Identifying Social Media Influencers using Graph Analytics

Introduction

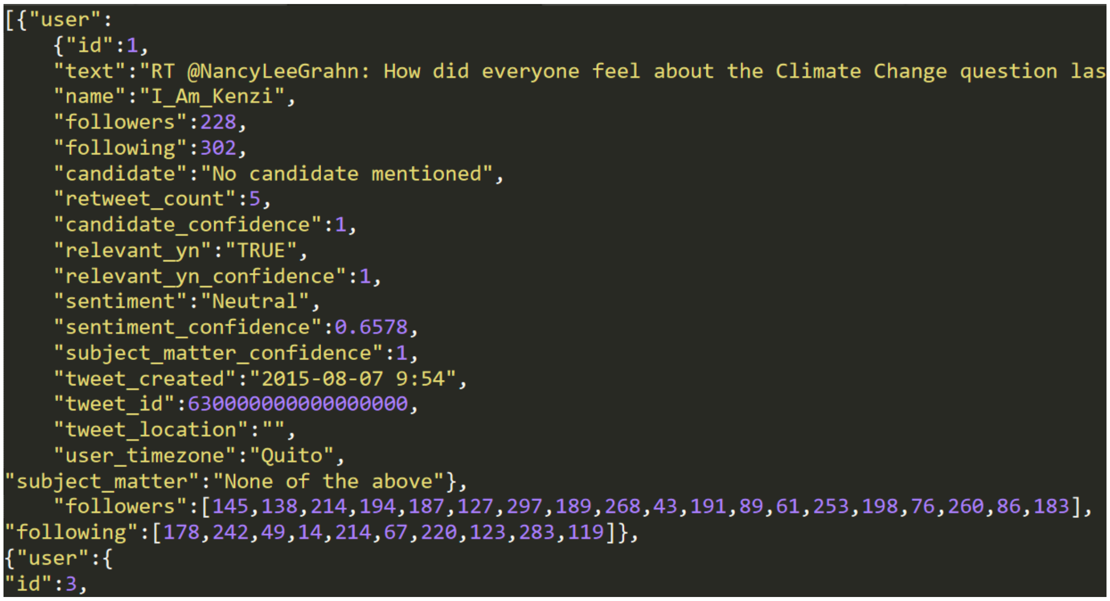
The Project implemented mainly aims to find the top influencers in the 2016 GOP debate held in the states. The major tools used for implementation of the project include- nodejs for data preprocessing, and Neo4j for graph analytics. Several graph-based algorithms are implemented to find the influencers in the social media network. This dataset is a twitter downloaded from Kaggle. Major algorithms to identify the influencers include- Centrality Algorithm: Betweenness, Degree, Page Rank and Closeness Algorithm. Louvain Modularity, Label Propagation Algorithm and Triangles Clustering Coefficient Algorithm is used to detect the communities in the network.

As most of the Neo4J is licenced, I could not find any suitable way to share the Neo4j saved cypher query scripts. Though, I have attached all the queries that I have executed in the folder Code Directory in the file “query.cql”. The code used for data preprocessing is written in nodejs and it is mentioned in the file named Data_Preprocessing.

Implementation

1) **Data Preprocessing**

The imported original dataset was modified by adding two attributes in the array format, “followers” and “following”. These arrays must have the ID’s of the followers and following such that one can form a graph based on the data provided. For instance, the dataset below provides one record in the processed dataset. The dataset used is in the JSON format.



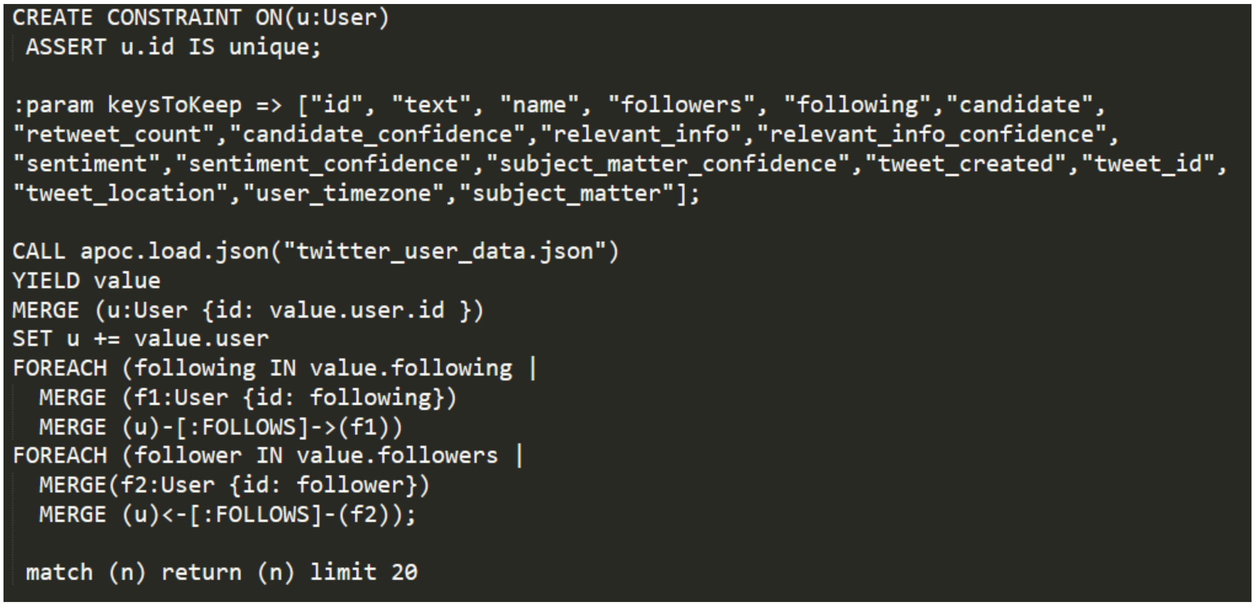
```
[{"user":
    {"id":1,
    "text":"RT @NancyLeeGrahn: How did everyone feel about the Climate Change question las
    "name":"I_Am_Kenzi",
    "followers":228,
    "following":302,
    "candidate":"No candidate mentioned",
    "retweet_count":5,
    "candidate_confidence":1,
    "relevant_yn":"TRUE",
    "relevant_yn_confidence":1,
    "sentiment":"Neutral",
    "sentiment_confidence":0.6578,
    "subject_matter_confidence":1,
    "tweet_created":"2015-08-07 9:54",
    "tweet_id":630000000000000000,
    "tweet_location":"",
    "user_timezone":"Quito",
"subject_matter":"None of the above"},
    "followers":[145,138,214,194,187,127,297,189,268,43,191,89,61,253,198,76,260,86,183],
"following":[178,242,49,14,214,67,220,123,283,119]},
{"user":{
"id":3,
```

2) Graph-Analytics

The query language in Neo4j is known as Cypher Query Language. The Implementation steps are mentioned in the following section. The implementation is executed in the Neo4j Browser.

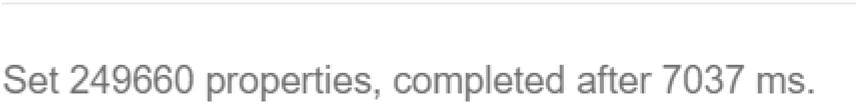
The code below depicts the formation of the graph code,

Firstly, a CONSTRAINT is developed on User, after which the parameters are set in the form of graph with the nodes. The function CALL apoc.load.json () is used to load the dataset in the Neo4j browser. This would create “FOLLOW” relationship between the nodes.



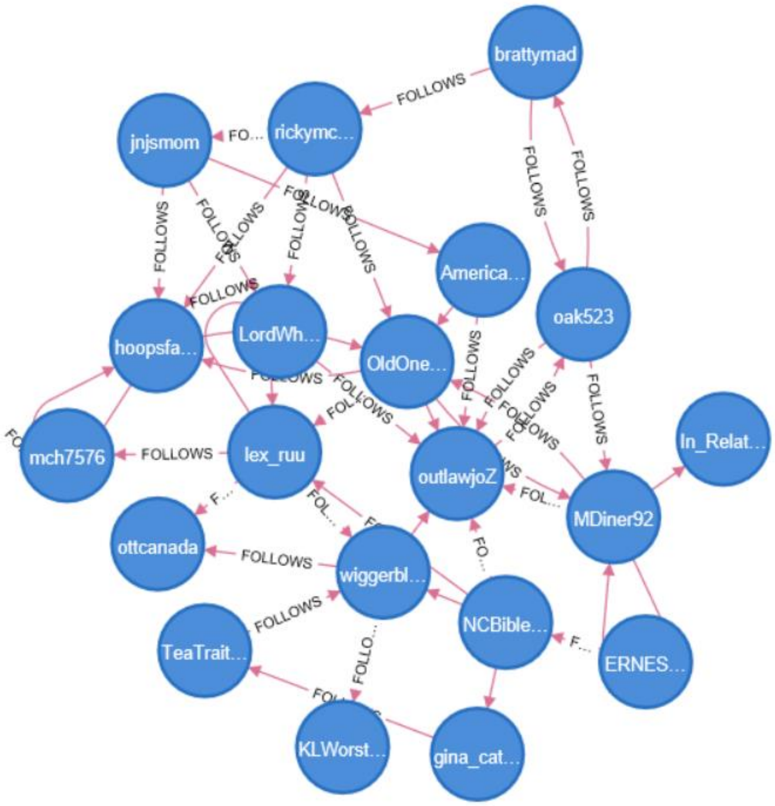
```
CREATE CONSTRAINT ON(u:User)
 ASSERT u.id IS unique;

:param keysToKeep => ["id", "text", "name", "followers", "following","candidate",
"retweet_count","candidate_confidence","relevant_info","relevant_info_confidence",
"sentiment","sentiment_confidence","subject_matter_confidence","tweet_created","tweet_id",
"tweet_location","user_timezone","subject_matter"];

CALL apoc.load.json("twitter_user_data.json")
YIELD value
MERGE (u:User {id: value.user.id })
SET u += value.user
FOREACH (following IN value.following |
  MERGE (f1:User {id: following})
  MERGE (u)-[:FOLLOWS]->(f1))
FOREACH (follower IN value.followers |
  MERGE(f2:User {id: follower})
  MERGE (u)<-[:FOLLOWS]-(f2));

 match (n) return (n) limit 20
```

The following output is received after executing the function CALL apoc.load.json ().



```
Set 249660 properties, completed after 7037 ms.
```

match (n) return (n) would load the output graph, with the connected nodes. The relationship has been created for 13,876 nodes but due to space complexity only 300 nodes can be displayed at a time. Also LIMIT 20 would give me the graph with only 20 nodes. The output is mentioned in the figure below.



Once the graph is made, the algorithms are executed in Graph Algorithms Playground

In the Graph Algorithms Playground Homepage, there are 4 options –

Centrality - Degree, Eigen Vector, Page Rank, articlerank, betweenness, approx betweenness, closeness, harmonic;

Community Detection - Louvain, LAbel Propogation, Connected Components, strongly connected components, triangles, triangleCount;

Path Finding - Shortest Path, A*, Single source shortest path, all pairs shortest path; and

Similarity - jaccard, overlap, cosine, pearson, Euclidean.

The following factors must be defined before executing the algorithms:

Label: This is usually for defining the nodes and for the proposed model it has been always set to user.

Relationship Type: This defines the relationship type for classifying the nodes. For the proposed model it has been set to “FOLLOWS”

Direction: As the graph we have constructed is a bidirectional type, thus there are three options to be considered while executing the algorithms- IN, OUT and BOTH. For the proposed model it has been set to Both. As we are considering the followers as well as following in the analysis.

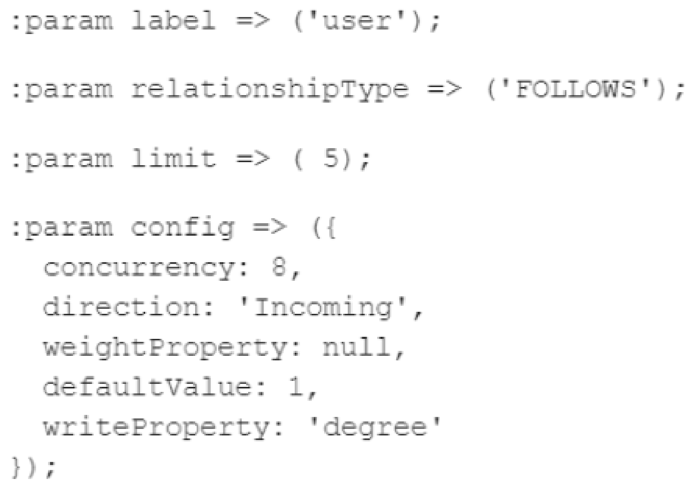
rows to show: This would show how many output rows do you want. It is set to top 5 influencers in the proposed model.

The outputs and implementations of the algorithm is shown below:

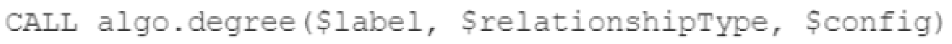
Influencers Identification Algorithms

1) **Degree centrality** measures the direct connections a node has.

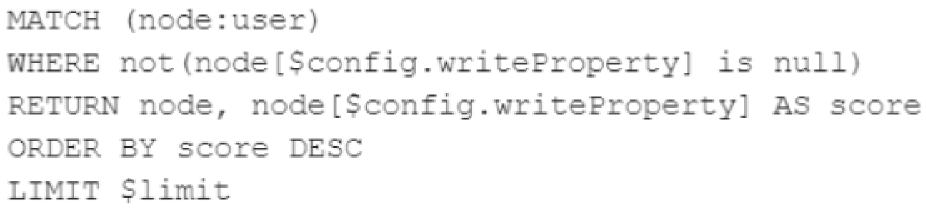
Code for execution of degree centrality



```
:param label => ('user');

:param relationshipType => ('FOLLOWS');

:param limit => ( 5);

:param config => ({
  concurrency: 8,
  direction: 'Incoming',
  weightProperty: null,
  defaultValue: 1,
  writeProperty: 'degree'
});
```

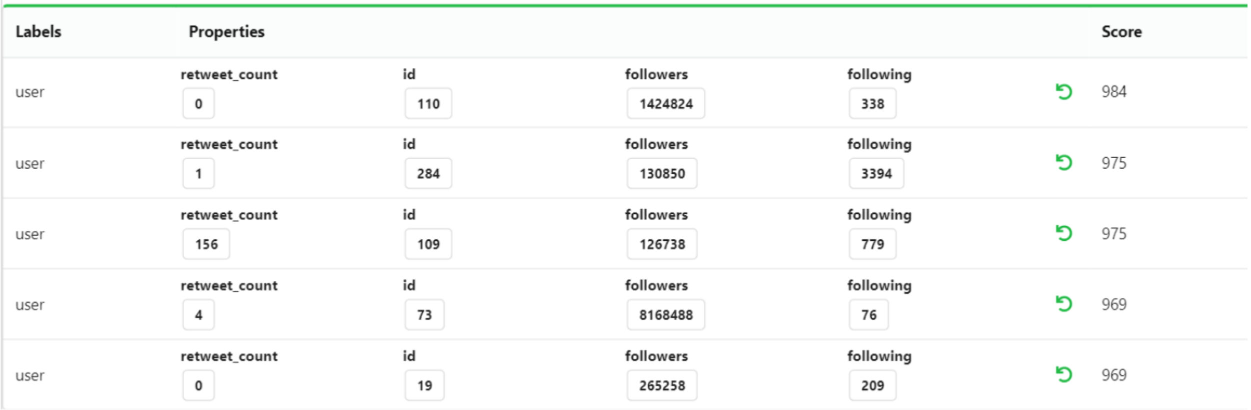


```
CALL algo.degree($label, $relationshipType, $config)
```



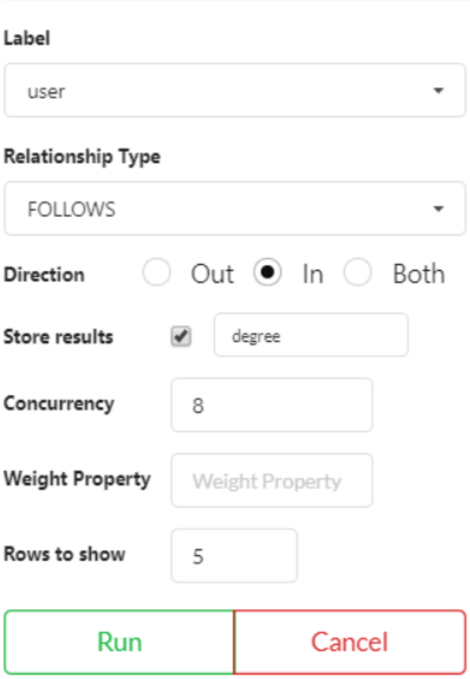
```
MATCH (node:user)
WHERE not(node[$config.writeProperty] is null)
RETURN node, node[$config.writeProperty] AS score
ORDER BY score DESC
LIMIT $limit
```

Output of degree centrality



Labels	Properties				Score
	retweet_count	id	followers	following	
user	0	110	1424824	338	984
user	1	284	130850	3394	975
user	156	109	126738	779	975
user	4	73	8168488	76	969
user	0	19	265258	209	969

This output can be obtained from the UI from Neo4j Browser as well.



Label: user

Relationship Type: FOLLOWS

Direction ○ Out ● In ○ Both

Store results ☑ degree

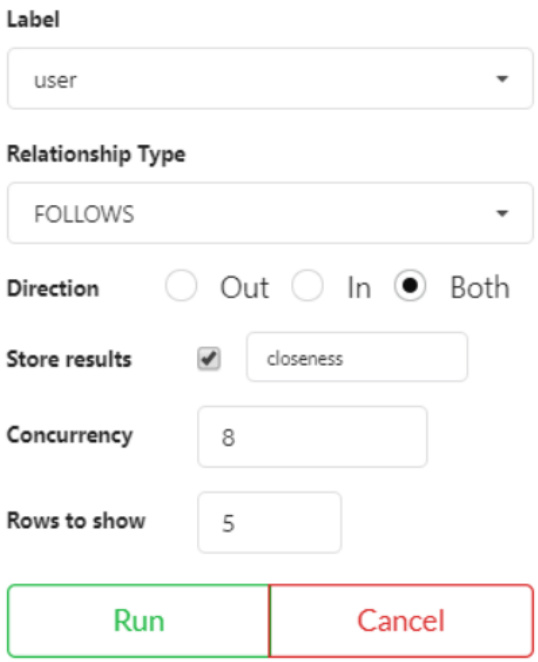
Concurrency: 8

Weight Property: Weight Property

Rows to show: 5

Closeness centrality detect nodes that can spread information efficiently through a graph.

UI characteristics



Label

user

Relationship Type

FOLLOWS

Direction ○ Out ○ In ● Both

Store results ☑ closeness

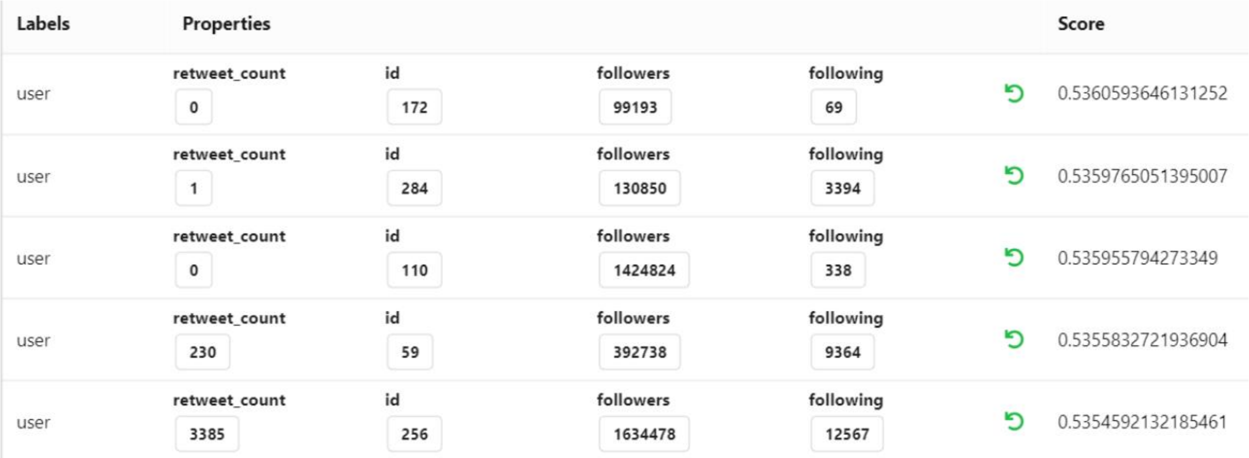
Concurrency 8

Rows to show 5

Code

```
:param label => ('user');

:param relationshipType => ('FOLLOWS');

:param limit => ( 5);

:param config => ({
  concurrency: 8,
  direction: 'Both',
  writeProperty: 'closeness'
});


CALL algo.closeness($label, $relationshipType, $config)


MATCH (node:user)
WHERE not(node[$config.writeProperty] is null)
RETURN node, node[$config.writeProperty] AS score
ORDER BY score DESC
LIMIT $limit
```

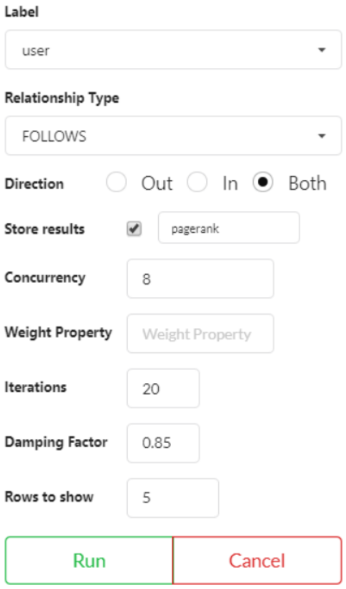
Output



Labels	Properties					Score
user	retweet_count 0	id 172	followers 99193	following 69	↺	0.5360593646131252
user	retweet_count 1	id 284	followers 130850	following 3394	↺	0.5359765051395007
user	retweet_count 0	id 110	followers 1424824	following 338	↺	0.535955794273349
user	retweet_count 230	id 59	followers 392738	following 9364	↺	0.5355832721936904
user	retweet_count 3385	id 256	followers 1634478	following 12567	↺	0.5354592132185461

Page Rank algorithm: This measures the connectivity of the nodes.

UI interface



Label: user

Relationship Type: FOLLOWS

Direction ○ Out ○ In ● Both

Store results ☑ pagerank

Concurrency 8

Weight Property: Weight Property

Iterations 20

Damping Factor 0.85

Rows to show 5

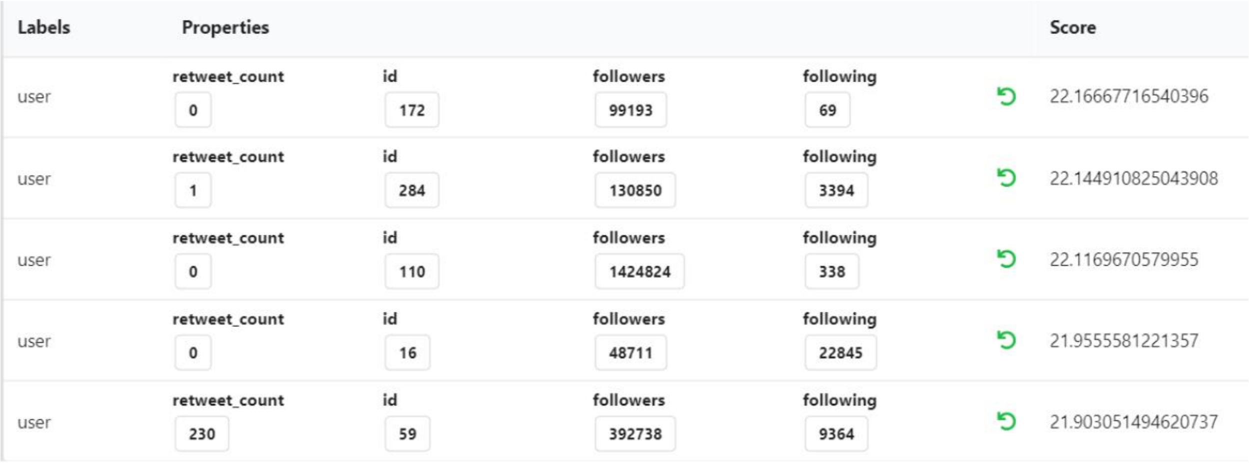
Code

```
:param label => ('user');

:param relationshipType => ('FOLLOWS');

:param limit => ( 5);

:param config => ({
  concurrency: 8,
  direction: 'Both',
  weightProperty: null,
  defaultValue: 1,
  dampingFactor: 0.85,
  iterations: 20,
  writeProperty: 'pagerank'
});
```

```
CALL algo.pageRank($label, $relationshipType, $config)
```

```
MATCH (node:user)
WHERE not(node[$config.writeProperty] is null)
RETURN node, node[$config.writeProperty] AS score
ORDER BY score DESC
LIMIT $limit
```

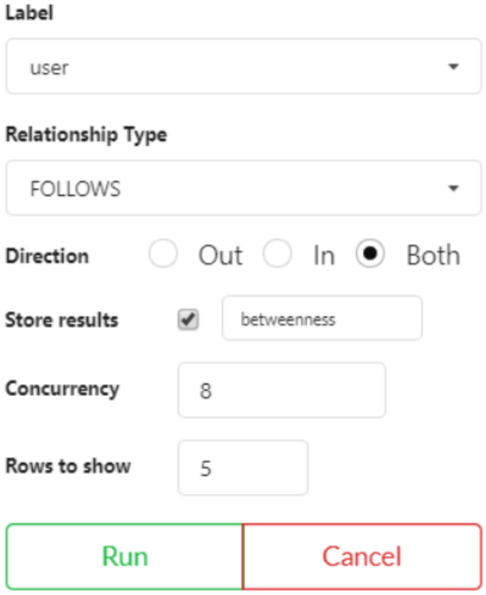
Output



Labels	Properties					Score
	retweet_count	id	followers	following		
user	0	172	99193	69	↺	22.16667716540396
user	1	284	130850	3394	↺	22.144910825043908
user	0	110	1424824	338	↺	22.1169670579955
user	0	16	48711	22845	↺	21.9555581221357
user	230	59	392738	9364	↺	21.903051494620737

Betweenness: the way of detecting the influence a node has over the flow of information in the graph

UI inputs



Label: user

Relationship Type: FOLLOWS

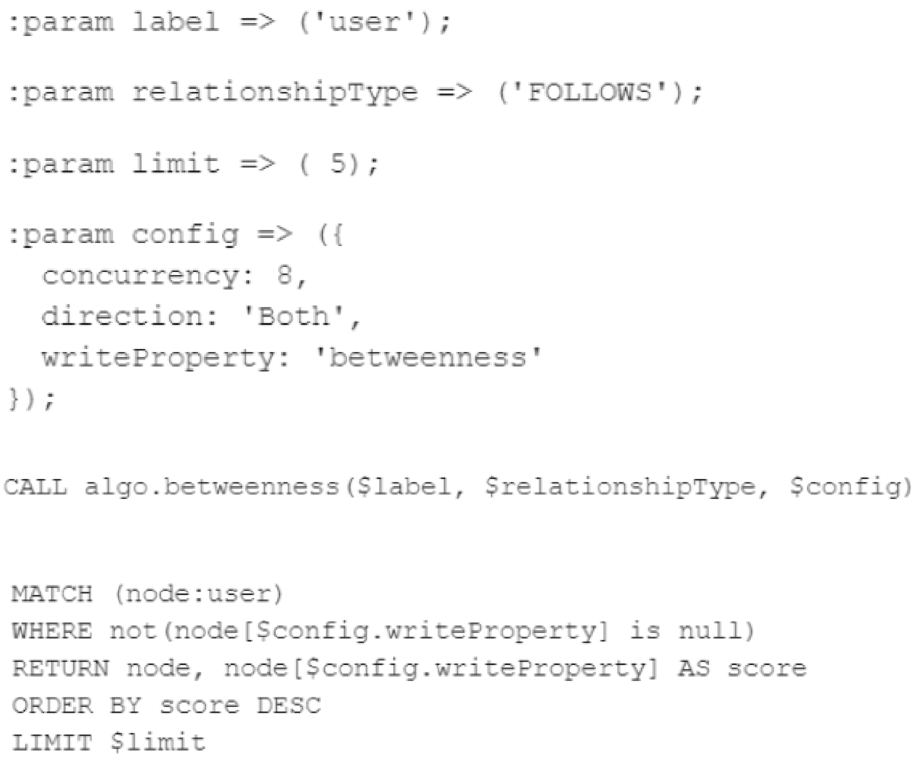
Direction ○ Out ○ In ● Both

Store results ☑ betweenness

Concurrency: 8

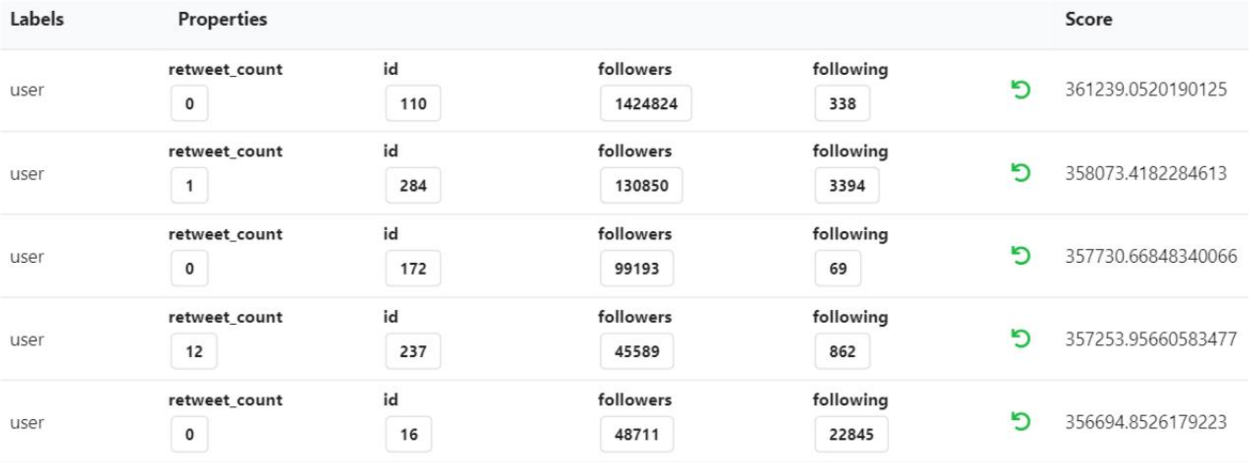
Rows to show: 5

Code



```
:param label => ('user');

:param relationshipType => ('FOLLOWS');

:param limit => ( 5);

:param config => ({
  concurrency: 8,
  direction: 'Both',
  writeProperty: 'betweenness'
});

CALL algo.betweenness($label, $relationshipType, $config)


 MATCH (node:user)
 WHERE not(node[$config.writeProperty] is null)
 RETURN node, node[$config.writeProperty] AS score
 ORDER BY score DESC
 LIMIT $limit
```

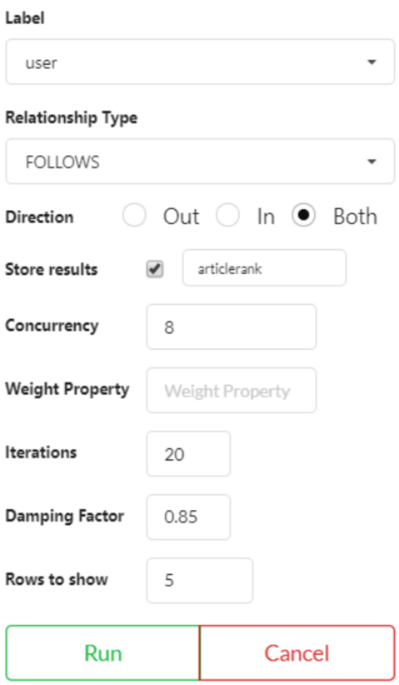
Output



Labels	Properties					Score
	retweet_count	id	followers	following		
user	0	110	1424824	338	↺	361239.0520190125
user	1	284	130850	3394	↺	358073.4182284613
user	0	172	99193	69	↺	357730.66848340066
user	12	237	45589	862	↺	357253.95660583477
user	0	16	48711	22845	↺	356694.8526179223

Article rank: A variant of page rank

UI Inputs



Label: user ▾

Relationship Type: FOLLOWS ▾

Direction ○ Out ○ In ● Both

Store results ☑ articlerank

Concurrency: 8

Weight Property: Weight Property

Iterations: 20

Damping Factor: 0.85

Rows to show: 5

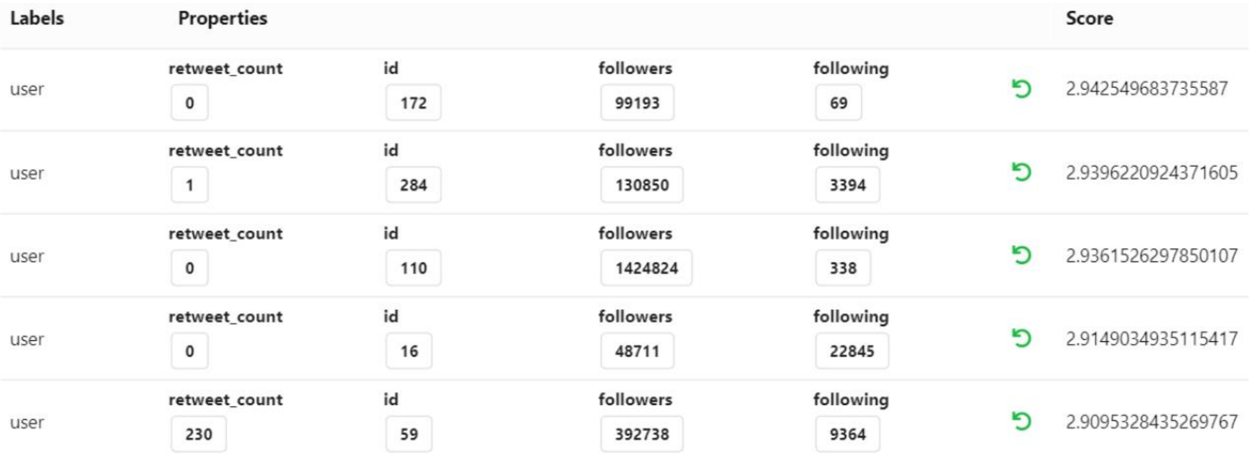
Code

```
:param label => ('user');

:param relationshipType => ('FOLLOWS');

:param limit => ( 5);

:param config => ({
  concurrency: 8,
  direction: 'Both',
  weightProperty: null,
  defaultValue: 1,
  dampingFactor: 0.85,
  iterations: 20,
  writeProperty: 'articlerank'
});
```

```
CALL algo.articleRank($label, $relationshipType, $config)
```

```
MATCH (node:user)
WHERE not(node[$config.writeProperty] is null)
RETURN node, node[$config.writeProperty] AS score
ORDER BY score DESC
LIMIT $limit
```

Output

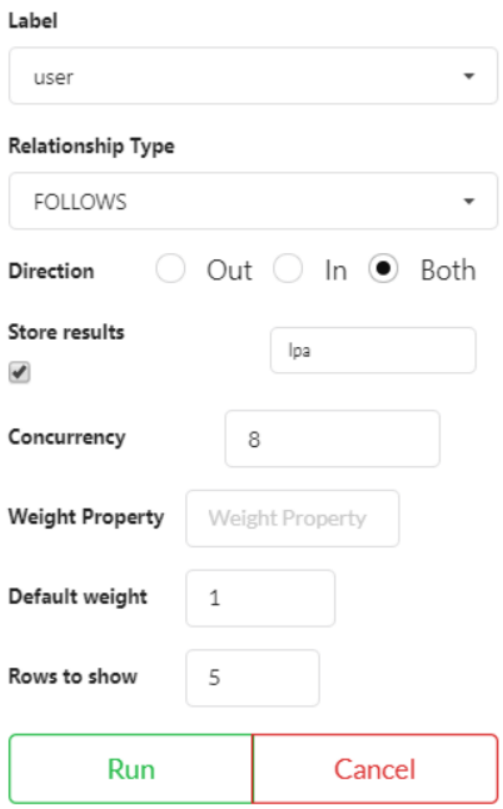


Labels	Properties					Score
user	retweet_count 0	id 172	followers 99193	following 69	⟲	2.942549683735587
user	retweet_count 1	id 284	followers 130850	following 3394	⟲	2.9396220924371605
user	retweet_count 0	id 110	followers 1424824	following 338	⟲	2.9361526297850107
user	retweet_count 0	id 16	followers 48711	following 22845	⟲	2.9149034935115417
user	retweet_count 230	id 59	followers 392738	following 9364	⟲	2.9095328435269767

Community Detection Algorithm

Label Propagation: Fast finding algorithms to find communities in a graph

UI input



Label: user ▾

Relationship Type: FOLLOWS ▾

Direction ○ Out ○ In ● Both

Store results ☑ lpa

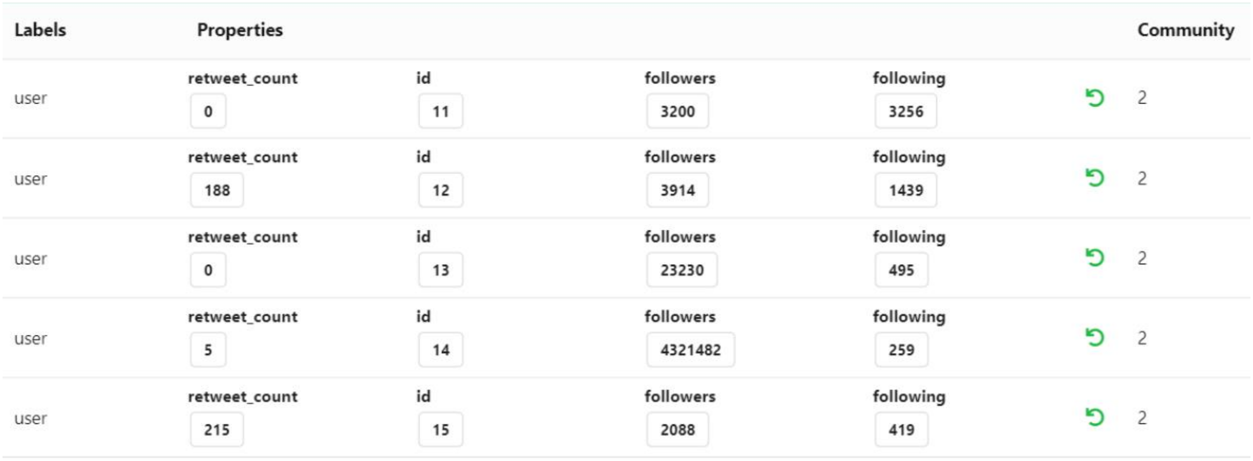
Concurrency 8

Weight Property Weight Property

Default weight 1

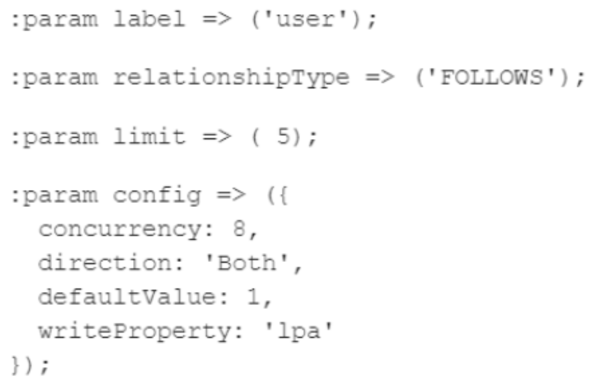
Rows to show 5

Output

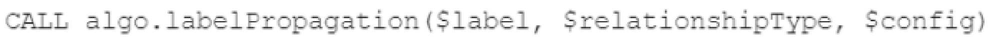


Labels	Properties					Community
user	retweet_count 0	id 11	followers 3200	following 3256	↺	2
user	retweet_count 188	id 12	followers 3914	following 1439	↺	2
user	retweet_count 0	id 13	followers 23230	following 495	↺	2
user	retweet_count 5	id 14	followers 4321482	following 259	↺	2
user	retweet_count 215	id 15	followers 2088	following 419	↺	2

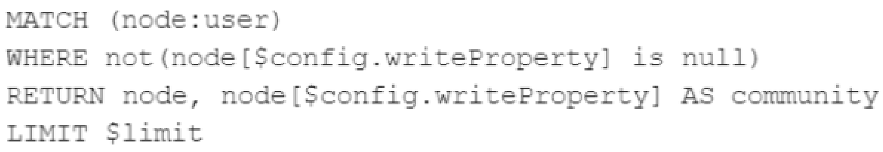
Code



```
:param label => ('user');

:param relationshipType => ('FOLLOWS');

:param limit => ( 5);

:param config => ({
  concurrency: 8,
  direction: 'Both',
  defaultValue: 1,
  writeProperty: 'lpa'
});
```



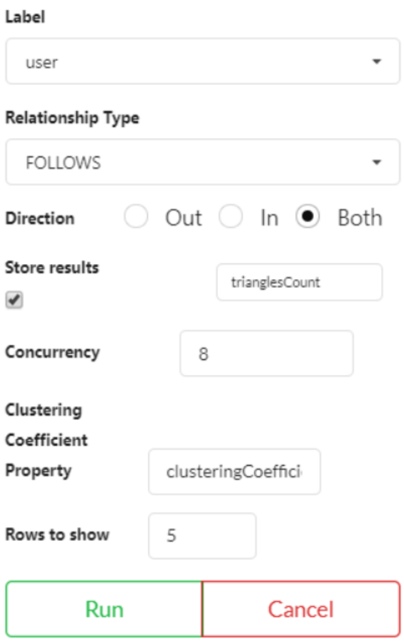
```
CALL algo.labelPropagation($label, $relationshipType, $config)
```



```
MATCH (node:user)
WHERE not(node[$config.writeProperty] is null)
RETURN node, node[$config.writeProperty] AS community
LIMIT $limit
```

Triangles Count Coefficient: this would find set of 3 nodes where each node has a relationship to all other nodes.

UI Input



Label

user

Relationship Type

FOLLOWS

Direction ○ Out ○ In ● Both

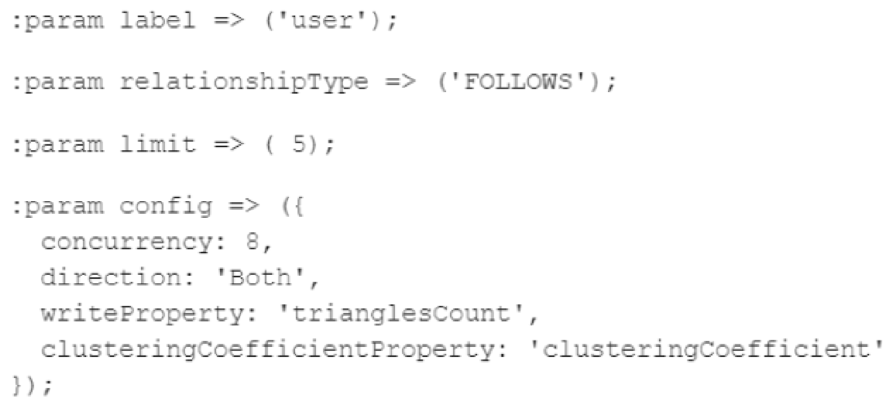
Store results ☑ trianglesCount

Concurrency 8

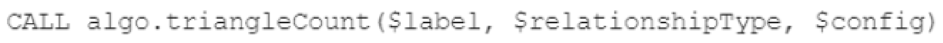
Clustering Coefficient Property clusteringCoeffici

Rows to show 5

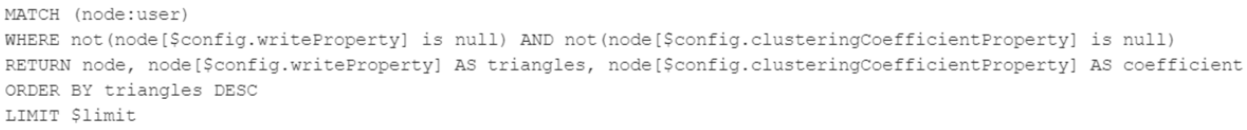
Code for execution of Triangles Coefficient Algorithm



```
:param label => ('user');

:param relationshipType => ('FOLLOWS');

:param limit => ( 5);

:param config => ({
  concurrency: 8,
  direction: 'Both',
  writeProperty: 'trianglesCount',
  clusteringCoefficientProperty: 'clusteringCoefficient'
});
```

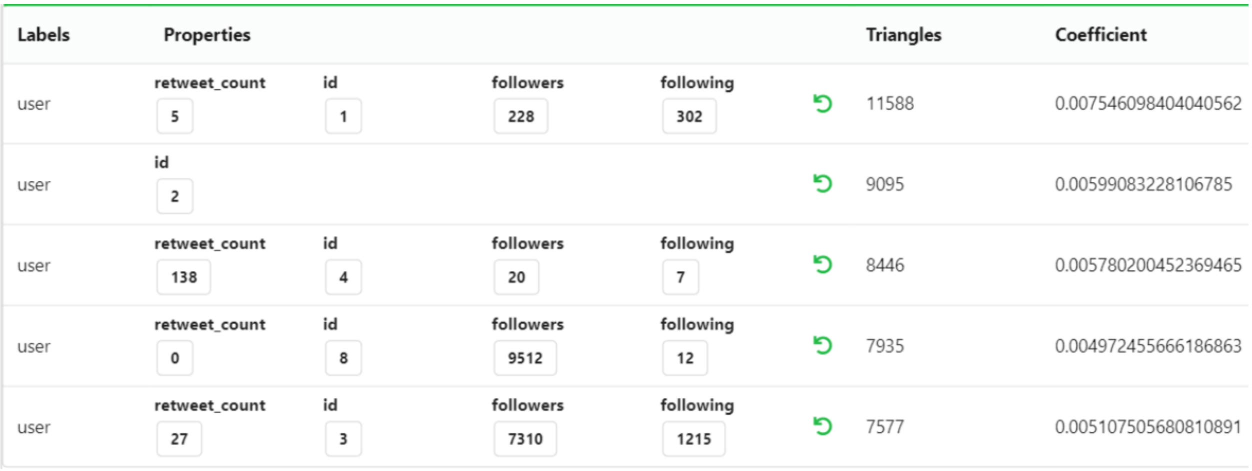


```
CALL algo.triangleCount($label, $relationshipType, $config)
```



```
MATCH (node:user)
WHERE not(node[$config.writeProperty] is null) AND not(node[$config.clusteringCoefficientProperty] is null)
RETURN node, node[$config.writeProperty] AS triangles, node[$config.clusteringCoefficientProperty] AS coefficient
ORDER BY triangles DESC
LIMIT $limit
```

output



Labels	Properties					Triangles	Coefficient
user	retweet_count 5	id 1	followers 228	following 302	↺	11588	0.007546098404040562
user	id 2				↺	9095	0.00599083228106785
user	retweet_count 138	id 4	followers 20	following 7	↺	8446	0.005780200452369465
user	retweet_count 0	id 8	followers 9512	following 12	↺	7935	0.004972455666186863
user	retweet_count 27	id 3	followers 7310	following 1215	↺	7577	0.005107505680810891