

Vulkanausbruch auf Palma



Quelle: [Spiegel](#)

Der Herr der Ringe und Wikidata

- Blogpost: <https://towardsdatascience.com/lord-of-the-wiki-ring-importing-wikidata-into-neo4j-and-analyzing-family-trees-da27f64d675e>
- Code auf Github:
https://github.com/tomasonjo/blogs/blob/master/Lord_of_the_wikidata/Part1_Importing_Wikidata_into_Neo4j_and_analyzing_family_trees.ipynb

```
// 00 Alles löschen
CALL apoc.periodic.iterate('MATCH (n) RETURN n',
'DETACH DELETE n', {batchSize:1000});
CALL apoc.schema.assert({}, {}, true)
YIELD label, key RETURN *;
```

Gliederung

- Datenimport von Herr der Ringe aus Wikidata in Neo4j
- Datenanalyse der importierten Daten
- Ergänzung der Daten
- Analyse mit Graphalgorithmen
 - Weakly connected component
 - Betweenness centrality

Graph Schema anzeigen

```
CALL db.schema.visualization();
```

WikiDataimport

```
?item wdt:P31 wd:Q989255
```

- “Wir möchten ein Element abrufen, das eine Instanz von (wdt:P31) einer Entität mit der ID Q989255 ist”.

```
SELECT ?item ?itemLabel WHERE
  { ?item wdt:P31 wd:Q989255 .
SERVICE wikibase:label
  { bd:serviceParam wikibase:language
    "[AUTO_LANGUAGE],en" } }
```

SPARQL Query und Neo4j Import

```
// Prepare a SPARQL query
WITH 'SELECT ?item ?itemLabel
      WHERE{
        ?item wdt:P31 wd:Q989255 .
        SERVICE wikibase:label { bd:serviceParam wikibase:language
          "[AUTO_LANGUAGE],en" }
      }' AS sparql
// make a request to Wikidata
CALL apoc.load.jsonParams(
      "https://query.wikidata.org/sparql?query=" +
      apoc.text.urlencode(sparql),
      { Accept: "application/sparql-results+json"}, null)
YIELD value
// Unwind results to row
UNWIND value['results']['bindings'] as row
```

Import der ethnischen Zugehörigkeiten

```
// Iterate over each race in graph
MATCH (r:Race)
// Prepare a SparQL query
WITH 'SELECT ?item ?itemLabel
      WHERE {
          ?item wdt:P31 wd:'' + r.id + ' .
          SERVICE wikibase:label { bd:serviceParam wikibase:label
            "[AUTO_LANGUAGE],en" }
      }' AS sparql, r
// make a request to Wikidata
CALL apoc.load.jsonParams( "https://query.wikidata.org/sparql"
    { Accept: "application/sparql-results+json"}, null)
YIELD value
UNWIND value['results']['bindings'] as row
WITH row['itemlabel']['value'] as name
```

Anzahl der Mitglieder der jeweiligen ethnischen Gruppe

```
MATCH (r:Race)
RETURN r.race as race,
       size((r)-[:BELONG_TO]-()) as members
ORDER BY members DESC
LIMIT 10;
```

Ergänzung des Graphen (Geschlecht, Todesursache etc.)

```
// Iterate over characters
MATCH (r:Character)
// Prepare a SparQL query
WITH 'SELECT *
      WHERE{
        ?item rdfs:label ?name .
        filter (?item = wd:' + r.id + ')
        filter (lang(?name) = "en" ) .
      OPTIONAL{
        ?item wdt:P21 [rdfs:label ?gender] .
        filter (lang(?gender)="en")
      }
      OPTIONAL{
        ?item wdt:P27 [rdfs:label ?country] .
        filter (lang(?country)="en")
      }
    ' AS query
RETURN query
```

NULL-Werte in den Daten und der MERGE-Befehl

- Ergebnisse aus Wikidata können auch NULL enthalten.
- FOREACH Trick:

```
FOREACH(ignoreme in case when r  
row['sibling'] is not null then [1]  
else [] end |  
MERGE (c:Character{url:row['sibling']['value']});
```

Todesursachen

```
MATCH (n:Character)
WHERE exists (n.manner_of_death)
RETURN n.manner_of_death as
manner_of_death, count(*) as count;
```

Mutterland und Vaterland

```
MATCH (c:Country)
RETURN c.name as country,
       size((c)-[:IN_COUNTRY]-()) as members
ORDER BY members
DESC LIMIT 10;
```

Ergänzung von Verwandtschaft aus Wikidata

```
// Iterate over characters
MATCH (r:Character)
WITH 'SELECT *
      WHERE{
        ?item rdfs:label ?name .
        filter (?item = wd:' + r.id + ' )
        filter (lang(?name) = "en" ) .
      OPTIONAL{
        ?item wdt:P22 ?father
      }
      OPTIONAL{
        ?item wdt:P25 ?mother
      }
      OPTIONAL{
        ?item wdt:P1038 ?relative
```

Mehrfachehen

```
MATCH p=() - [:SPOUSE] - () - [:SPOUSE] - ()  
RETURN p LIMIT 10;
```

Kinder mit mehreren Partnern

```
MATCH (c:Character)<-[:HAS_FATHER|HAS_MOTHER]-()-[:HAS_FATHER|HAS_MOTHER]
WITH c, collect(distinct other) as others
WHERE size(others) > 1
MATCH p=(c)<-[:HAS_FATHER|HAS_MOTHER]-()-[:HAS_FATHER|HAS_MOTHER]
RETURN p;
```

Datenqualität: Geschwister und Nichtgeschwister

```
MATCH p=(a:Character)-[:HAS_FATHER|:HAS_MOTHER]->()  
<-[:HAS_FATHER|:HAS_MOTHER]-(b:Character)  
WHERE NOT (a)-[:SIBLING]-(b)  
RETURN p  
LIMIT 5;
```

Geschwisterkante ergänzen

```
MATCH p=(a:Character)-[:HAS_FATHER|:HAS_MOTHER]->()  
<-[:HAS_FATHER|:HAS_MOTHER]-(b:Character)  
WHERE NOT (a)-[:SIBLING]-(b)  
MERGE (a)-[:SIBLING]-(b);
```

Länderzugehörigkeit für die “neuen” Geschwister

```
MATCH (country)<-[:IN_COUNTRY]-(s:Character)-  
[:SIBLING]-(t:Character) WHERE NOT (t)-[:IN_COUNTRY]->()  
MERGE (t)-[:IN_COUNTRY]->(country);
```

Beruf, Sprache, Gruppen und Ereignisse

```
MATCH (r:Character)
WHERE exists (r.id)
WITH 'SELECT *
      WHERE{
        ?item rdfs:label ?name .
        filter (?item = wd:' + r.id + ')
        filter (lang(?name) = "en" ) .
      OPTIONAL {
        ?item wdt:P106 [rdfs:label ?occupation ] .
        filter (lang(?occupation) = "en" ).
      }
      OPTIONAL {
        ?item wdt:P103 [rdfs:label ?language ] .
        filter (lang(?language) = "en" ) .
      }
    }
```

Gruppen und Berufe

```
MATCH (n:Group)<-[:MEMBER_OF]-(c)
OPTIONAL MATCH (c)-[:HAS_OCCUPATION]->(o)
RETURN n.name as group,
       count(*) as size,
       collect(c.name)[..3] as members,
       collect(distinct o.name)[..3] as occupations
ORDER BY size DESC;
```

Feinde und weitere Gegenstände

```
MATCH (r:Character)
WHERE exists (r.id)
WITH 'SELECT *
      WHERE { ?item rdfs:label ?name .
              filter (?item = wd:' + r.id + ' )
              filter (lang(?name) = "en" ) .
            OPTIONAL{ ?item wdt:P1830 [rdfs:label ?owner ] .
                      filter (lang(?owner) = "en" ). }
            OPTIONAL{ ?item wdt:P7047 ?enemy } }' AS sparql, r
CALL apoc.load.jsonParams( "https://query.wikidata.org/sparql"
                          , apoc.text.urlencode(sparql)
                          , { Accept: "application/sparql-res"
YIELD value
WITH value, r
WHERE value['results']['bindings'] <> []
```

Feindschaft in der direkten Familie

```
MATCH p=(a) - [:SPOUSE|SIBLING|HAS_FATHER|HAS_MOTHER] - (b)
WHERE (a) - [:ENEMY] - (b)
RETURN p;
```

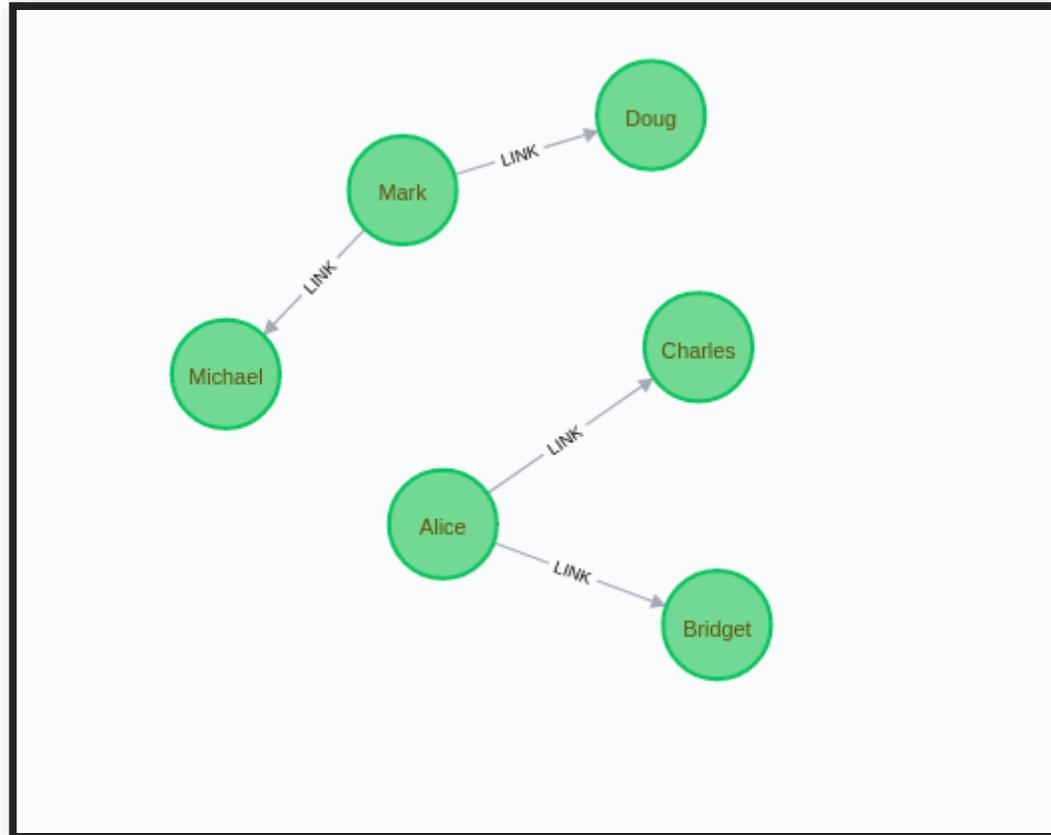
Feindschaft in der erweiterten Familie

```
MATCH p=(a) - [:SPOUSE|SIBLING|HAS_FATHER|HAS_MOTHER*..2] - (b)
WHERE (a) - [:ENEMY] - (b)
RETURN p;
```

Familiengraph

```
// Erstellen der Graphprojektion  
CALL gds.graph.create('family', 'Character',  
    ['SPOUSE', 'SIBLING', 'HAS_FATHER', 'HAS_MOTHER']);  
  
// Löschen der Graphprojektion  
CALL gds.graph.drop('family');
```

Weakly Connected Component (WCC)



Weakly Connected Component (WCC)

```
CALL gds.wcc.stats('family')
YIELD componentCount, componentDistribution
RETURN componentCount as components,
        componentDistribution.p75 as p75,
        componentDistribution.p90 as p90,
        apoc.math.round(componentDistribution.mean,2) as mean
        componentDistribution.max as max;
```

WCC Ergebnisse in den Graphen schreiben

```
CALL gds.wcc.write('family',  
  {writeProperty:'familyComponent'});
```

Gruppen im Graphen und ethnische Zugehörigkeiten

```
MATCH (c:Character)
OPTIONAL MATCH (c)-[:BELONG_TO]->(race)
WITH c.familyComponent as familyComponent,
     count(*) as size,
     collect(c.name) as members,
     collect(distinct race.race) as family_race
ORDER BY size DESC LIMIT 5
RETURN familyComponent,
        size,
        members[..3] as random_members,
        family_race;
```

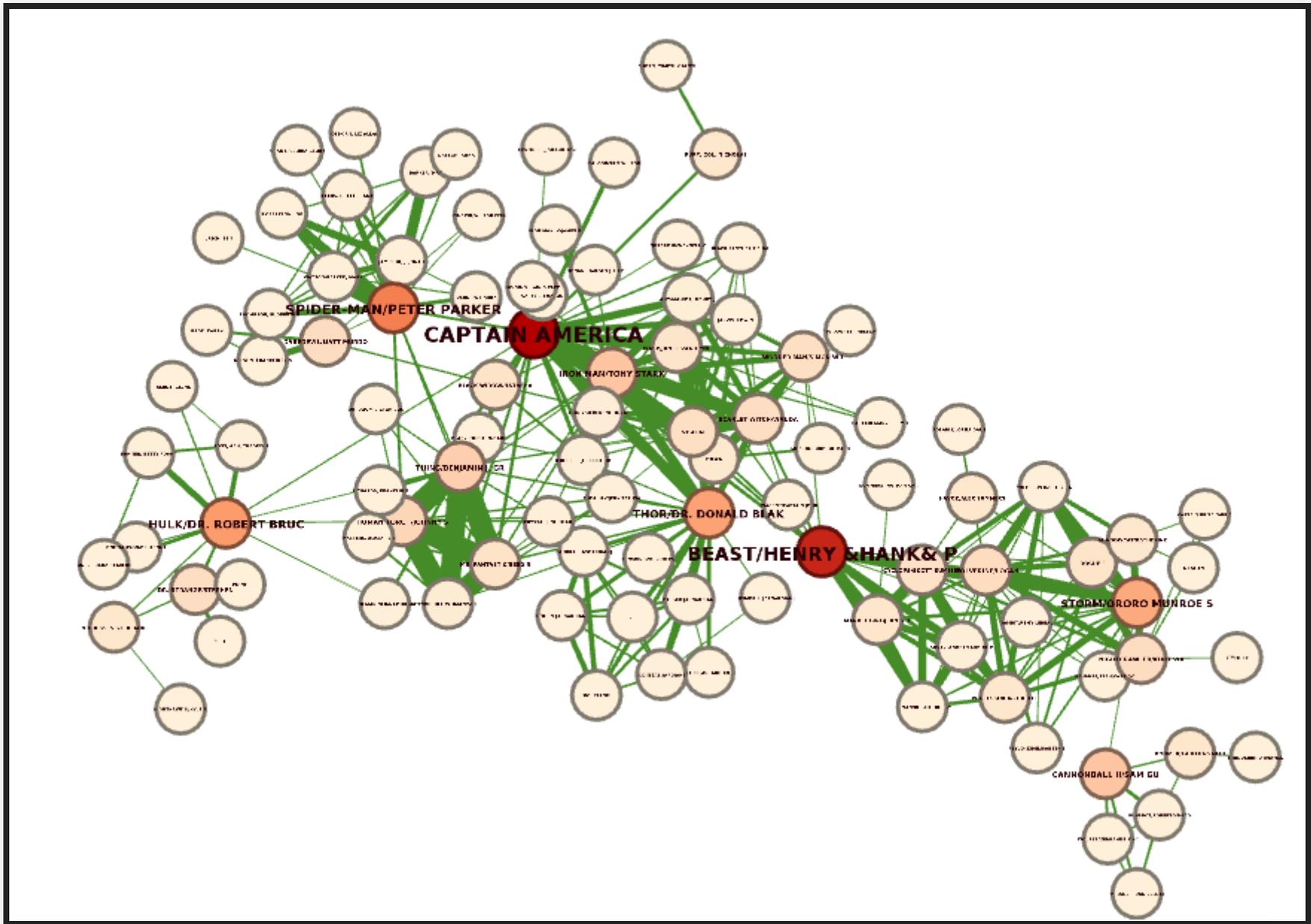
Heirat zwischen verschiedenen ethnischen Gruppen

```
MATCH (c:Character)
WHERE c.familyComponent = 9 // fix the family component
MATCH p=(race)<-[:BELONG_TO]-(c)-[:SPOUSE]-(other)-[:BELONG_TO]-
WHERE race <> other_race AND id(c) > id(other)
RETURN c.name as spouse_1,
       race.race as race_1,
       other.name as spouse_2,
       other_race.race as race_2;
```

Die Vorfahren der Elfen

```
MATCH (c:Character)
WHERE (c)-[:BELONG_TO]->(:Race{race:'half-elven'})
MATCH p=(c)<-[:HAS_FATHER|HAS_MOTHER*..20]-(end)
WHERE NOT (end)<-[:HAS_FATHER|HAS_MOTHER]-()
WITH c, max(length(p)) as descendants
ORDER BY descendants DESC
LIMIT 5
RETURN c.name as character,
       descendants;
```

Betweenness centrality am Beispiel von Captain America



Betweenness centrality

```
CALL gds.alpha.betweenness.stream({
nodeQuery:"MATCH (n:Character)
WHERE n.familyComponent = 9 // fix component number !
RETURN id(n) as id",
relationshipQuery:"MATCH (s:Character)-[:HAS_FATHER|HAS_MOTHER]
RETURN id(s) as source, id(t) as target",
validateRelationships:false})
YIELD nodeId, centrality
RETURN gds.util.asNode(nodeId).name as character,
centrality
ORDER BY centrality DESC LIMIT 10;
```

Geschichtswissenschaften

Zusammenfassung

- Wikidata als Quelle für Informationen in den digitalen Geisteswissenschaften
- Daten- und Informationsmodellierung
- Digitale Quellenkritik
- Algorithmenkritik
- Perspektiven

Quellen

Blogpost zu LOTR in Neo4j

Zugehöriges Github-Repo

Präsentation

<http://jlu-buster.mni.thm.de:9550/browser/>

<http://jlu-buster.mni.thm.de:9570/browser/>

<https://git.thm.de/aksz15/images/-/blob/master>