CSE 590 Term Project
Marvel Universe Assemble! . . . in a Graph Database

Aliaa Elshamekh

January 26, 2021

1 Introduction

Marvel Comics, originally called Timely Comics Inc., has been publishing comic books for several decades. "The Golden Age of Comics" name that was given due to the popularity of the books during the first years, was later followed by a period of decline of interest in superhero stories due to World War ref. In 1961, Marvel relaunched its superhero comic books publishing line. This new era started what has been known as the Marvel Age of Comics. Characters created during this period such as Spider-Man, the Hulk, the Fantastic Four, and the X-Men, together with those created during the Golden Age such as Captain America, are known worldwide and have become cultural icons during the last decades. Later, Marvel's characters popularity has been revitalized even more due to the release of several recent movies which recreate the comic books using spectacular modern special effects. Nowadays, it is possible to access the content of the comic books via a digital platform created by Marvel.

More information about the Marvel Universe can be found here.

The Marvel Comics character collaboration graph was originally constructed by Cesc Rosselló, Ricardo Alberich, and Joe Miro from the University of the Balearic Islands. They compare the characteristics of this universe to real-world collaboration networks, such as the Hollywood network, or the one created by scientists who work together in producing research papers [1].

The Marvel Universe network (MU) is defined as the network whose nodes are significant Marvel characters and where two characters are linked when they jointly appear in a significant way in the same comic book. Here, It is only considered comics published after Issue 1 of Fantastic Four (dated November 1961), which is understood as the point of departure of the Marvel Age of Comics.

Any study like this one must be based on a database, which puts the main restriction to its scope. In this case, the database we have used is the Marvel Chronology Project (MCP), which, according to its creator, R. Chappell [2], catalogs every canonical appearance by every significant Marvel character. Thus, the "significant characters" represented by nodes in our network and the "significant appearances" that yield the links in it are, actually, nothing but those characters and appearances currently included in the MCP database. Nevertheless, all in all, this database collects over 96,000 appearances by more than 6,500 characters in about 13,000 comic books, and thus yields quite a complete picture of the Marvel Universe.

2 Dataset Analysis and Graph Data Model Design

We will use the marvel social network data, which was downloaded from a kaggle competition post. There are 3 CSVs available.

- nodes.csv — Node name and type.
- edges.csv — Heroes and the comic in which they appear.
- hero-network.csv — Edges between heroes that appear in the same comic.

We only need the **edges.csv** file, because if we know which hero appeared in which comic, that is enough for us to import the social graph. We also do not need the hero-network.csv, that hold information about unweighted hero network, because we will create our own weighted hero network with the help of cypher and apoc.

I am going to show some results in Sandbox Bloom so that I can have more friendly user interface.

The graph data is simple. It consists if two nodes :

- Hero.
- Comic

With one relationship :

- APPEARED_IN which is between the two nodes mentioned above.

At the begining the data model design is as shown in figure 1

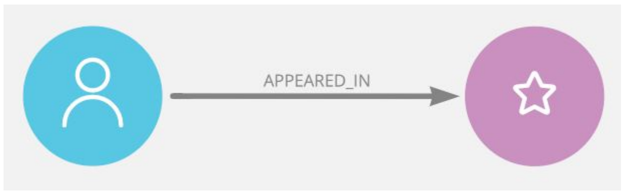


Figure 1: Data model for Marvelous Universe data set

3 Graph Database Implementation and Import

- Create of the nodes inside Neo4j .

```
CALL apoc.schema.assert(
{},
{Comic:['name'],Hero:['name']})
```

- Importing the file **edges.csv** into Neo4j and create **APPEARED_IN** relationship

```
LOAD CSV WITH HEADERS FROM
"file:///edges.csv" as row
MERGE (h:Hero{name:row.hero})
MERGE (c:Comic{name:row.comic})
MERGE (h)-[:APPEARED_IN]->(c)
```

- Size of the Marvelous graph database.

Graph details	
Name	Count
Hero	6,439
Comic	12,651
APPEARED_IN	96,104

In figure 2 we can see sample from the output for Heroes in Comic *AA2 35*.

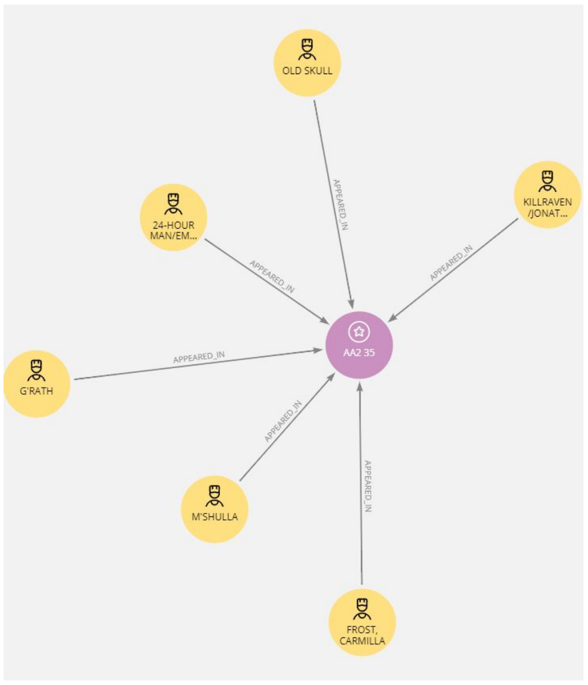
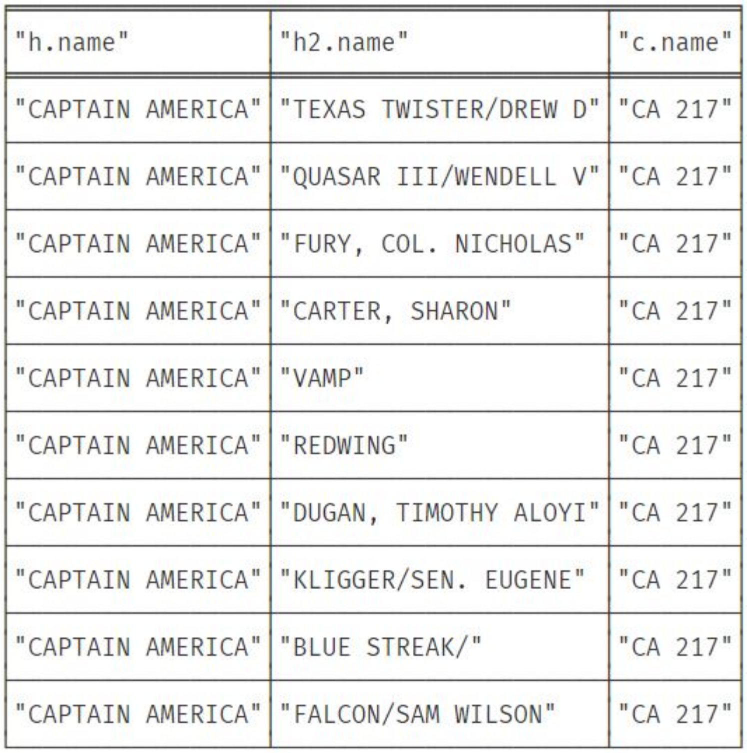


Figure 2: Sample from the graph database

4 Exploration of Graph Database

- **Heroes that appeared in the same commic with the *CAPTIN AMERICA***

```
MATCH (h:Hero{name:"CAPTAIN AMERICA"})-[a:APPEARED_IN]-(c:Comic)-[r:APPEARED_IN]-(h2:Hero)
 RETURN h.name,h2.name,c.name
 LIMIT 10
```



"h.name"	"h2.name"	"c.name"
"CAPTAIN AMERICA"	"TEXAS TWISTER/DREW D"	"CA 217"
"CAPTAIN AMERICA"	"QUASAR III/WENDELL V"	"CA 217"
"CAPTAIN AMERICA"	"FURY, COL. NICHOLAS"	"CA 217"
"CAPTAIN AMERICA"	"CARTER, SHARON"	"CA 217"
"CAPTAIN AMERICA"	"VAMP"	"CA 217"
"CAPTAIN AMERICA"	"REDWING"	"CA 217"
"CAPTAIN AMERICA"	"DUGAN, TIMOTHY ALOYI"	"CA 217"
"CAPTAIN AMERICA"	"KLIGGER/SEN. EUGENE"	"CA 217"
"CAPTAIN AMERICA"	"BLUE STREAK/"	"CA 217"
"CAPTAIN AMERICA"	"FALCON/SAM WILSON"	"CA 217"

Figure 3: 10 Heroes that appeared with CAPTAIN AMERICA in the same comic

- **Create a weighted un-directed hero network**
  Create a weighted hero network, where we will assume, that ***the more comics hero appeared together , the more they know each other.***
  We will save the number of comics heroes appeared together as a weight property of the **KNOWS** relationship.

  We are going to use the **apoc.periodic.iterate** to help us with batching. As our graph gets bigger, we do not want run global refactoring or calculations in one transaction, but we want to batch it. Apoc comes in handy when using batching with cypher. To create a weighted un-directed hero social graph using the following query :

```
CALL apoc.periodic.iterate(
"MATCH (p1:Hero)-->(:Comic)<--(p2:Hero) where id(p1) < id(p2) RETURN p1,p2",
"MERGE (p1)-[r:KNOWS]-(p2) ON CREATE SET r.weight = 1 ON MATCH SET r.weight = r.weight + 1"
, {batchSize:5000, parallel:false,iterateList:true})
```

  The graph model will be as in figure 4 after creating KNOWS relationship.

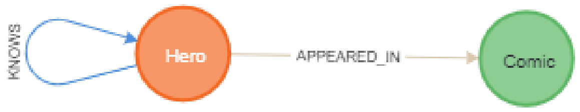
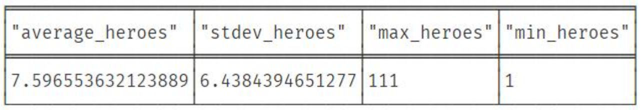


Figure 4: Updated data model for Marvelous Universe data set

- Matching two heroes , that appeared in the same comic

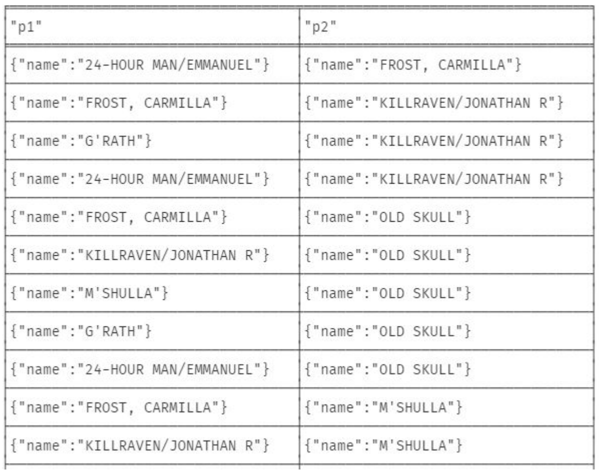
```
MATCH (p1:Hero)-->(:Comic)<--(p2:Hero)
WHERE id(p1) < id(p2)
RETURN p1,p2
```



"average_heroes"	"stdev_heroes"	"max_heroes"	"min_heroes"
7.596553632123889	6.4384394651277	111	1

Figure 5: Degree statistics summary analysis

The output sample is as shown in 6



"p1"	"p2"
{"name":"24-HOUR MAN/EMMANUEL"}	{"name":"FROST, CARMILLA"}
{"name":"FROST, CARMILLA"}	{"name":"KILLRAVEN/JONATHAN R"}
{"name":"G'RATH"}	{"name":"KILLRAVEN/JONATHAN R"}
{"name":"24-HOUR MAN/EMMANUEL"}	{"name":"KILLRAVEN/JONATHAN R"}
{"name":"FROST, CARMILLA"}	{"name":"OLD SKULL"}
{"name":"KILLRAVEN/JONATHAN R"}	{"name":"OLD SKULL"}
{"name":"M'SHULLA"}	{"name":"OLD SKULL"}
{"name":"G'RATH"}	{"name":"OLD SKULL"}
{"name":"24-HOUR MAN/EMMANUEL"}	{"name":"OLD SKULL"}
{"name":"FROST, CARMILLA"}	{"name":"M'SHULLA"}
{"name":"KILLRAVEN/JONATHAN R"}	{"name":"M'SHULLA"}

Figure 6: Matching two heroes , that appeared in the same comic output sample

- Degree of nodes in the Marvelous Universe graph.

```
MATCH (c:Comic)
RETURN avg(apoc.node.degree(c,'APPEARED_IN'))AS average_heroes,
stdev(apoc.node.degree(c,'APPEARED_IN' )) AS stdev_heroes,
max(apoc.node.degree(c,'APPEARED_IN'))AS max_heroes,
min(apoc.node.degree(c,'APPEARED_IN')) AS min_heroes
```

- Process on node at a time to update the weight of KNOWS relationships between heroes.

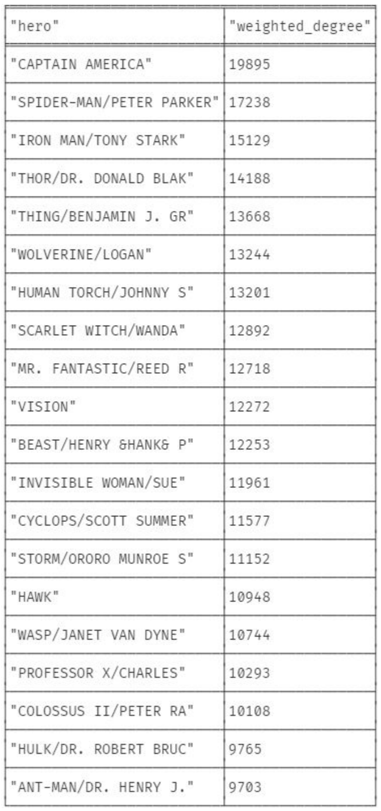
  ```
  MERGE (p1)-[r:KNOWS]-(p2)
  ON CREATE SET r.weight = 1
  ON MATCH SET r.weight = r.weight + 1
  ```

  we do not specify the direction of the relationship. We treat this network as un-directed, so the direction of relationship has no implication.We use MERGE to create the relationship. It means that it will try to match the relationship and if it doesn't find one it will create one.

- Heroes with top weighted degree.

  ```
  MATCH (h:Hero)-[t:KNOWS]-()
  RETURN h.name AS hero,sum(t.weight) AS weighted_degree
  ORDER BY weighted_degree DESC LIMIT 20
  ```

  The weighted degree is simply is the sum of all weights of the relationships connected to given node. The top 25 heroes with highest degree as as shown in figure 7



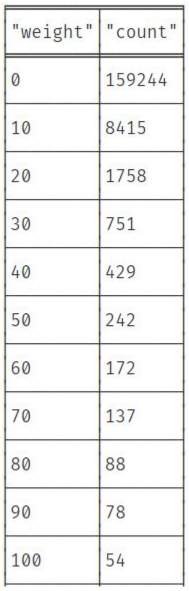
"hero"	"weighted_degree"
"CAPTAIN AMERICA"	19895
"SPIDER-MAN/PETER PARKER"	17238
"IRON MAN/TONY STARK"	15129
"THOR/DR. DONALD BLAK"	14188
"THING/BENJAMIN J. GR"	13668
"WOLVERINE/LOGAN"	13244
"HUMAN TORCH/JOHNNY S"	13201
"SCARLET WITCH/WANDA"	12892
"MR. FANTASTIC/REED R"	12718
"VISION"	12272
"BEAST/HENRY &HANK& P"	12253
"INVISIBLE WOMAN/SUE"	11961
"CYCLOPS/SCOTT SUMMER"	11577
"STORM/ORORO MUNROE S"	11152
"HAWK"	10948
"WASP/JANET VAN DYNE"	10744
"PROFESSOR X/CHARLES"	10293
"COLOSSUS II/PETER RA"	10108
"HULK/DR. ROBERT BRUC"	9765
"ANT-MAN/DR. HENRY J."	9703

Figure 7: Heroes with top weighted degree.

- Weight distribution.
  Check the distribution of similarity weights between pairs of heroes. Weight value translates to the number of common Comics Heroes have appeared in.

  ```
  MATCH ()-[k:KNOWS]->()
  RETURN (k.weight / 10) * 10 as weight,count(*) as count
  ORDER BY count DESC LIMIT 20
  ```

  159,244 out of total 171,644 relationships (94%) in our network has weight 9 or less. This means that most of our Heroes have only briefly met.The output is in figure 8



"weight"	"count"
0	159244
10	8415
20	1758
30	751
40	429
50	242
60	172
70	137
80	88
90	78
100	54

Figure 8: Weight distribution

5 Graph Analytics

- **Triangle count**:

  - Create sub-graph of all Heroes who know each other. (Graph Projection)

    ```
    CALL gds.graph.create(
    'myGraph',
    'Hero',
    {
    KNOWS: { orientation: 'UNDIRECTED'}
    }
    )
    ```

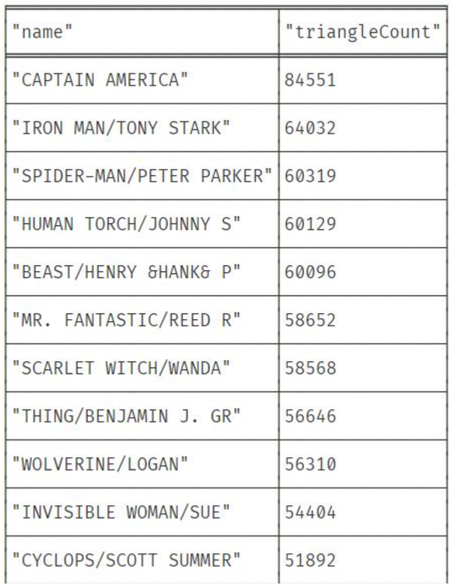
  - Triangle Count for each Hero node.

    ```
    CALL gds.triangleCount.stream('myGraph')
    YIELD nodeId, triangleCount
    RETURN gds.util.asNode(nodeId).name AS name, triangleCount
    ORDER BY triangleCount DESC
    ```

    The out put can be shown in figure 9

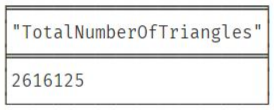
  - Triangle Counting

    ```
    CALL gds.triangleCount.stream('myGraph')
    YIELD nodeId, triangleCount
    RETURN sum(triangleCount)/3 as TotalNumberOfTriangles
    ```



"name"	"triangleCount"
"CAPTAIN AMERICA"	84551
"IRON MAN/TONY STARK"	64032
"SPIDER-MAN/PETER PARKER"	60319
"HUMAN TORCH/JOHNNY S"	60129
"BEAST/HENRY &HANK& P"	60096
"MR. FANTASTIC/REED R"	58652
"SCARLET WITCH/WANDA"	58568
"THING/BENJAMIN J. GR"	56646
"WOLVERINE/LOGAN"	56310
"INVISIBLE WOMAN/SUE"	54404
"CYCLOPS/SCOTT SUMMER"	51892

Figure 9: Triangles per Hero



"TotalNumberOfTriangles"
2616125

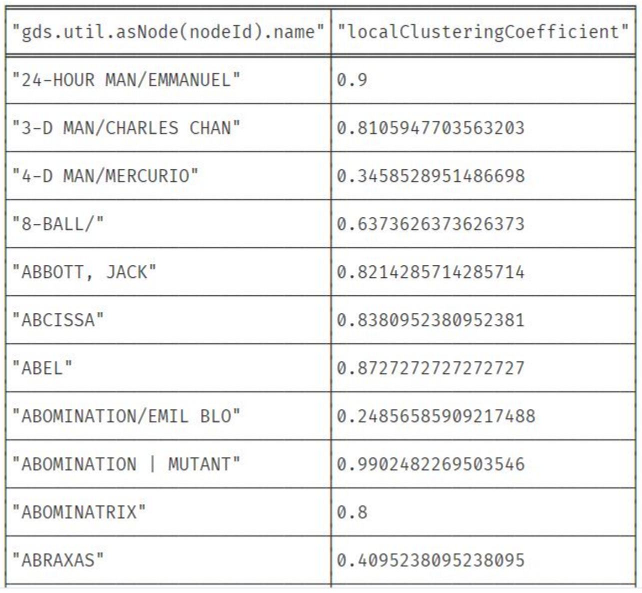
Figure 10: Total Number of triangles

- **Clustering coefficient**
  In graph theory, a clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together. Evidence suggests that in most real-world networks, and in particular social networks, nodes tend to create tightly knit groups characterized by a relatively high density of ties; this likelihood tends to be greater than the average probability of a tie randomly established between two nodes.

  The local clustering coefficients of a node is the likelihood that its neighbors are also connected.For example,the likelihood that one's friends are also friends.

  ```
  CALL gds.localClusteringCoefficient.stream('myGraph')
  YIELD nodeId, localClusteringCoefficient
  Return gds.util.asNode(nodeId).name, localClusteringCoefficient
  ```

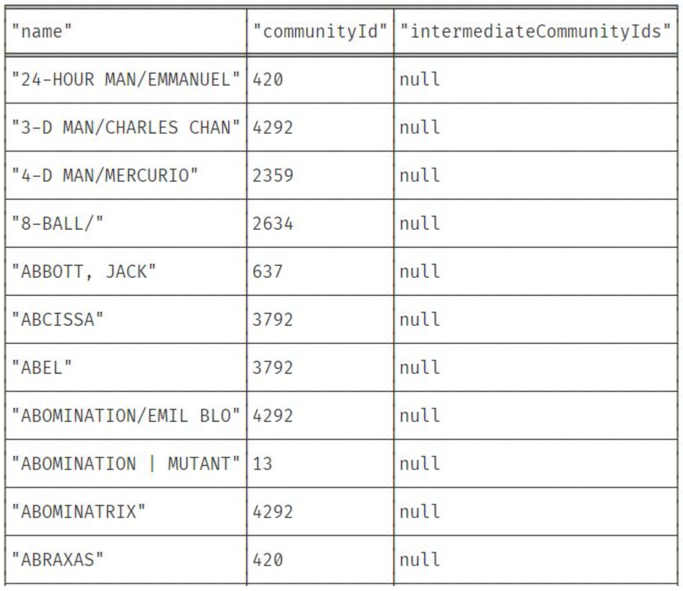
- **Community Detection Louvain method**
  Communities are groups of nodes within a network that are more densely connected to one another than to other nodes. Modularity is a metric that quantifies the quality of an assignment of nodes to communities by evaluating how much more densely connected the nodes within a community are compared to how connected they would be, on average, in a suitably defined random network. The Louvain method of community detection is an algorithm for detecting communities in networks that relies upon a heuristic for maximizing the modularity.



"gds.util.asNode(nodeId).name"	"localClusteringCoefficient"
"24-HOUR MAN/EMMANUEL"	0.9
"3-D MAN/CHARLES CHAN"	0.8105947703563203
"4-D MAN/MERCURIO"	0.3458528951486698
"8-BALL/"	0.6373626373626373
"ABBOTT, JACK"	0.8214285714285714
"ABCISSA"	0.8380952380952381
"ABEL"	0.8727272727272727
"ABOMINATION/EMIL BLO"	0.24856585909217488
"ABOMINATION | MUTANT"	0.9902482269503546
"ABOMINATRIX"	0.8
"ABRAXAS"	0.4095238095238095

Figure 11: Cluster Coefficient for myGraph

```
CALL gds.louvain.stream('myGraph')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).name AS name, communityId, intermediateCommunityIds
```



"name"	"communityId"	"intermediateCommunityIds"
"24-HOUR MAN/EMMANUEL"	420	null
"3-D MAN/CHARLES CHAN"	4292	null
"4-D MAN/MERCURIO"	2359	null
"8-BALL/"	2634	null
"ABBOTT, JACK"	637	null
"ABCISSA"	3792	null
"ABEL"	3792	null
"ABOMINATION/EMIL BLO"	4292	null
"ABOMINATION | MUTANT"	13	null
"ABOMINATRIX"	4292	null
"ABRAXAS"	420	null

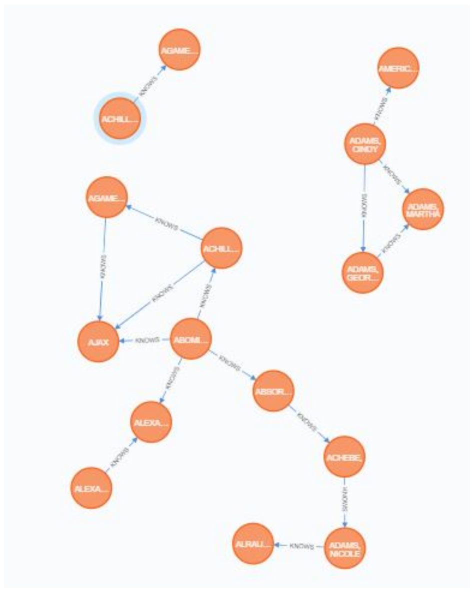
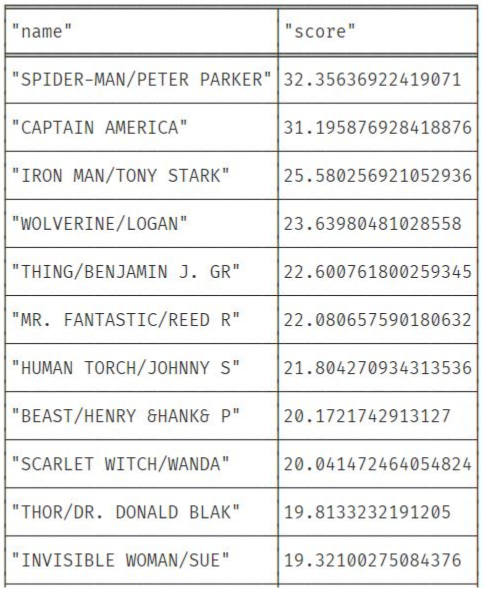


Figure 12: Community Detection sample from the output

- **Pagerank**
  PageRank is Google's popular search algorithm. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the node is.

```
CALL gds.pageRank.stream('myGraph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC
```



"name"	"score"
"SPIDER-MAN/PETER PARKER"	32.35636922419071
"CAPTAIN AMERICA"	31.195876928418876
"IRON MAN/TONY STARK"	25.580256921052936
"WOLVERINE/LOGAN"	23.63980481028558
"THING/BENJAMIN J. GR"	22.600761800259345
"MR. FANTASTIC/REED R"	22.080657590180632
"HUMAN TORCH/JOHNNY S"	21.804270934313536
"BEAST/HENRY &HANK& P"	20.1721742913127
"SCARLET WITCH/WANDA"	20.041472464054824
"THOR/DR. DONALD BLAK"	19.8133232191205
"INVISIBLE WOMAN/SUE"	19.32100275084376

Figure 13: Page Rank sample from the output

SPIDER-MAN/PETER PARKER and CAPTIN AMERICA has the highest pagerank scores. We can see the relationships with threshold of minimum weight is 100 among other heroes in figure 14

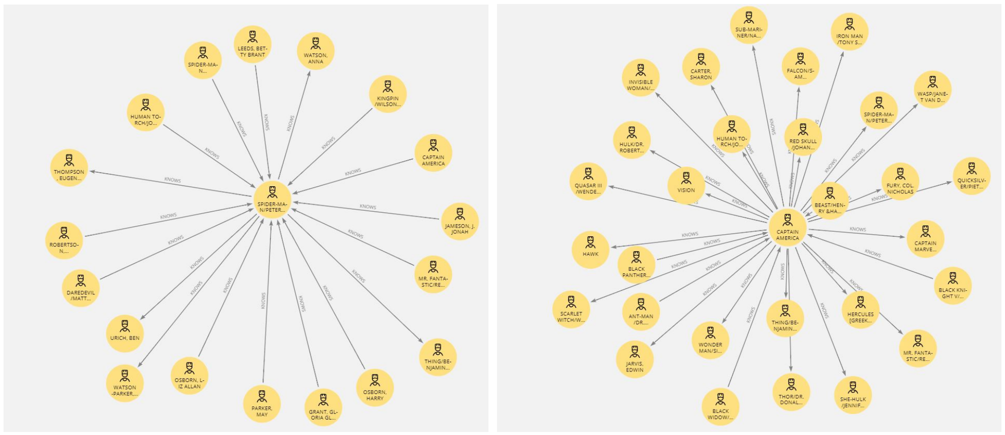
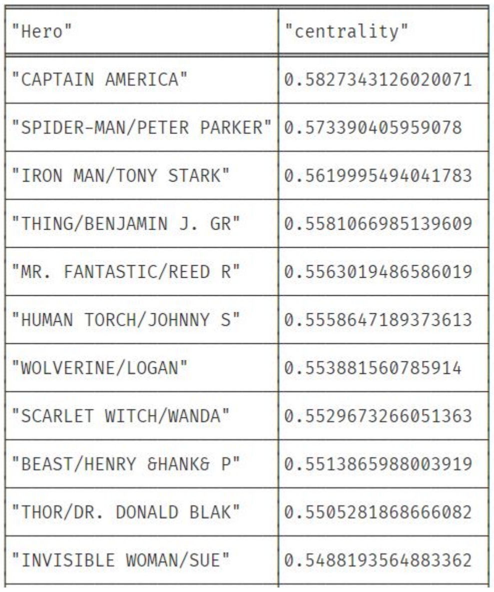


Figure 14: Relationships of SPIDER-MAN and CAPTIN AMERICA nodes

- **Closeness centrality-Wasserman and Faust**
  The Closeness Centrality of a node measures the distance from that node to all other nodes. Nodes with a high closeness score have the shortest distances to all other nodes. The premise of this algorithm is that nodes with short distance to other nodes can spread information very efficiently through the network.

  The greater the raw closeness score, the greater the time it takes on average for information originating at random points in the network to arrive at the node. Equally, one can interpret closeness as the potential ability of a node to reach all other nodes as quickly as possible.

```
CALL gds.alpha.closeness.stream({
nodeProjection: 'Hero',
relationshipProjection: 'KNOWS', improved:true
})
YIELD nodeId, centrality
RETURN gds.util.asNode(nodeId).name AS Hero, centrality
ORDER BY centrality DESC
```



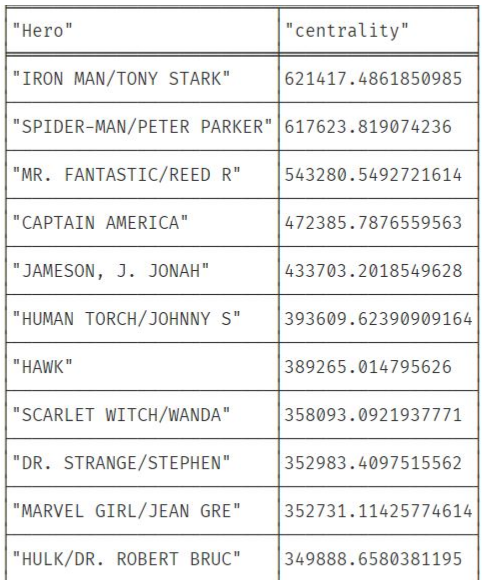
"Hero"	"centrality"
"CAPTAIN AMERICA"	0.5827343126020071
"SPIDER-MAN/PETER PARKER"	0.573390405959078
"IRON MAN/TONY STARK"	0.5619995494041783
"THING/BENJAMIN J. GR"	0.5581066985139609
"MR. FANTASTIC/REED R"	0.5563019486586019
"HUMAN TORCH/JOHNNY S"	0.5558647189373613
"WOLVERINE/LOGAN"	0.553881560785914
"SCARLET WITCH/WANDA"	0.5529673266051363
"BEAST/HENRY &HANK& P"	0.5513865988003919
"THOR/DR. DONALD BLAK"	0.5505281868666082
"INVISIBLE WOMAN/SUE"	0.5488193564883362

Figure 15: Closeness Centrality sample from the output

Captain America is in such a privileged position, that he will be leading in all categories of centralities. We can observe that nodes in more tight communities have higher closeness centrality indexes while those on the brinks and less connected have smaller values.

- **Betweenness Centrality**
  Betweenness centrality quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.

```
CALL gds.alpha.betweenness.stream({
nodeProjection: 'Hero',
relationshipProjection: 'KNOWS'
})
YIELD nodeId, centrality
RETURN gds.util.asNode(nodeId).name AS Hero, centrality
ORDER BY centrality DESC
```



"Hero"	"centrality"
"IRON MAN/TONY STARK"	621417.4861850985
"SPIDER-MAN/PETER PARKER"	617623.819074236
"MR. FANTASTIC/REED R"	543280.5492721614
"CAPTAIN AMERICA"	472385.7876559563
"JAMESON, J. JONAH"	433703.2018549628
"HUMAN TORCH/JOHNNY S"	393609.62390909164
"HAWK"	389265.014795626
"SCARLET WITCH/WANDA"	358093.0921937771
"DR. STRANGE/STEPHEN"	352983.4097515562
"MARVEL GIRL/JEAN GRE"	352731.11425774614
"HULK/DR. ROBERT BRUC"	349888.6580381195

Figure 16: Betweeness Centrality sample from the output

6 References

1. arXiv:cond-mat/0202174v1 [cond-mat.dis-nn]

2. R. Chappell. Marvel chronology project, September 2001. ¡http://www.chronologyproject.com¿.

3. Marvel Social graph in Neo4j.

4. Sandbox Bloom.