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*

*MATCH cyc=(a:T4)-[p:Precedes\*1..2]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*





I define proximate repetitions as cycles with a length about the size of a paragraph. Good examples are “I alone can fix it” in C1:

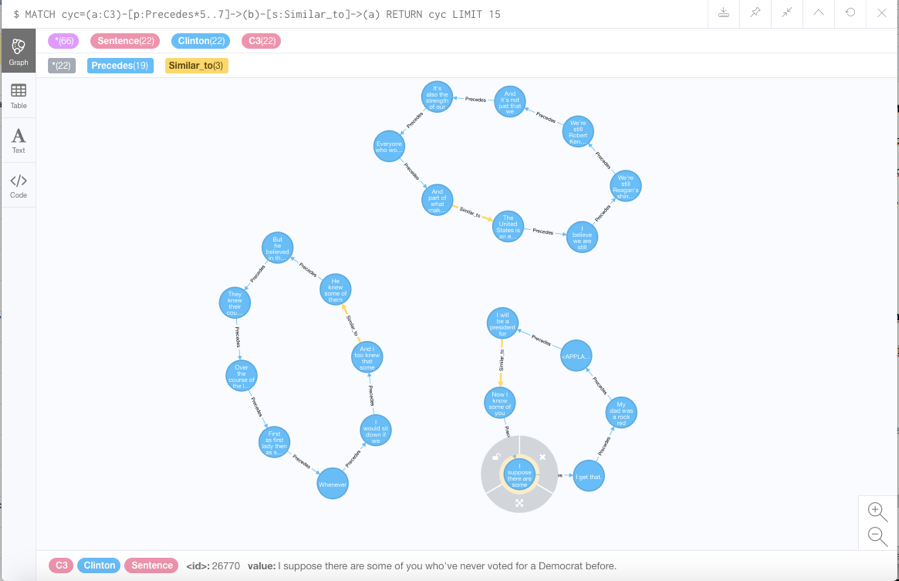
*MATCH cyc=(a:C1)-[p:Precedes\*5..7]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*

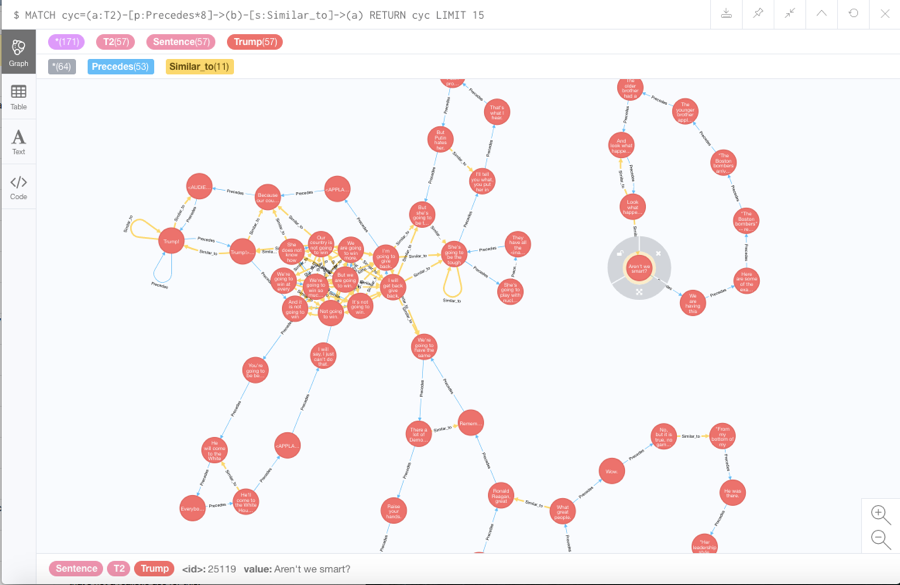
Every cycle in this view of C3:

MATCH cyc=(a:C3)-[p:Precedes\*5..7]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15

And “Arent we smart” in T2:

*MATCH cyc=(a:T2)-[p:Precedes\*8]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*



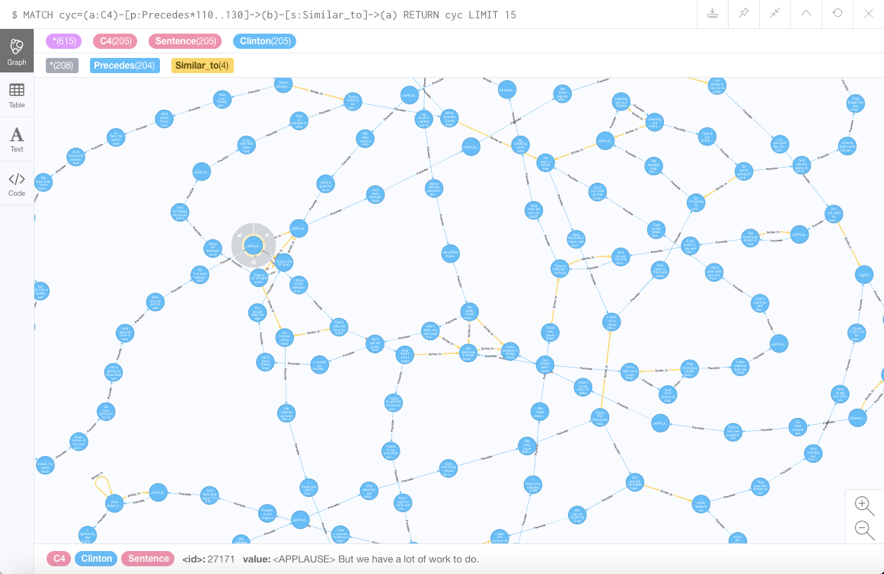


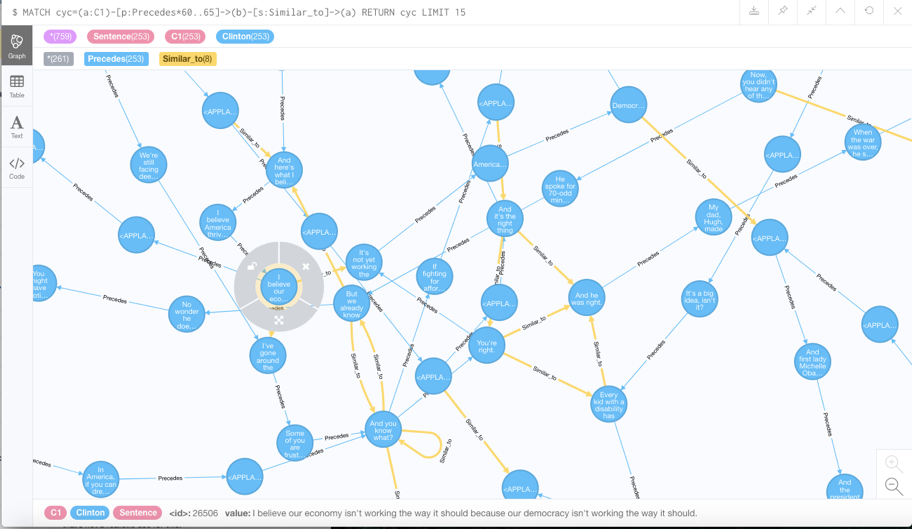
One of my hopes for this project was that cycles of around 20 would highlight minor themes. I checked a few views and didn’t find this to be the case. Could long cycles still highlight major themes or topics? I did see evidence of this in Clinton speeches. It certainly has to be searched for, most of the cycles are not thematically informative but see “lot of work to do” and “stronger together” in C4:

*MATCH cyc=(a:C4)-[p:Precedes\*110..130]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*

And “economy isn’t working the way it should” and “Donald Trump” in C1:

*MATCH cyc=(a:C1)-[p:Precedes\*60..65]->(b)-[s:Similar\_to]->(a) RETURN cyc LIMIT 15*





**Summary**

Donald Trump’s campaign speeches have many more sentences with similar words than Hillary Clinton’s. The results here were intuitively accurate but don’t tell us anything we didn’t already know. It would be interesting to compare texts where the differences are not so evident. These two are so different that it was hard to tune the model so that it worked for both Clinton and Trump. I think this method is a useful way to visualize different texts and see some of the structure but there is also a lot of noise. We found some good shorter cycles but the longer ones just seem to be good at finding those words you wish you removed in retrospect. The model gets obfuscated by repetitions of proper nouns and “Thank You”s. For any spike in cycle frequency there was a tight connection of similar sentences and then a couple of long loops that connect to that.

To take this further, I would like to control for this issue. Possibly I could merge similar consecutive sentences into one node, and add “thank” and “win” to stop words. I’d also like to explore different ways to encode for sentence meaning. I could try stemming, representing sentences as a sum of word2vec vectors or encode up a level on NLTKs WordNet to get more general than Bag-of-Words. Noun phrase chunking before the one-hot-encoding may also help.

I’d also like to explore Clinton speeches using a lower similarity threshold, and compare different texts. The Lucy dataset has essays from writers at all different levels. I’d be interested to see if common characteristics emerge at different levels of writing experience. Finally I’m interested to see if mediums can be identified, if there are attributes of verse and prose that stand out using this model.

**References**

* Brown, D. W. (2017) Clinton-Trump Corpus. Retrieved from http://www.thegrammarlab.com
* Browniee, Jason. (2017). How to Prepare Text Data for Machine Learning with scikit-learn. Retrieved from <https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/>
* Djordjević, Zoran, B. Relationships, Graph Databases, Neo4j [Slides].
* Djordjević, Zoran, B. Natural Language Processing [Slides].
* Neo4j, Inc. The Neo4j Developer Manual v3.3. (2017). Retrieved from <http://neo4j.com/docs/developer-manual>
* [Scikit-learn: Machine Learning in Python](http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html), Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
* Zhu, Xiaojin. Persistent Homology: An Introduction and a New Text Representation for Natural Language Processing. Retrieved from <http://pages.cs.wisc.edu/~jerryzhu/pub/homology.pdf>