Star Wars Analyse

Lukas Sebastian Hofmann - ERAS-201611324
Roberto Tejedor Moreno - 11896183P

Table of Contents

01

Introduction

You can enter a subtitle here if you need it

Overview

What?

- Star Wars (franchise) fantasy and inspiration for generations
- Different connections between movies and characters
- Undiscovered / Hidden knowledge

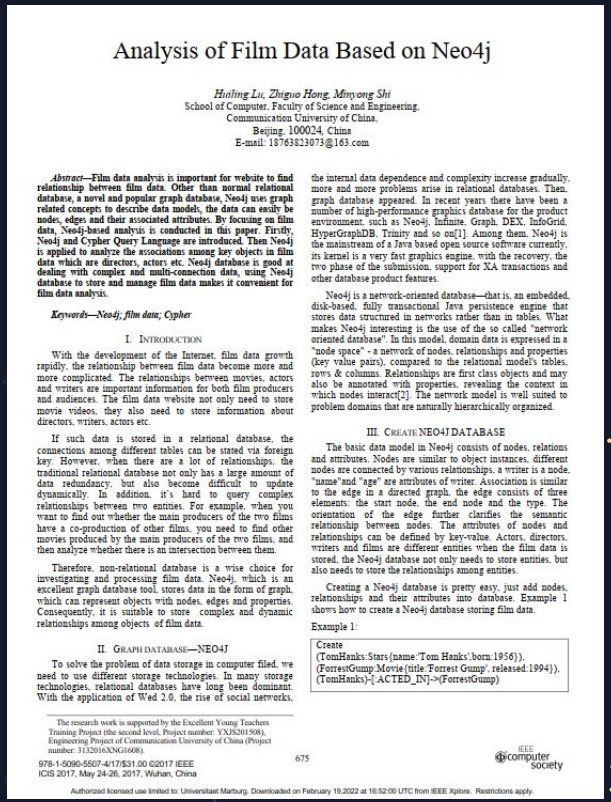
How?

- General scientific approach by Huiling Lu, Zhiguo Hong, Minyong Shi
- Specialization on a Star Wars social network (Graph-Structure / Semistructured)
- Improvements, validation and further development
- Dataset includes: 119 Nodes and 636 Relations in Neo4J

IEEE
Advancing Technology for Humanity

Paper-Overview

- Graph database to handle low structured data
- Neo4j is the most convenient
  - Good at dealing with complex and multi-connection data
  - Easy to visualize database
  - Data expressed in a network of nodes, relationships and properties
  - Labels to group nodes
- Cypher to extract information
  - Find patterns of relationships (MATCH)
  - Similar to SQL



Analysis of Film Data Based on Neo4j

Huiling Lu, Zhiguo Hong, Minyong Shi
School of Computer, Faculty of Science and Engineering,
Communication University of China,
Beijing, 100024, China
E-mail: 18763823073@163.com

***Abstract*—Film data analysis is important for website to find relationship between film data. Other than normal relational database, a novel and popular graph database, Neo4j uses graph related concepts to describe data models, the data can easily be nodes, edges and their associated attributes. By focusing on film data, Neo4j-based analysis is conducted in this paper. Firstly, Neo4j and Cypher Query Language are introduced. Then Neo4j is applied to analyze the associations among key objects in film data which are directors, actors etc. Neo4j database is good at dealing with complex and multi-connection data, using Neo4j database to store and manage film data makes it convenient for film data analysis.**

***Keywords*—Neo4j; film data; Cypher**

I. INTRODUCTION

With the development of the Internet, film data growth rapidly. the relationship between film data become more and more complicated. The relationships between movies, actors and writers are important information for both film producers and audiences. The film data website not only need to store movie videos, they also need to store information about directors, writers, actors etc.

If such data is stored in a relational database, the connections among different tables can be stated via foreign key. However, when there are a lot of relationships, the traditional relational database not only has a large amount of data redundancy, but also become difficult to update dynamically. In addition, it's hard to query complex relationships between two entities. For example, when you want to find out whether the main producers of the two films have a co-production of other films, you need to find other movies produced by the main producers of the two films, and then analyze whether there is an intersection between them.

Therefore, non-relational database is a wise choice for investigating and processing film data. Neo4j, which is an excellent graph database tool, stores data in the form of graph, which can represent objects with nodes, edges and properties. Consequently, it is suitable to store complex and dynamic relationships among objects of film data.

II. GRAPH DATABASE—NEO4J

To solve the problem of data storage in computer filed, we need to use different storage technologies. In many storage technologies, relational databases have long been dominant. With the application of Wed 2.0, the rise of social networks, the internal data dependence and complexity increase gradually, more and more problems arise in relational databases. Then, graph database appeared. In recent years there have been a number of high-performance graphics database for the product environment, such as Neo4j, Infinite, Graph, DEX, InfoGrid, HyperGraphDB, Trinity and so on[1]. Among them, Neo4j is the mainstream of a Java based open source software currently. its kernel is a very fast graphics engine, with the recovery, the two phase of the submission, support for XA transactions and other database product features.

Neo4j is a network-oriented database—that is, an embedded, disk-based, fully transactional Java persistence engine that stores data structured in networks rather than in tables. What makes Neo4j interesting is the use of the so called "network oriented database". In this model, domain data is expressed in a "node space" - a network of nodes, relationships and properties (key value pairs), compared to the relational model's tables, rows & columns. Relationships are first class objects and may also be annotated with properties, revealing the context in which nodes interact[2]. The network model is well suited to problem domains that are naturally hierarchically organized.

III. CREATE NEO4J DATABASE

The basic data model in Neo4j consists of nodes, relations and attributes. Nodes are similar to object instances, different nodes are connected by various relationships, a writer is a node, "name"and "age" are attributes of writer. Association is similar to the edge in a directed graph, the edge consists of three elements: the start node, the end node and the type. The orientation of the edge further clarifies the semantic relationship between nodes. The attributes of nodes and relationships can be defined by key-value. Actors, directors, writers and films are different entities when the film data is stored, the Neo4j database not only needs to store entities, but also needs to store the relationships among entities.

Creating a Neo4j database is pretty easy, just add nodes, relationships and their attributes into database. Example 1 shows how to create a Neo4j database storing film data.

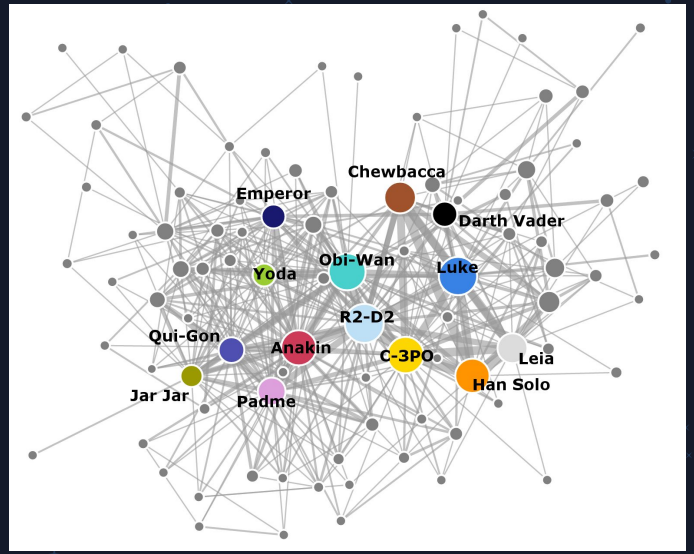
Example 1:

```
Create
(TomHanks:Stars{name:'Tom Hanks',born:1956}),
(ForrestGump:Movie{title:'Forrest Gump', released:1994}),
(TomHanks)-[:ACTED_IN]->(ForrestGump)
```

The research work is supported by the Excellent Young Teachers Training Project (the second level, Project number: YXJS201508), Engineering Project of Communication University of China (Project number: 3132016XNG1608).

978-1-5090-5507-4/17/$31.00 ©2017 IEEE
ICIS 2017, May 24-26, 2017, Wuhan, China

Existing Project-Overview



- Two different projects
  - The first six movies
  - All nine projects (this we used)
- Basic dataset build already in 2015
- Further development from different contributors
- A good example of a graph-network
- Representative also for different movies
- Representative for different problems
- We: Rebuild parts and have had adding functionality

02

Technologies

You can enter a subtitle here if you need it

Technologies

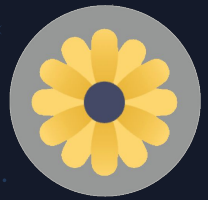
Python

Jupyter Notebook

Libraries

- Neo4j-driver
- Neo4jupyter
- Icypher
- Pandas/numpy
- Networkx
- Custom libraries

Neo4j Bloom

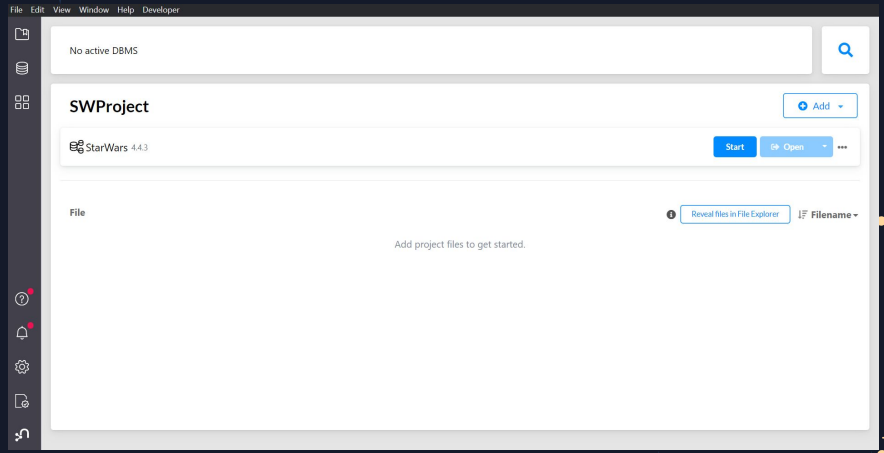


Neo4j Web Interface

Roberto

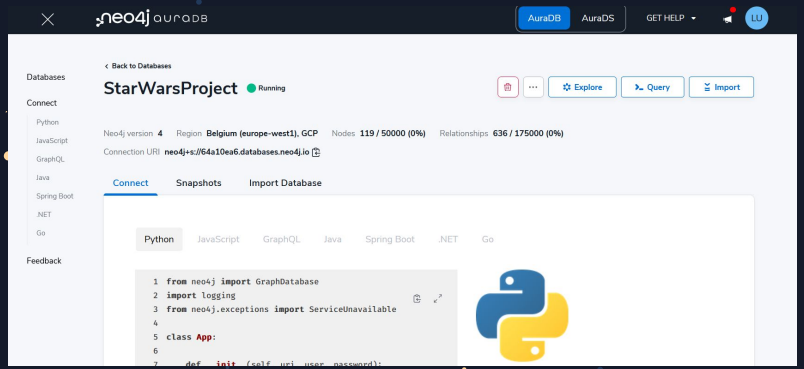
Neo4J@Desktop

- Desktop version of Neo4j allows to create a local database to interact with
- Some queries may take more time when executed locally
- Interactive display of query results

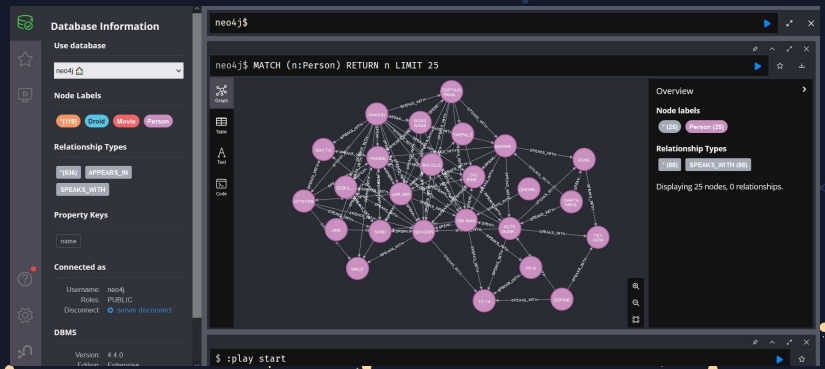


Neo4J@Azure

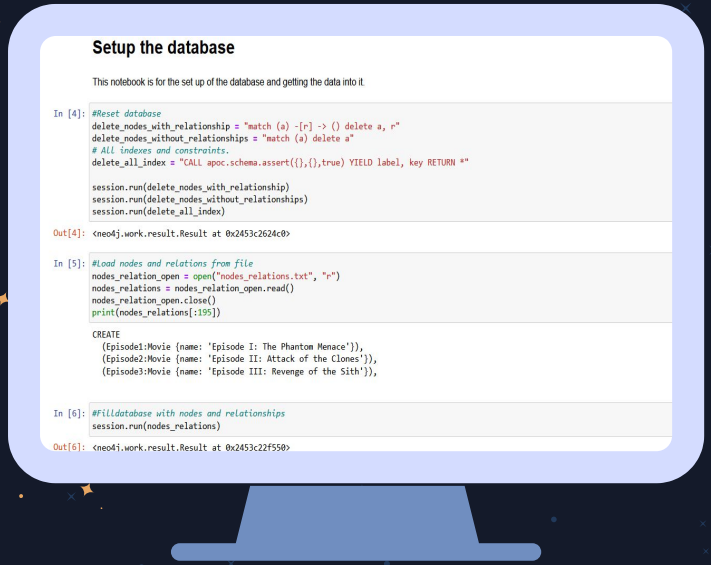
- Neo4j database on Azure cloud
- Much faster results than local mode
- Similar interface as Neo4j@Desktop
- Low migration problems



Genera web-interface



DB interface



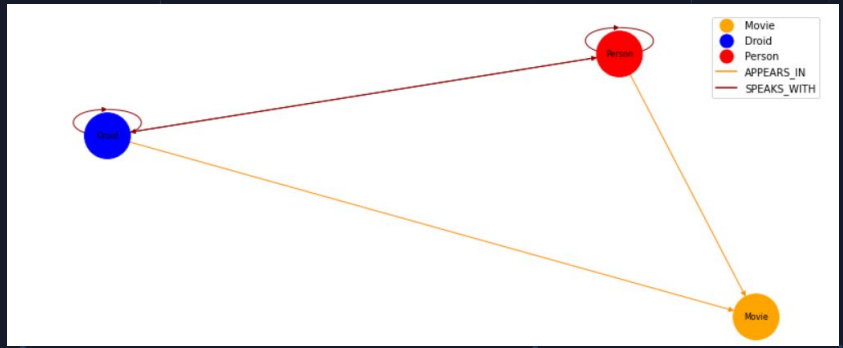
03

Implementation

- SetUp & Property Search
- Clustering
- Counting
- Recommandation
- Hidden Connections

SetUp & Property Search

- SetUp / Resetting / Deleting the entries
- Testing the data--import / database (structure)
- Getting familiar with the syntax
- Getting the basic attributes of the dataset
- Getting the basic relationship between different nodes



Clustering

- Not meant in the ML general meaning rather in the Neo4J context
  - Rather including:
    - Grouping
    - Classification
    - Clustering

```
#With cases
cluster_by_count_categories = """MATCH pattern=((m:Movie)-[a:APPEARS_IN]-(p:Person))
                                 with count(pattern=((m:Movie)-[a:APPEARS_IN]-(p:Person))) as connections, m
                                 Return
                                 CASE
                                   WHEN connections <= 20 THEN "Group 1: " + m.name
                                   WHEN connections <= 30      THEN "Groupe 2: " + m.name
                                   ELSE "Group 3: " + m.name
                                 END AS Clusters
                                 Order by connections"""
df = pd.DataFrame (session.read_transaction(
                                lambda tx: tx.run(cluster_by_count_categories).data()), columns = ['Clusters'])

df.head()
```

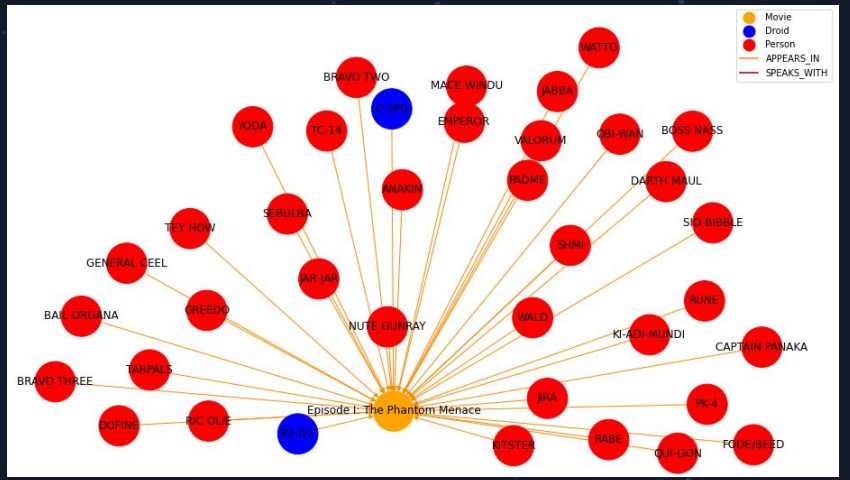
	Clusters
0	Group 1: Episode VI: Return of the Jedi
1	Group 1: Episode V: The Empire Strikes Back
2	Group 1: Episode IV: A New Hope
3	Groupe 2: Episode III: Revenge of the Sith
4	Groupe 2: Episode VII: The Force Awakens

Counting

- Number of nodes that achieve a condition
  - The character with most appearances
  - The movie with most characters
  - The node with most edges

```
most_appear = '''
    MATCH p=(n)-[:APPEARS_IN]->()
    WHERE n:Person OR n:Droid
    RETURN n.name AS Name, count(RELATIONSHIPS(p)) AS Appearances
    ORDER BY Appearances DESC, Name
'''
results = session.read_transaction(
        lambda tx: tx.run(most_appear).data())
df = pd.DataFrame(results)
df
```

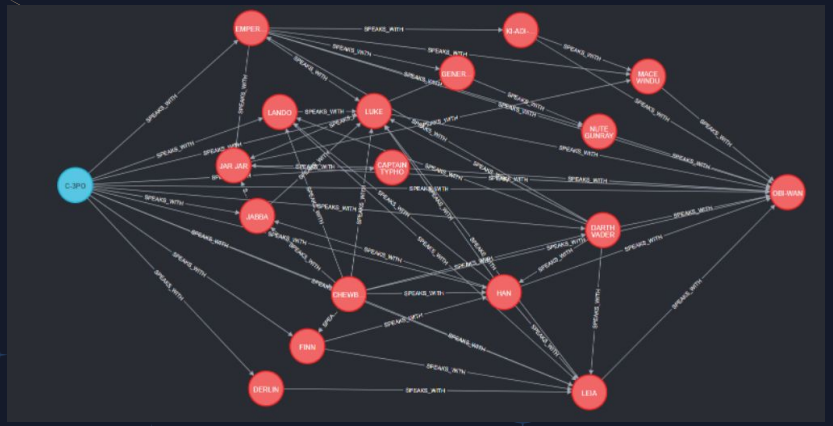
	Name	Appearances
0	C-3PO	7
1	R2-D2	7
2	OBI-WAN	6
3	EMPEROR	5



Recommendation

Important information that helps to recommend or not a character or movie

- Character “X” is good connected to character “Y”
- In which movies appears my favorite character?
- Who is the most important character?

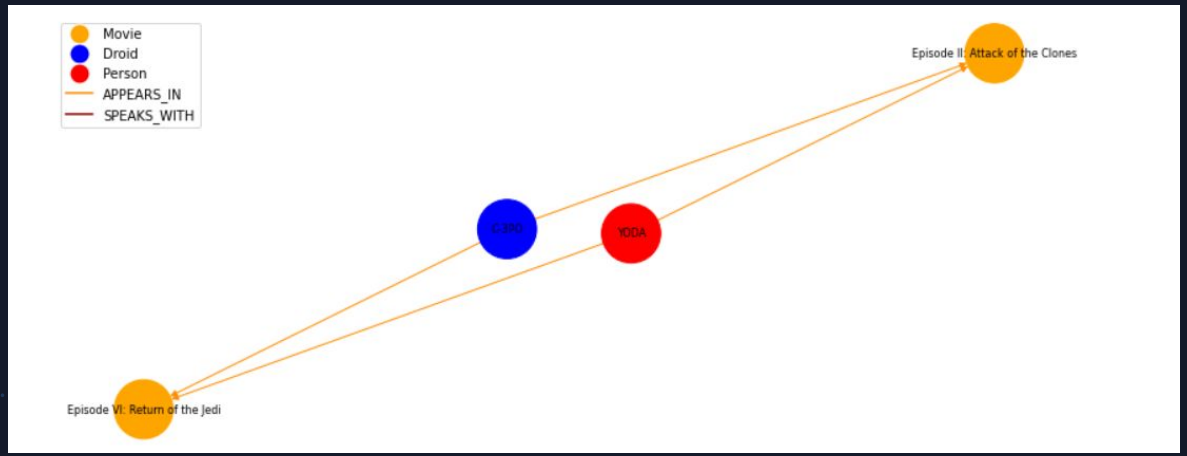



```
most_important = '''
    MATCH (ch)
    WHERE ch:Person OR ch:Droid
    CALL {
        WITH ch MATCH p=(ch)-->() RETURN count(p) AS occurrences
        UNION
        WITH ch MATCH p=(ch)<--() RETURN count(p) AS occurrences
    }
    WITH ch, collect(occurrences) AS occ
    WITH ch, occ[0] AS Outgoing, occ[1] AS Incoming, abs(occ[0]-occ[1]) AS Balance, occ[0]+occ[1] AS Total
    WHERE Total IS NOT NULL
    RETURN ch.name,  Outgoing, Incoming, Balance, Total
    ORDER BY Total DESC, Balance
'''
results = session.read_transaction(
        lambda tx: tx.run(most_important).data())
df = pd.DataFrame(results)
df.head()
```

	ch.name	Outgoing	Incoming	Balance	Total
0	ANAKIN	46	0	46	46
1	OBI-WAN	21	22	1	43
2	C-3PO	33	10	23	43
3	PADME	14	23	9	37
4	LUKE	14	17	3	31

Hidden connections

- Path between
  - Character and Character
  - Character and Movie
  - Movie and Movie
- Hidden Path Finding of the paper could not be proved



```
match movie1
where movie1.title="The Shawshank Redemption"
match movie2
where movie2.title="Forrest Gump"
match
searchPath=movie1<-[*]-(people1)-[*]->movie-[*]-(people
2)-[*]->movie2
return  distinct movie.title,searchPath
```

return 1 line, times 20 ms

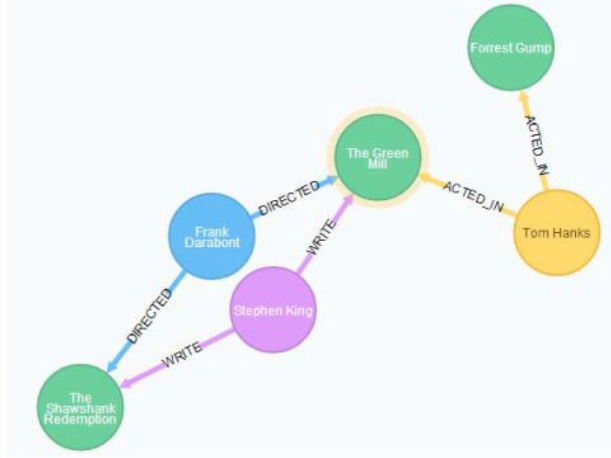
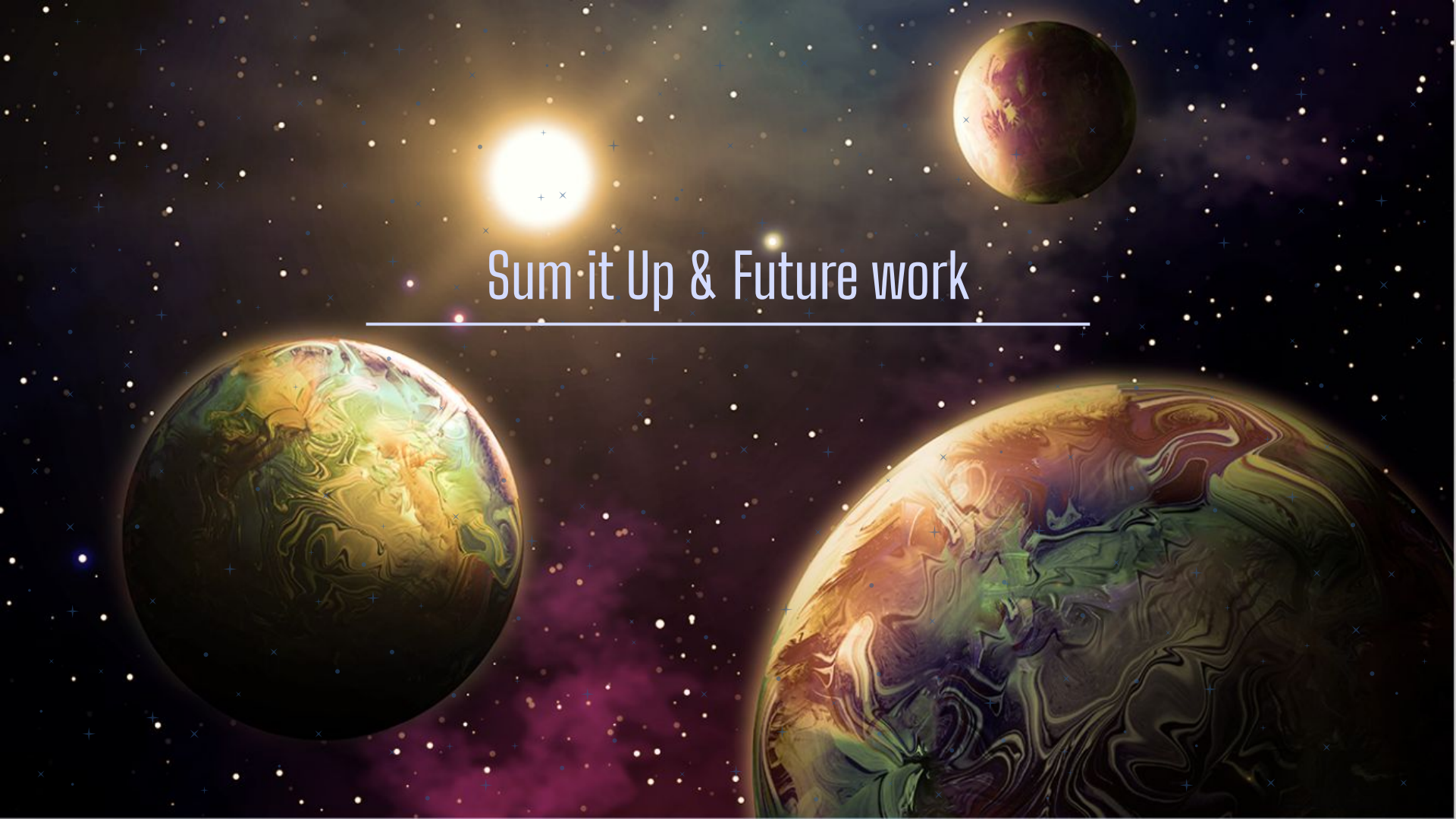


Figure 4. Query result

In our case this query is not “working” and even in the Azure Cloud we run out of memory problems. In our dataset there are just too many Nodes / Connections. Due to this presented approach is not general useful and maybe just works on a smaller dataset. It needs a limitation and enumeration over the results.

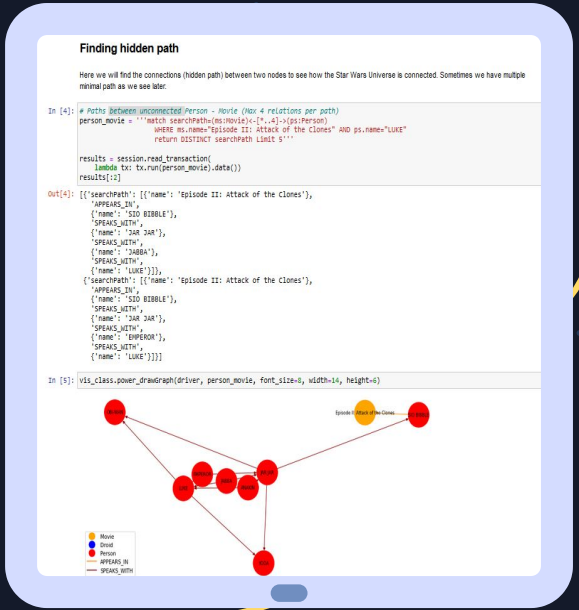
Let’s go to the code

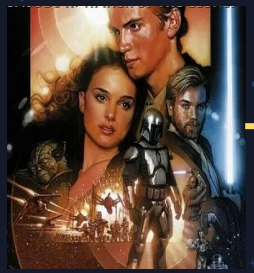
Sum it Up & Future work



Conclusion of our project

- Powerful general graph database (Neo4J)
- However, a some functionalities have to be developed (like visualisation)
- The paper is in the most points correct but also has “bugs”
- Developed functionality which everyone can adapt
- Knowledge-discovery in Star Wars





Future Work

Creating different nodes/relationships e.g.

- To games
- To Books / Comics
- To audibles
- Add conversations (text) and context

→ Enhancing data

Adding nodes and relationship properties

- Age
- Title
- Said text
- Relevance

→ For deeper data analyses and ML

Implement an interactive Web- interface

- User creates queries (non-code)
- Add/Delete/Modify dataset

→ Easier access for non-programmer

Q&A

End of the presentation

Thanks for your attention

The force is with you (all)
Celebrate the 4 of May

The team

Lukas Sebastian Hofmann

Roberto Tejedor Moreno

https://github.com/SoftwareStackLukas/StarWarsAnalyse (Our Repo)

Sources

- https://neo4j.com/
- http://evelinag.com/blog/2015/12-15-star-wars-social-network/index.html
- https://neo4j.com/graphgists/exploring-the-star-wars-social-network/
- https://github.com/merqurio/neo4jupyter
- https://ieeexplore.ieee.org/document/7960078