UNIVERSITY OF PIRAEUS

DEPARTMENT OF DIGITAL SYSTEMS

Postgraduate Studies Programme "Information System & Services"

Area «Big Data and Analytics»

"Evaluation of Centrality Algorithms for Information Spread in Social Networks"

Natalie Oskian – ME1610

Supervisor Professor

C. Doulkeridis

ACADEMIC YEAR 2019-2020

Abstract

The increased interest in social networks and how the information spread within them, has highlighted the need to identify what makes the nodes that contribute more to the information spread important. Centrality indices are measures of a node's importance in a given network. Many of those have been proposed over the years, although there is not a common approach on how to use them in the case of information spread maximization. In this study, we evaluate five existing centrality algorithms, based on their performances as centrality measures to select nodes that will spread information across the graph. The centrality algorithms that are evaluated are Degree, Closeness, Betweenness, Eigenvector and PageRank. The results indicate the importance of nodes' in-degree and of relationships' direction. Although none of the algorithms outperforms the others in all cases, Degree Centrality has consistently good performance. The algorithm that achieves the lowest spreads of information is Betweenness.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 General information

In this study, we have researched the spread of information on social networks. More specifically, we focus on the methods of selecting which nodes will spread information more efficiently. By "efficiently", we mean that the information will spread as much as possible, starting with a minimum number of nodes, and with the minimum possible total distance to cover. The latter means that it is desired to have a big spread quickly and then to try reaching as many distant nodes as possible.

In social networks, people get connected and perform actions. By following, or liking, a person or page that you are interested in, you can watch their activity. Several topics, ideas, news, pieces of art, events, and many other information can be spread widely across the network. The promotion of such information can be easily and cheaply achieved. The pros versus a traditional advertising method are that people are reached with this information via people that they find interesting and most probably influence them.

Which nodes of a social network are those that can spread information more efficiently, though? In graph theory, centrality is a measure that identifies the most important nodes within a graph. What makes a node important, though? Several centrality algorithms have been implemented over the last decades, and each of them sees the importance of nodes from another perspective.

A node can usually be considered as important if it has many connections, if it is connected to popular nodes that in turn are connected to other popular nodes, or if it serves as a bridge between clusters.

Many researchers have tried to identify the most important nodes of graphs by using existing or improved versions of centrality algorithms. However, the results are still inconclusive as to the way that one can choose the most suitable centrality for a network with a specific structure.

In this study, we attempt to evaluate common centrality measures for their performances on information spread. We try to interpret the results and identify the areas that need improvement based on the networks' structure.

1.2 Problem Statement

The problem that we try to solve in this study is that there is not one common approach to selecting nodes for efficient information spread within social networks. There is a need to evaluate existing centrality algorithms in several social graphs to understand their strong points and weaknesses in each case. We need to examine if characteristics of the graphs' structures can define which centrality algorithm is the most suitable to achieve the best results.

We will achieve this by studying three different datasets of Facebook pages and running five centrality algorithms to score their nodes. Then, we will examine the results of the information spreads achieved when using the top scored nodes with each centrality measure.

1.3 Dissertation Structure

In Chapter 2 "*Background*", we have written a Literature Review about published researches that are related to this dissertation. A description of the technologies used for the purpose of this study follows next. The technologies that are described concern the graph database management system, the query language, the libraries that were used and the graph algorithms, in general.

In Chapter 3 "*Solution Architecture*", you can find the description of the solution architecture, described with a block diagram and a high-level description of how the solution is designed. All modules are, then, described in more detail and those concern the environment that hosts the graph database, the graph database itself, the user interface for managing the graphs, the libraries and what was used in each one of them and graph visualization tool.

In Chapter 4 "*Graph Algorithms*", the Centrality algorithms that are chosen for evaluation are described in detail. For each one of them there is information about their history, use, the use cases for which they suit most, and explanation of how they work.

In Chapter 5 "*Experimental Study*", we have described the whole experiment step by step. There is an explanation of the datasets, the Cypher commands used to run the process, the graph data models, the graph stats, and tables and charts with the top scored nodes and the nodes where the information spreads. Finally, the results are described and discussed.

In Chapter 6 "*Conclusions and Future Work*", the conclusions drawn by the results of the experiment are described in detail. The problems and difficulties that were met, as well as the

results that were unexpected, are also referred. In the end, there are some ideas for further investigation and possible improvement of the results.

2 Background

2.1 Literature Review

Many research studies have been conducted in the past years regarding node importance and information spread in social networks. Many researchers have tried to define what importance means in specific contexts. Others focused on evaluating existing centrality algorithms using several different methodologies, and some tried to improve existing algorithms to better suit specific needs.

In "Centralities: Capturing the Fuzzy Notion of Importance in Social Graphs", the authors reviewed, compared, and highlighted several centrality measures in contemporary social networks. The method they followed to assess the importance of nodes was to sequentially remove nodes, starting from the highest-ranking node and the others following by descending rank order and observe the remaining graph as to the size of the biggest connected component and the average route length. The results were what they expected, with betweenness centralities succeeding to disrupt the graph's structure efficiently and eccentricity, degree, Eigenvector, and closeness not significantly affecting the graph's connectivity. (Merrer & Tredan, 2009)

The "Spread of (mis)information in social networks" analyzed the spread of misinformation in large societies. According to the paper, the critical role in misinformation spread is held by "forceful agents", defined as those who influence the beliefs of other individuals, without them ever changing their opinions. The analysis exploited the fact that belief evolution is a stochastic process (Markov chain). The results showed that social network matrices with large second eigenvalues placed tight bounds on the spreading of misinformation. Another finding was that the location of these forceful agents within the network played an important role. Each social network would result in very different limitations on spreading, based on the agents' locations. (Acemoglu, Ozdaglar, & ParandehGheibi, 2010)

In their paper "Why Rumors Spread Fast in Social Networks", the authors analyze how news spread in social networks by simulating an information spread process in others than the existing social network topologies. They proved that information spreads in sub-logarithmic time in existing social networks. At the same time, they need at least logarithmic time for the same

information spreading process in the other social network topologies. As existing real-world social networks, they consider topologies that demonstrate preferential attachment. Their findings show the importance of nodes with few neighbors in the quick dissemination. High-degree nodes broadcast news to a broad audience, while the low-degree nodes transfer the news quickly from one neighbor to another. (Doerr, Fouz, & Friedrich, 2012)

In "Social Contagion: An Empirical Study of Information Spread on Digg and Twitter Follower Graphs", the authors studied the dynamics of information spread on social networks and how the network's structure affects it analyzing data from Digg and Twitter. Their findings showed that in Digg networks that were dense and highly connected, the information appeared to spread through an interconnected community, while on Twitter, the spreads were more tree-like. They also found that when top nodes in Twitter became part of an information spreading, they broadened the spread and increased its size without affecting its depth. However, in Digg, when top users submitted a story, they always resulted in a big spread of a specific size. In contrast, for a story submitted by a poorly connected user to grow, there had to be involvement from a top user, and that was unusual. (Lerman, Ghosh, & Surachawala, 2012)

In "Six degrees of information: Using social network analysis to explore the spread of information within sport social networks", the study was about exploring how sports events organizers spread information through Twitter. The results showed how the organizers leverage Twitter and top users to help promote the events. Using sociograms and quantitative analysis, the key findings answered how the organizers attracted followers, how fast they gained their followers, and which users helped more in the fast spread of the information. (Hambrick, 2012)

In “Degree Centrality and Eigenvector Centrality in Twitter”, the authors applied Degree and Eigenvector centralities on Twitter data, to find the most influential nodes. The results showed significant difference between the two centrality measures. The influential nodes according to each algorithm were completely different, and Eigenvector would indicate nodes with low weight or degree. (Maharani, Adiwijaya, & Gozali, 2014)

In “Identification of Influential Nodes from Social Networks based on Enhanced Degree Centrality Measure”, the researchers combined the clustering coefficient value with degree centrality. The so-called Enhanced Degree Centrality Measure was applied to three Facebook datasets for

performance analysis. The results compared to those of Degree Centrality and SPIN, showed an increase in the spread achieved. (Srinivas & Velusamy, 2015)

In the journal article “Influence maximization on social graphs: A survey” (Li, Fan, Wang, & Tan, 2018), the researchers studied the problem of Influence Maximization from an algorithmic perspective. More specifically, they reviewed existing information diffusion models, they classified and compared such algorithms based on their objectives and studied the techniques in combination with social networks’ context attributes.

2.2 Technologies

2.2.1 Neo4j

Social networks represent social relations among entities. In other words, social networks are graphs and can be studied as such. In this study, three graphs are created and analyzed by importing social networks' data into Neo4j, the graph database system.

Neo4j is an open-source, distributed data store that is used to model graph problems. It was released in 2007 and is sponsored by Neo4j, Inc., which also offers enterprise licensing and support for Neo4j.

In Neo4j, data modeling is adaptable to changing requirements as the graph evolves. Through its flexibility, it captures new data sources, entities, and the relationships among them as they occur. The easy adaptation of the database to the changes provides quick responsiveness to changing requirements, making Neo4j an extremely agile solution.

Data modeling is a three-step process that involves the following steps:

1. Definition of requirements in the form of questions that need to be answered by the model
2. Identification of entities and their relationships
3. Configuration of patterns from the initial questions and their execution

This whole process is applied iteratively and incrementally and is repeated whenever the requirements change.

Neo4j implements the property graph data model, a relationship-centric approach to data modeling, where data is organized as nodes, relationships, and properties. This implementation is at the storage level, which means that nodes, edges, and properties are stored on disk in stores specific for each type. Then, the traversal of the graph is done with the use of pointers. It is designed to ensure that nodes and edges are stored efficiently and that nodes can be connected with any number of relationships of any type without sacrificing performance. This provides an efficient, flexible, and adaptive solution of storing any data in the form of nodes, properties, and relationships. Apart from that, Neo4j has evolved the traditional model to the labeled property graph, where nodes can be tagged with labels.

Mainly, in Neo4j, everything is defined in either one of the following forms:

- **Nodes**: the fundamental unit of a graph, which can also be viewed as an entity
- **Relationships**: the connection between two nodes
- **Properties**: attributes that are related to either nodes or relationships
- **Labels**: they are used to group nodes into sets

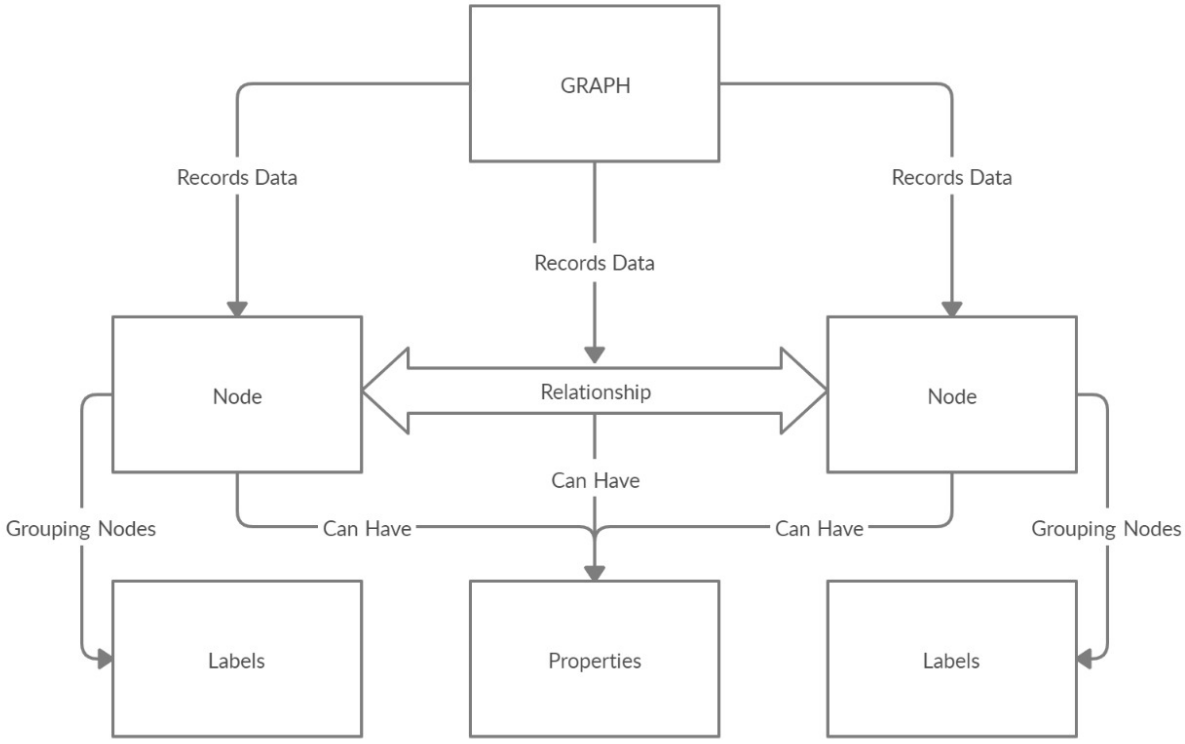

Figure 1:Neo4j's labeled property graph

The data is fetched from the data model through traversals. Traversals are an essential aspect of graphs, where given a starting node and following the relationships with other nodes, paths are created. Neo4j provides two types of traversal; the breadth-first and depth-first.

Neo4j is a schema-less or schema-optional graph database. The schema does not have to be defined unless there is a need to provide some structure to the data for performance reasons. In that case, the schema can be defined, and indices, constraints, and rules can be created over data.

Neo4j is an ACID-compliant and transactional database that ensures security, reliability, scalability, and high performance. It ensures that modification in the data happens in transactions to guarantee consistency. Also, because transactions in Neo4j dictate the state, idempotency is

ensured. Reapplying transactions for a recovery event results in replaying the transactions as of a given safe point.

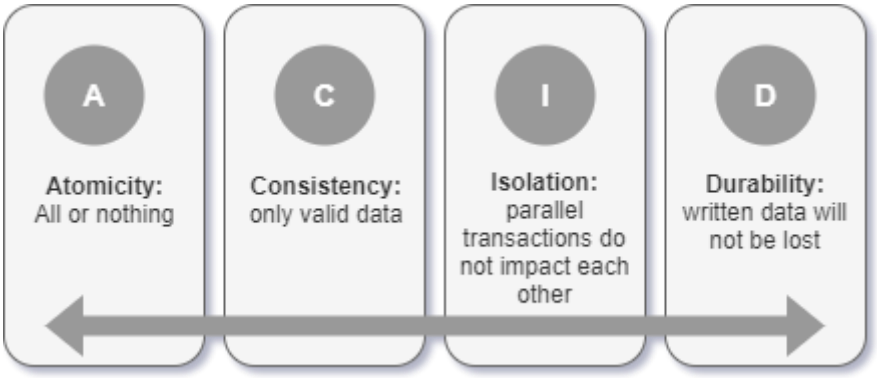

Figure 2: Neo4j is an ACID-compliant database

Neo4j is implemented in Java but can also be accessed by software written in other languages. It is supported by a vibrant ecosystem of libraries, tools, drivers, and guides provided by partners, users, and community contributors. The integrated tools sit on top of a common protocol, API, and a query language to provide effective access for different uses.

Diving deeper inside the Neo4j Graph Database and its native graph technology, compared to others that are non-native, the advantages of it become instantly apparent. It is obvious in technology that anything trying to be good at everything usually fails miserably, and as such, graph databases are not different.

Some of the functionalities commonly found with databases are:

- batch and transactional workloads
- memory access and disk access
- SQL and XML access
- graph and document data storage

DB developers usually need to identify upfront the use cases for which they should optimize their DBMS. Thus, if a graph is to be used heavily, they might decide to go “graph first,” aka native graph technologies or another DB where graphs were “patched” later on to satisfy a niche audience, known as non-native.

Native graph technology is split into two main categories: storage and processing.

Graph storage refers to the infrastructure used for storing graph data. When such storage is built with graphs in mind, it is called native. This type of storage considers the design of graphs, ensuring all data is stored efficiently, keeping connected nodes and relationships closely stored, thus increasing performance.

Non-native storage is external to the graph and decoupled from its logic. Other types are used in these cases, like relational, NoSQL, and other databases.

Finally, what makes Neo4j a right choice is that it shares many of the qualities of a traditional relational database management system while using an entirely different data model that is well suited for use cases with densely connected entities.

2.2.2 Cypher

One of the defining features of Neo4j is its query language, Cypher. Cypher emerged from the Neo4j graph database's evolution and was initially intended to be used only alongside Neo4j. It was mostly an invention of Andrés Taylor, an engineer in Neo4j, in early 2011. The openCypher project made Cypher available to everyone in late 2015 to standardize Cypher as the common graph query language, just like SQL is in the RDBMS world (Francis, et al., 2018).

Querying a graph database using the Java API would require visiting the whole graph and skip the nodes that don't match the requirements. Changes in the query would require to rethink the code, change it, and rebuild it. This is because an imperative language doesn't work well in pattern matching.

Cypher is a declarative query language that is used to query a Neo4j database. Being declarative means that it focuses on the results, rather than on retrieving them, making it human-readable and expressive. This helps the developers focus on the domain model instead of learning complicated procedures to access the database.

Data is evolving, not fixed, and, sometimes, could even be not known. Using object-oriented principles with a set of functions to access the data would mean revisiting the code regularly.

The solution to this problem is patterns and pattern matching. Cypher is based on these principles while being robust and straightforward.

- **Pattern**: Describes the shape of data that need to be found in a dataset
- **Pattern matching**: The process to find or match a pattern or sequence of patterns against a given dataset

Some well-known graph query languages that implement pattern matching and can be used to query graphs are SPARQL, GREMLIN, and MQL. However, they require much work in maintenance and enhancements. Also, they couldn't be the query language for Neo4j, as they failed to meet one or more of its primary goals:

- Declarative
- ASCII art pattern
- External DSL
- SQL familiarity
- Closures

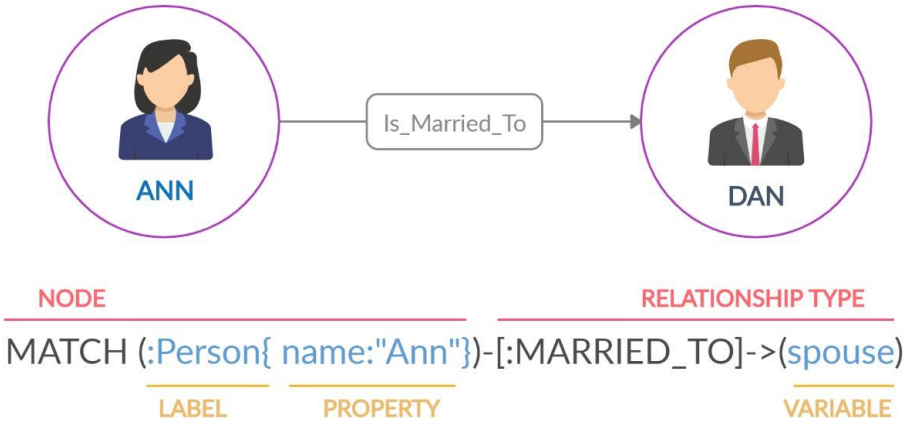

Figure 3: Cypher's ASCII art pattern syntax

Having considered all the above goals, Neo4j implemented a new declarative graph query language, Cypher, as a query language for the Neo4j graph database.

Cypher borrows much of its structure from SQL. This makes things easy for developers who are familiar with SQL. It also helped them achieve one of their primary goals of SQL familiarity. It was designed to be as powerful and capable as SQL is for Relational Database Management Systems but based on the components and needs of a graph database.

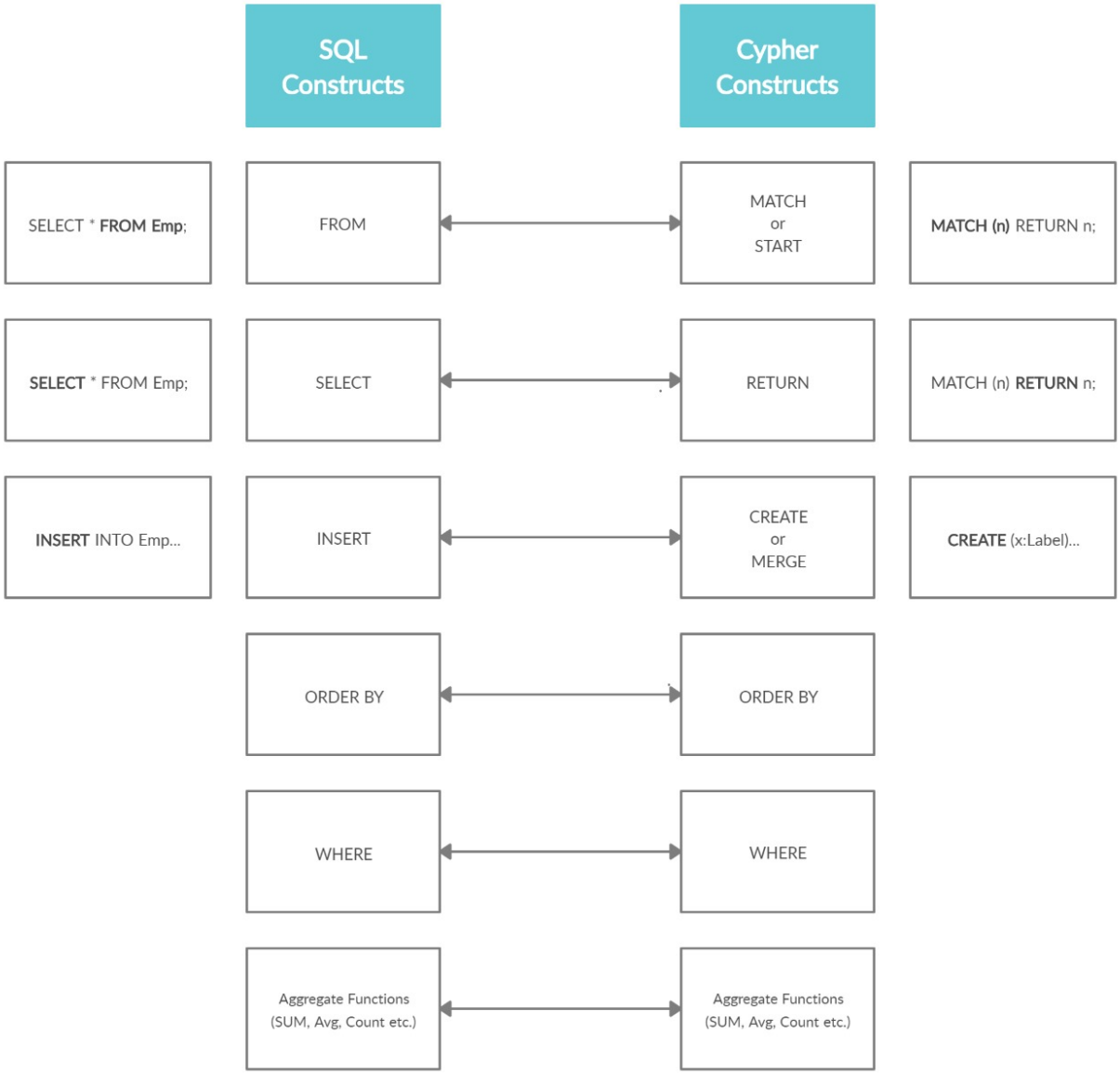

Figure 4: Similarity between the structure of Cypher and SQL

2.2.3 Graph Data Science Library

Graph Data Science (GDS) library is an open-source add-on for Neo4j that is used for graph analytics. It was developed by Neo4j's Product Engineering and was released in 2020. It provides

a set of high-performance standard graph algorithms for Community Detection, Similarity Calculation, Centrality, Pathfinding, and Link Prediction, exposed as Cypher procedures. It also provides APIs for the implementation of custom algorithms.

GDS library uses a specialized in-memory graph format to represent the data so that the algorithms run as efficiently as possible. So, before running an algorithm, it is required to load the data from the database to an in-memory computational graph, the graph catalog. The underlying data from the database is transformed into a data structure optimized for global traversals and aggregations for extremely efficient and scalable execution over large graphs. The data that is loaded can be filtered through graph projections.

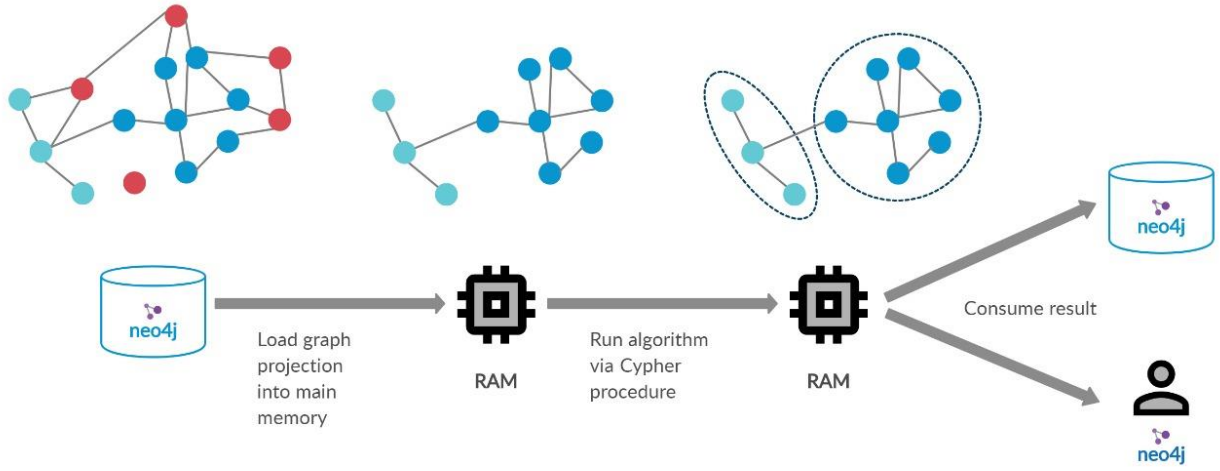

Figure 5: GDS library workflow

There is no limit in the number of computational graphs stored in memory. There can be multiple with different names and can be used by their name as a reference to execute algorithms against different projections. The results of running an algorithm as a Cypher procedure with GDS can be either updating the computational graph, stream results out, or writing the results back to the database.

2.2.4 APOC

APOC is the largest and most widely used library for Neo4j. It was developed by a developer working in Neo4j as a standard utility library for common procedures and functions. With the

library's use, developers across platforms can only focus on writing their functionality for business logic and business requirements specific needs.

APOC stands for both of the following:

- Awesome Procedures on Cypher
- A Package Of Components

It was first released in 2016, with version 3.0.12 of Neo4j. It includes over 450 standard procedures that are well-supported and easy to run as separate functions or included in Cypher queries.

APOC's collection of functions and procedures that are available for use in Cypher cover the following subjects:

- Collection operations (sorting, min, max, etc)
- Graph operations (indices and refactoring)
- Text search
- Conversions
- Ranking
- Geospatial operations
- Data integration
- Reporting
- Getting a meta graph representing the graph

2.2.5 Graph Algorithms

Graph algorithms are used to calculate metrics for graphs, nodes, or relationships. They can provide insights on relevant entities in the graph, patterns, or existing sub-structures like communities.

Many graph algorithms iteratively traverse the graph to compute the requested outputs using random walks, breadth-first or depth-first searches, or pattern matching. Because of the probably

immense number of possible paths with long distances, many of the approaches have high algorithmic complexity. There are optimized versions of algorithms that utilize specific structures of the graph, memorize already explored parts, and parallelize operations.

The graph algorithms are divided into the following groups that represent different graph problems:

- Community Detection
- Centrality
- Similarity
- Path finding
- Link prediction

The graph algorithms that will be used to find the most critical nodes in the graphs are called Centrality algorithms. The word "important" can be interpreted in many ways, resulting in many different definitions of centrality. Finding what makes a node important in a specific use case defines the most suitable algorithm to solve the problem. Common approaches to distinguish important nodes are the count of directly related nodes, the ability of a node to link multiple clusters, and the nodes being related to other important nodes.

3 Solution Architecture

In this solution architecture, the actor is a researcher studying graphs. They import data to create the graphs and run queries and procedures to explore them and enhance them. The actor's interaction with the graph database is happening via the graph's user interface, where they can use a query language to get the results desired. The outputs of this architecture may be the created graphs, query results, the results of algorithms or an enriched graph.

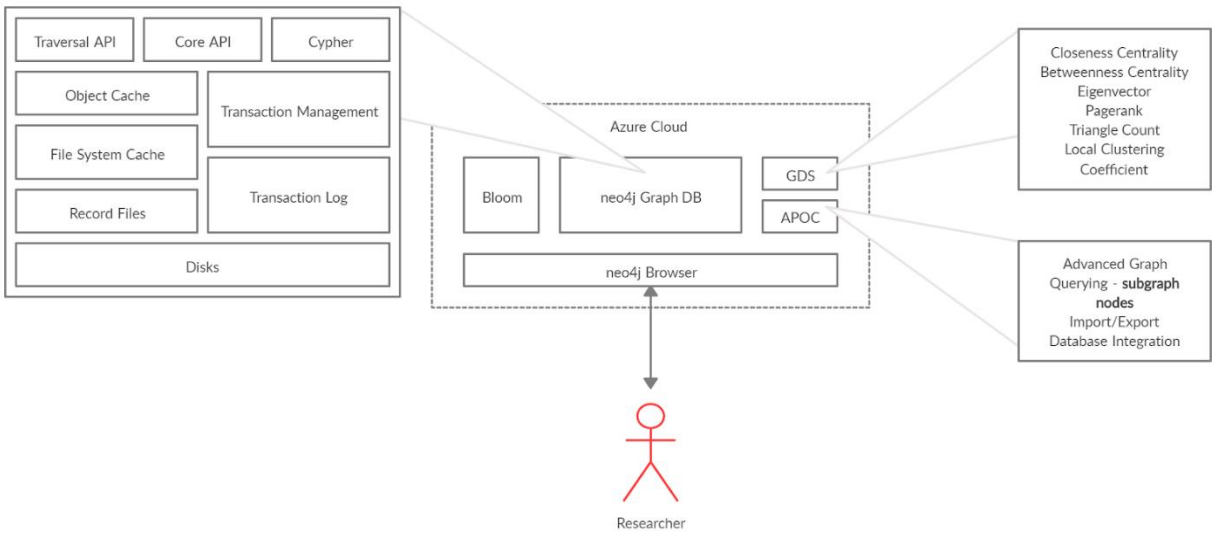

Figure 6: The main components of the suggested solution

As per Figure 6, all implemented and utilized modules live inside the Azure Cloud, Microsoft's Cloud SaaS, PaaS, and IaaS. This service was selected because of past personal experience over the equivalents from Amazon and Google. Combining the power of Neo4j, the leading graph database worldwide with the capabilities of Microsoft's cloud offering was key to extract the value from the imported data and conduct this research.

The depicted architecture is a visual representation of the main components that compose the suggested solution, as described within this dissertation's boundaries. In no way is it a complete architecture of the systems used. The modules displayed are the following:

- Neo4j Browser
- Neo4j Graph DB
- GDS library

- APOC library
- Bloom

Further below, a deeper analysis of each module will be performed to clarify what purpose each one serves and how they interact with each other.

3.1 Neo4j Browser

From the researcher's perspective, the neo4j Browser is the front-end module that the Researcher actor directly interacts with to access the rest of the system's functionalities. It's an online browser interface to query and view the data in the database.

There are two ways to access the neo4j Browser, either directly from a web browser or through the neo4j Desktop, by connecting to the Graph DB. Specifically, in this case, both ways were utilized for separate tasks depending on the performance requirements of each task.

This module offers rich graph visualization capabilities. Once a query is executed through the dedicated input field, the browser will populate the results directly in any of the three available and appropriate formats:

- Visual graph
- Table
- ASCII-table

The user may swap between them through the browser’s GUI and focus on specific areas of the query results through the visual view.

3.2 Neo4j Graph DB

This is the core graph database where all the connected data related to the platform are stored and retrieved from.

It is the most critical module of this system, the place where all nodes and relationships reside. There are no other persistence layers or mechanisms. It manages transactions and analytics and is optimized to traverse data by utilizing the relationships within the graph to discover connections in-between.

Below is a high-level overview of Neo4j's features.

3.2.1 Multi-database

Neo4j provides the capability to operate and manage multiple databases inside each installation of the Neo4j database management system. Data segmentation may apply depending on the use case and different needs that may arise, for example, separate databases.

3.2.2 Neo4j Fabric

Neo4j Fabric gives the option to break down graph data into smaller graphs and store them in different databases. These pieces can be accessed separately or combined to provide a view of all the data when needed.

3.2.3 Cypher

Cypher is a graph query language, as mentioned in the Technologies section, that enables reading and writing data into the graph. It is a simple yet powerful way to interact with both nodes and relationships.

3.2.4 Data access controls

Data security is a complex and significant task, all the more when handling sensitive information and regulatory aspects. There are many ways to secure data – access levels, roles, environments, permissions, architecture, etc. Neo4j provides these capabilities and regularly improves or adds to them.

3.2.5 Reactive drivers

Neo4j reactive drivers embrace the reactive manifesto (Bonér, Farley, Kuhn, & Thompson, 2014) and its principles by using specific drivers to pass data between the DB and clients. It is much faster and way more productive for individual professional developers to take advantage of this approach and process queries to return results. This influences communications between the driver and the database and can be managed dynamically based on each specific use case.

3.2.6 "Graph First" approach

Focusing on the left box of Figure 6 and the Neo4j graph database architecture, the one used for this research, it is visible that all layers are designed with graph data in mind. From Cypher queries

to disk storage, not a single decision was made without considering maximizing the speed of traversals during arbitrary graph algorithms.

The graph data are stored in files, segmented by particular areas of the graph.

- neostore.nodestore.db; stores node related data
- neostore.relationshipstore.db: stores relationship related data
- neostore.propertystore.db: stores the key/value properties
- neostore.labelstore.db: stores label related data

The data on the disk is all stored as linked lists of fixed-size records. The properties are stored as a linked list of property records that hold a key and a value and point to the next property. All nodes and relationships reference their first property record. Each node also references the first relationship in its relationship chain. Relationships reference their start and end nodes and the previous and next relationship records for the start and end node, respectively. By splitting data this way, highly performant graph traversals are enabled.

On the contrary, non-native graph storage uses a columnar DB or other generic data stores instead of being explicitly developed for graph technologies. Although the techniques and technologies employed may be familiar with most the engineers, the gap between the two implementations (graph data against non-graph storage) increases concerns and potential risks on performance and scalability. In the scenario where a non-native graph database is live on production and retrieval queries are executed upon it continuously, the algorithms used to write those data, due to the non-native nature of the infrastructure, will result in scattered data in distant places of the storage medium. Hence every time, in order to retrieve this data, they need to be reassembled from the storage, resulting in thousands of queries per minute.

Graphs, in general, are prone to errors when writing data due to their nature. Each time a connection is stored, three write operations must be made. One operation to store the relationship and two updates, one on each connected node. If one of these operations fails, the graph becomes corrupted. Hence fully ACID-compliant transactions are commonly used to ensure integrity.

Native graph processing, the other main element when speaking about native graphs, is about the way operations are processed by the graph database. Specifically, index-free adjacency makes the most impact on the performance of native graph processing.

Index-free adjacency is a concept designed to achieve the highest efficiency in storing and processing graph data. When writing data, this technique ensures that each node is stored directly next to its adjacent nodes and relationships, thus speeding up the process.

Reading is then even faster, making retrieval easy without the need for indexes, or at least heavily. On the other hand, non-native databases rely mostly on indexes to optimize specific use cases of querying data, while at the same time slowing down the operations by using too many them.

Index-free adjacency is way more efficient than working with indexes. Query times are proportional to the size of the graph, instead of increasing as the size of the data stored increases. Any large graph dataset, especially when speaking about Big Data, would become impossible to work with due to its size. Data graph queries execute at a constant pace regardless of the size of the graph.

Relationships are easier to find within a graph because they are considered first-class entities and, as such, are easier to traverse in any direction as well.

Figures 7 and 8 demonstrate the main differences between native and non-native graph queries.

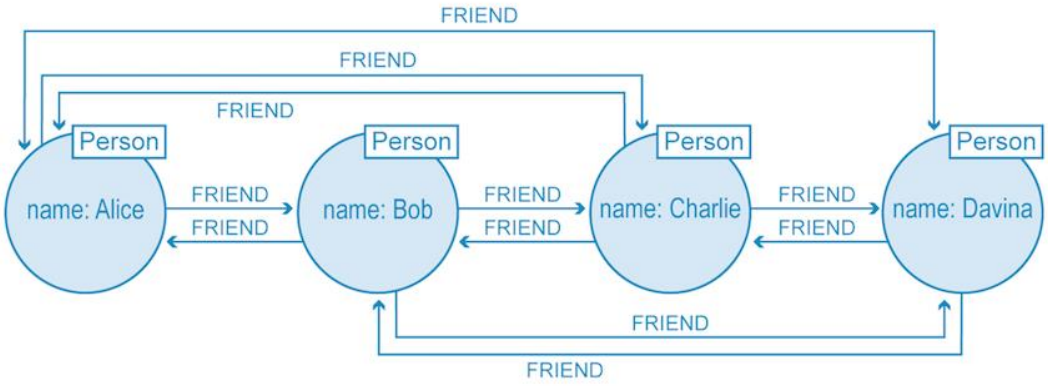

Figure 7:Native graph storage

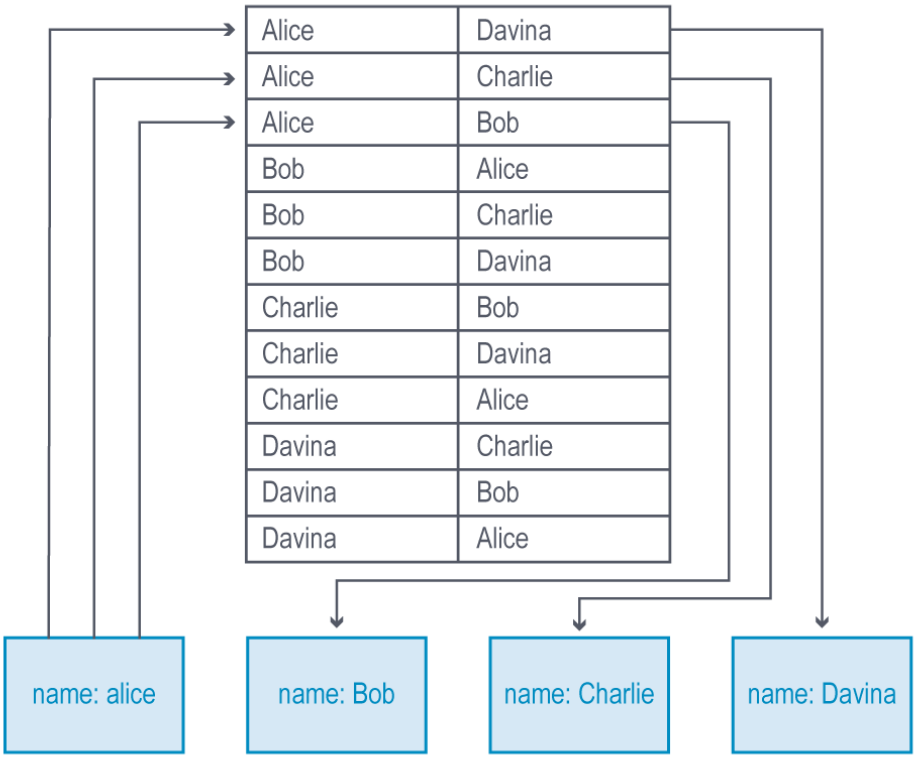

Figure 8: Non-native graph storage

To sum up, both native and non-native have their pros and cons. For example, a development team is more likely already familiar with a non-native graph database, or if the dataset is small the differences between the two are not important.

But if scaling up is likely and the datasets are expected to grow, then a native approach is the safest choice. Either way, every tool has its own purpose.

3.3 GDS library

As described in Chapter 2, this is an officially supported library and most commonly used for its graph algorithms. Graph algorithms enable the capability to analyze connected data as their calculations are built to operate on relationships. They enumerate steps to process a graph in order to discover its general qualities and specific quantities.

The ones mainly explored within the scope of this research are presented in Figure 6 on the top-right box:

- Closeness Centrality
- Betweenness Centrality
- Eigenvector
- PageRank
- Triangle Count
- Local Clustering Coefficient

3.4 APOC library

As described in Chapter 2, this is a standard utility library for Neo4j, including various procedures and functions. APOC was created to provide developers a well-written, consistent alternative to writing essential custom functions from scratch, thus avoiding possible duplication and creating technical debt. With the addition of APOC, the system leveraged its functionality to allow focusing mainly on business logic. It is the most popular library for neo4j at the time, containing more than 450 standard procedures that may run independently or included within Cypher queries.

Specifically, for this research, the subgraphNodes procedure was used mostly. This procedure finds subgraph nodes that can be reached from the start node by following the relationships.

3.5 Bloom

Bloom is a fairly new addition to the Neo4j products. It is a visualization tool that may cover a user's needs without any programming skills or other relevant knowledge. It provides data exploration through a GUI that enables users to navigate and query the connected data.

3.6 Block diagram summary

Figure 6 is a high-level block diagram depicting the modules used to complete this piece of work. It is certainly not a complete product, and as such, the actor's persona is a Researcher with the skillset needed to install, setup, manage and maintain the system as well as execute the appropriate queries to deduct their results. This could be used for future developments as a subset of tools to create a broader set of functionalities packaged in a much richer set of features. A typical block diagram, as described by Neo4j, with multiple different actors that interact with the system via front-ends to retrieve data and perform their researches, is displayed further below in Figure 9.

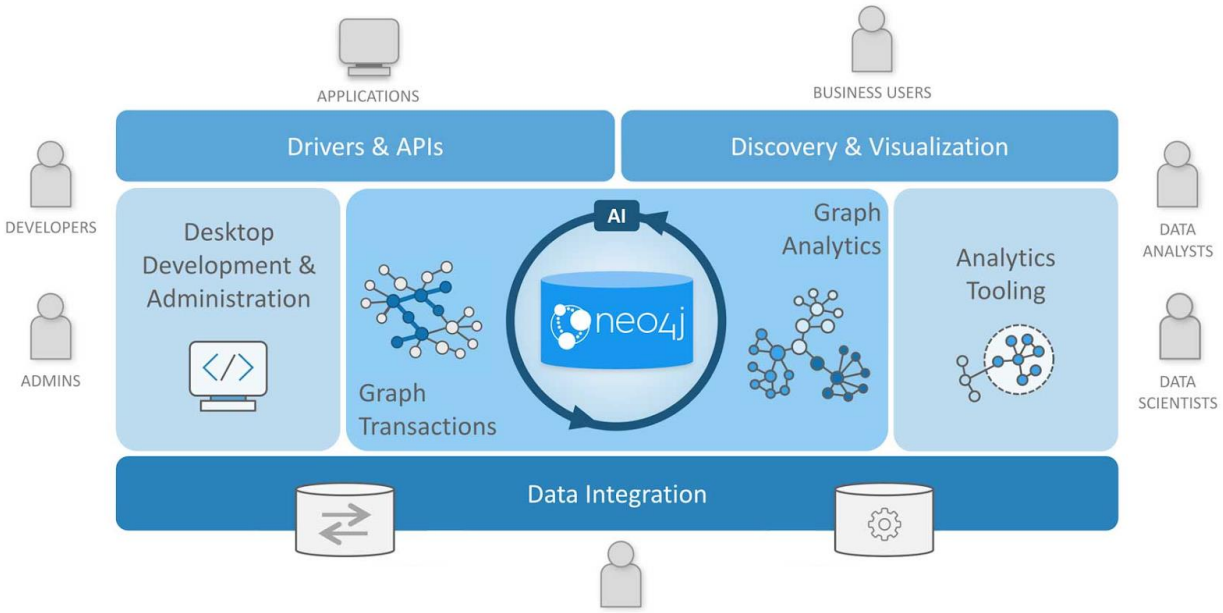

Figure 9: Neo4j block diagram (https://neo4j.com/product/)

4 Graph Algorithms

4.1 Centrality algorithms

The idea of centrality was first introduced in a research conducted by Alex Bavelas at the Group Networks Laboratory, M.I.T., in 1948. The studies were about human communication in small groups with the hypothesis that there was a relation between the structural centrality and the influence in group processes. That first research triggered many more, the results of which were often confusing or contradictory. Since then, many algorithms and methods are implemented to detect nodes that have some importance in a graph.

Centrality algorithms are used to understand better the roles of nodes in a graph and their impact on that network. They are useful because they help understand group dynamics such as accessibility, the speed at which things spread, and bridges between groups. Many of these algorithms were implemented for social network analysis. However, they are also being used in a variety of industries and fields.

4.1.1 Degree Centrality

The Degree Centrality algorithm is the simplest one in concept. It is often used as a baseline metric of a graph's connectedness. It was proposed by Linton C. Freeman in 1979 in his paper "Centrality in Social Networks: Conceptual Clarification", where he examined the concepts of both point and network centrality (Freeman L. C., 1978).

A node's degree is defined as the number of edges tied to it. In a directed graph, there is usually the need for two types of degree centrality, the in-degree, and the out-degree. The in-degree refers to the number of edges that are directed to the node, while the out-degree is the number of edges that start from the node directing to other nodes.

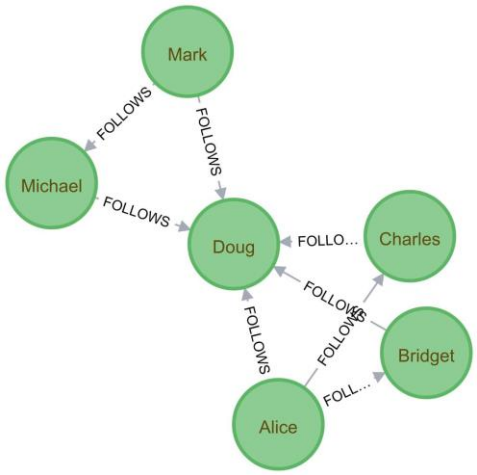

Figure 10: Doug has the highest in-degree score and Alice has the highest out-degree score

In social networks, a high in-degree of a node is an indication of popularity. In a study for an epidemic, such nodes are considered at high risk to get contaminated or spread the contamination. Although simple, degree centrality has significant usability in many networks and can provide valuable information for the graph.

A graph's average degree is the total number of edges divided by the total number of nodes. The average degree alone cannot provide much insight as it can be heavily skewed by high degree nodes. The degree distribution is the probability that a randomly selected node will have a certain number of edges. Those measures, along with the minimum degree, maximum degree, and standard deviation, provide a quick estimation of the graph's potential for spreading things.

Degree centrality is very useful when there is an interest in immediate connectedness or near-term probabilities. However, it is also applied to global analysis while exploring the entire graph.

4.1.2 Closeness Centrality

The closeness centrality of a node is defined as the average shortest distance of the node from all other nodes in the graph. The definition was made by Alex Bavelas in 1950, in his paper "Communication patterns in task-oriented groups", a psychological research in group communication.

Closeness centrality measures how central a node is to a graph by calculating the shortest paths to all other nodes. It is a way of detecting nodes that can spread information efficiently through a

subgraph. Nodes that have the highest closeness scores have the shortest distances from all other nodes.

The algorithm calculates the lengths of the shortest paths between all pairs of nodes in a graph and then calculates the sum of these distances for every node. The closeness centrality score for a node is the inverted result of its sum of shortest distances.

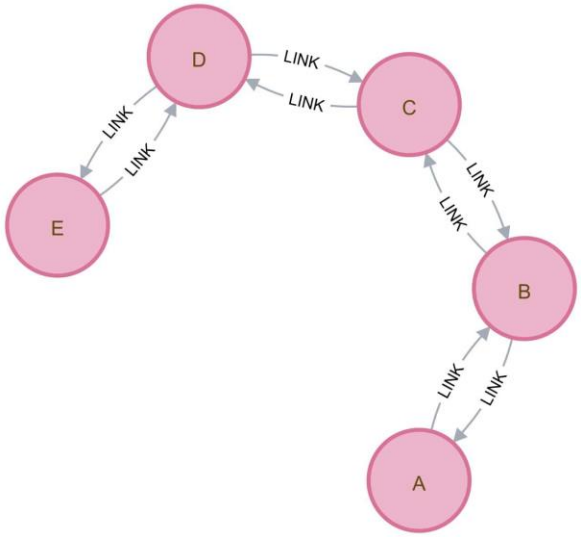

Figure 11: C is the best connected node in this graph

It is common to normalize these scores so that they represent the average lengths of shortest paths rather than their sums. This normalization allows comparing closeness centrality scores of nodes among graphs with different sizes.

The original algorithm, as explained, can only be applied to connected graphs, as the distance between disconnected components is infinite. If there is at least one node that is unreachable from all other nodes, its sum of distances from all other nodes is infinite. In practice, variations of the original formula are preferred to handle cases with disconnected groups.

A variation of the original formula was proposed by Stanley Wasserman and Katherine Faust in their book (Wasserman & Faust, 1994), which is an improved version for calculating closeness in graphs with many disconnected subgraphs. The formula returns the ratio of the fraction of nodes that are reachable in the group to the average distance from the reachable nodes. This formula is useful for detecting important nodes in the entire graph rather than within their subgraph, as nodes from small components will get lower closeness scores.

Another variation of the original algorithm is the Harmonic Closeness Centrality (Marchiori & Latora, 2000). Also known as Valued Centrality, it is a variation of the original Closeness Centrality algorithm, implemented to solve the issue with disconnected graphs. This improved version sums the inverse of the distances of a node to all other nodes. As a result, infinite values, caused by unreachable nodes, become irrelevant. As with the original formula, it is also possible to calculate the normalized harmonic centrality of nodes.

In general, closeness centrality is a very useful measure for finding nodes hat are best placed to influence quickly a big part of the graph. However, in a highly connected network it is very likely that all nodes will have similar scores. Sometimes, it is more useful to use Closeness Centrality to find important nodes within clusters.

4.1.3 Betweenness Centrality

Betweenness centrality was introduced in the paper "A set of measures of centrality based on betweenness" (Freeman L. , 1977) and is used to find those nodes in a graph that act as bridges among groups.

First, it calculates the shortest paths between every pair of nodes in the graph. The score for each node is based on the number of the calculated shortest paths that pass through the node. The nodes with the highest scores are those that lie on the biggest numbers of shortest paths.

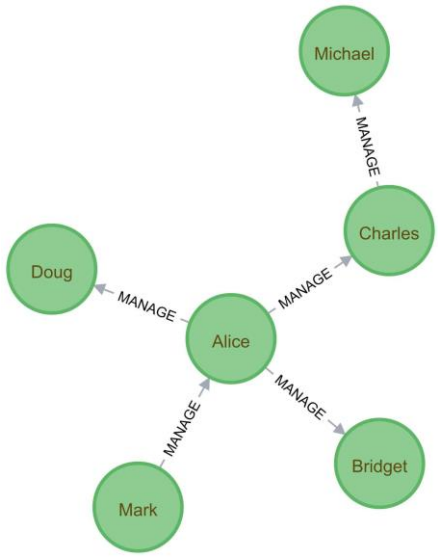

Figure 12: Alice is the main broker in this network

In this approach, the importance of the nodes lies in the fact that sometimes the most important ones in a graph are not those with the most connections. Nodes that connect groups may have the most control over them and are able to facilitate the flow of information. Betweenness centrality is typically used to find nodes that serve as a bridge from one part of a graph to another.

A node, as well as a relationship, can be considered as bridges in a network. They can be identified in a graph by searching for the node or relationship that would cause disconnection in the graph when removed. A group of nodes can also have a betweenness score if treated as a node.

Betweenness centrality is typically used to find bottlenecks, control points and vulnerabilities in a network.

4.1.4 Eigenvector Centrality

Eigenvector Centrality, or Eigencentrality, was proposed in the paper "Power and Centrality: A Family of Measures" (Bonacich, 1987) and was the first centrality measure considering the transitive importance of a node in a graph, rather than only its direct importance.

The nodes' scores are relative, based on the concept that nodes that connect to high-scoring nodes get a higher score. Two nodes with the same number of connections can have different eigenvector scores depending on the nodes with which they connect.

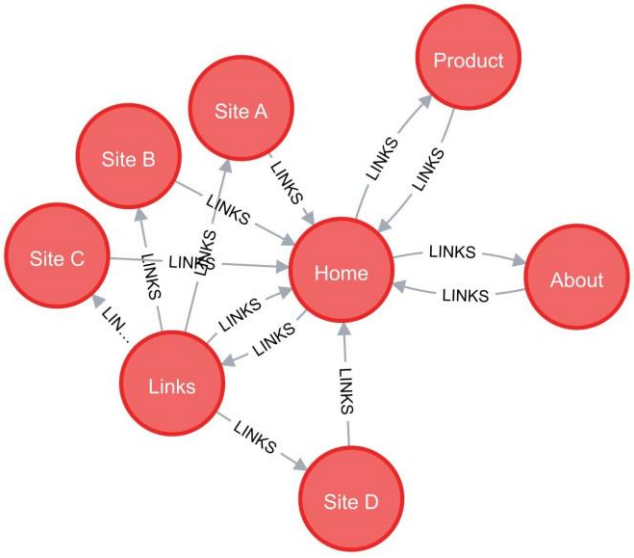

Figure 13: The Home page has the highest Eigenvector Centrality because it has incoming links from all other pages

A high score means that the node has a greater level of influence within the network. A node with a high degree score will have a relatively low eigenvector centrality score if its connections are with low-scored nodes. Also, a node with high betweenness centrality score would have a low eigenvector centrality score if it is far from the powerful nodes.

Unlike degree centrality, which weighs every contact equally, the eigenvector weighs contacts according to their centralities. Eigenvector centrality can also be considered as a weighted sum of not only direct connections but also indirect connections of every length. Thus, it considers the entire pattern in the network (Bonacich, 2007).

Eigenvector centrality is useful when there is a need to identify nodes that have a wide-reaching influence within a network.

4.1.5 PageRank

PageRank is a variant of the Eigenvector Centrality, that measures the transitive influence of nodes by considering the influence of neighbors and their neighbors. It was developed by Larry Page and Sergey Brin at Stanford University in 1996, as part of a research about a search engine. The idea behind their implementation was that information on the web should be ordered by link popularity. This means that a page should rank higher if there are many links to it by other pages. The first paper describing PageRank, as part of the Google search engine, was published in 1998 (Brin & Page, 1998).

In the beginning of the first iteration of the algorithm, all nodes have the same PageRank score. This value used to be 1, but in later versions of PageRank each node begins with a value between 0 and 1. In the end of the iteration, each node has granted equally its PageRank score to the nodes it links to. The PageRank transferred by an outbound link is equal to its own PageRank score divided by the total number of its outbound links.

In PageRank, links from a node to itself are ignored. Also, regardless of the number of links from a node to a node, as long as a connection between them exists, it counts as one. Nodes with zero out-degree are assumed to be connected to all other nodes in the graph.

PageRank’s algorithm takes into account a damping factor. This is the probability that an action will continue further with another hop. In case of a person that clicks randomly on links, they will

eventually stop clicking at some point. The probability, at every step, that they will continue with a new click is the damping factor and it is usually set around 0.85.

Although the PageRank is a variant of the Eigenvector centrality, the fact that it uses in-degrees makes it more suitable for use in directed graphs. However, it does not make sense to use PageRank in undirected graphs.

4.2 Community Detection Algorithms

A network is considered to have community structure when its nodes can be grouped into sets within which they are densely connected. Communities can overlap, as long as their members are more related within the group than with nodes outside the group. The identification of these groups provides valuable insight into group behaviors and emergent phenomena.

Community detection helps in revealing clusters, isolated groups and the overall graph's structure. These findings can help when there is a need to find similar behavior or preferences within groups. Also, Community detection algorithms are usually used early in an analysis to plan the next steps correctly based on the graph's characteristics. It is also common to use them to produce graph visualizations for general inspection.

4.2.1 Triangle Count

Triangle Count is a Community Detection algorithm that measures the number of triangles formed by nodes and to what degree the nodes tend to cluster together. A triangle is defined as a set of three nodes where each one has undirected relationships with the other two.

A Triangle Count run for a node, calculates how many triangles pass through the node and the probability that its neighbors are connected among them. It can also be used globally for the evaluation of the whole graph.

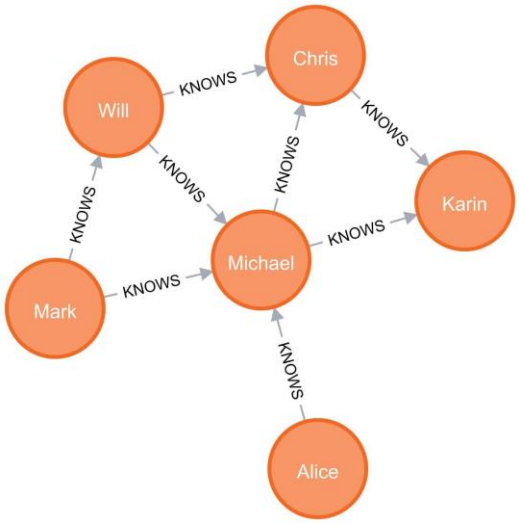

Figure 14: 'Michael' node has the most triangles

Triangle Count is useful when there is a need to determine a group's stability or as part of calculating general network measures. It is popular in social network analysis for detecting communities.

4.2.2 Clustering Coefficient

The Clustering Coefficient algorithm measures the degree to which a group is clustered compared to the maximum degree that it could be clustered. Triangle Count is used in its calculations to provide a ratio of existing triangles to possible relationships. The maximum value 1 means that the group is a clique with its every node connected to all other nodes.

The clustering coefficient algorithm can be run locally and globally.

The local clustering coefficient of a node is the probability that its neighbors are also connected among them. This computation involves triangle counting. A node's clustering coefficient can be found by multiplying the number of triangles passing through it by two and then dividing that by the maximum number of relationships in the group, which is equal to the degree of that node, minus one.

The global clustering coefficient is the normalized sum of all the local clustering coefficients. Clustering coefficients is an effective means to find obvious groups like cliques, in which every node is related with all other nodes, but there is also the ability to specify thresholds to set levels.

Clustering Coefficient calculates the probability that random nodes will be connected. It can also be used to quickly evaluate the cohesiveness of a specific group or the whole graph. The Clustering Coefficient and Triangle Count algorithms are usually presented together because they are often used together. These algorithms combined are used for resiliency estimation and exploration of network structures.

4.2.3 Weakly Connected Components

The Weakly Connected Components, or just Connected Components algorithm finds subgraphs in an undirected graph, in which every node is reachable from every other node of that same group, without considering the direction of relationships.

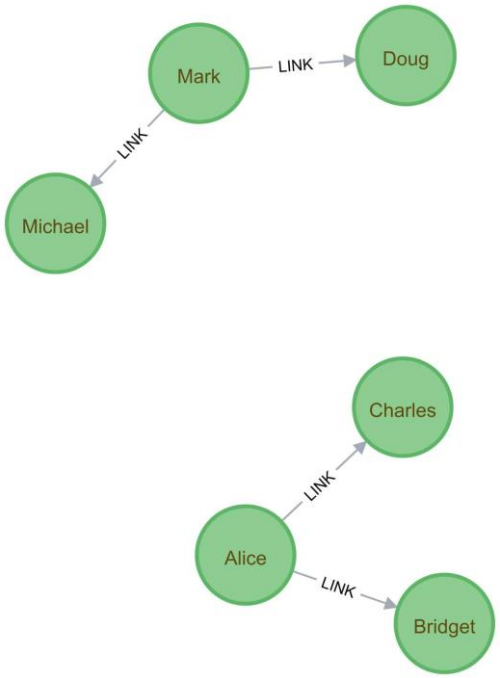

Figure 15: The graph has two weakly connected components, each with three nodes

It is often used in early steps of analysis to understand the graph's structure. It is useful to run Connected Components as a preparatory step for general graph analysis to test whether a graph is connected. This quick test helps to avoid running algorithms on a disconnected component of a graph and getting wrong results.

4.2.4 Strongly Connected Components

Strongly Connected Components is run on a directed graph to discover sets of connected nodes in which all nodes are reachable from all other nodes in the same set, following the direction of relationships, but not necessarily directly.

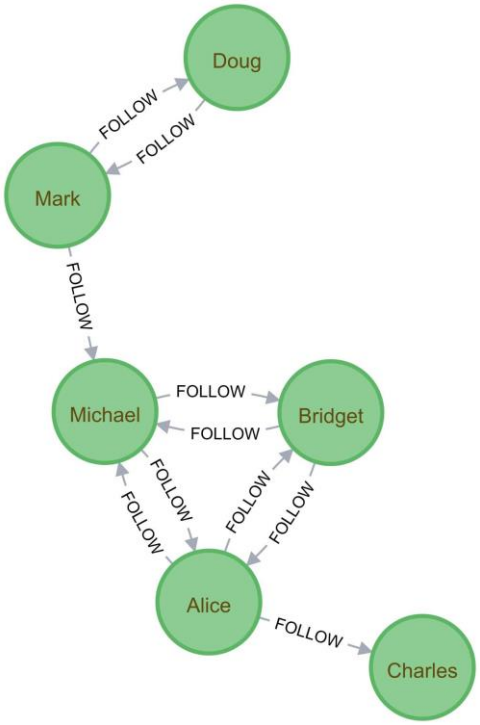

Figure 16: The graph has three strongly connected components

It differs from the Weakly Connected Components algorithm because it needs a path to exist in both directions, whereas Strongly Connected Components only needs a path to exist between two nodes regardless of the directions.

Strongly Connected Components algorithm is also used in early steps of graph analysis to see the graph structure or for tight clusters identification that might need independent investigation. A strongly connected component can be useful for profiling similar behaviors or trends for applications like recommendation engines. It can also be used to discover and collapse groups into single nodes for further analysis.

5 Experimental Study

5.1 Experimental methodology

The problem investigated in this experiment is the potential information spread in social networks by using common centrality algorithms. For this purpose, datasets were collected that contain relationships among social network pages. The datasets were chosen based on their stats, sizes and because of their identical structures.

To run the experiment, Neo4j was selected as the graph database management system to store, process and query the graphs. The choice was made mainly because of Neo4j's fast read and write performance, its installation options and the query language that accompanies it, Cypher. Because of the expensiveness of the graph algorithms that were evaluated, the database was hosted in Microsoft Azure cloud, in a virtual machine with enough memory and disk space to run them efficiently.

After the Azure VM and Neo4j installations, the data was imported into the database. The graph analysis process followed to help get an idea of the graphs' structures. In this phase, community detection algorithms were used to explore the graphs' connectedness. Also, the graphs' stats were calculated and gathered, such as the number of nodes and edges, the degree distribution and the density.

In the next phase, for each graph, the centrality scores from the five different centrality algorithms were calculated and written as node properties. The centrality algorithms chosen for evaluation for this experiment's purpose were Degree, Closeness, Betweenness, Eigenvector and PageRank.

Next, for each graph and for each centrality measure, the n nodes with the highest centrality scores were found. The number n of the top nodes was calculated as the 0.05% of total nodes in the graph. Then, for these nodes, the distinct nodes that direct to them were found beginning with neighbor nodes. The maximum distance from the top nodes was increased by 1 in each step, until the number of distinct nodes stopped increasing.

After having collected all the results and converted them to percentages of the total graphs, the algorithms' performances were interpreted and compared for each graph. The results show for each centrality algorithm the percentage of the graph that will directly receive the information, the

maximum percentage of information spread that it can achieve, the spread's rate and the number of hops that are required to achieve the maximum spread.

5.2 Datasets

Facebook is a social network that allows users to create their profiles and interact with each other. As it is used worldwide by millions of users, it provides a great example of a large social graph, for which one can find multiple open datasets that are available for research.

For the purpose of this study, datasets are chosen that are provided by the Network Repository and represent Facebook pages and the "likes" among them. The three datasets (Ahmed, Rossi, & K., 2015) concern pages of artists, public figures and politicians and all three have the exact same format where a node represents a Facebook page and an edge represents the like from or to a page.

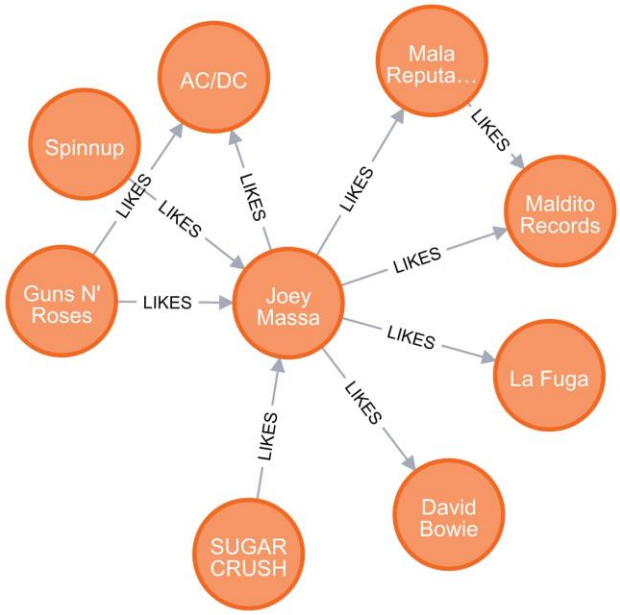

Figure 2: Sample subgraph of the artists' Facebook pages graph

The chosen datasets are quite simple in their structure with one node having two properties and the nodes connecting with directed relationships of one type.

Node	Properties
Artist | PublicFigure | Politician	Id, name

Relationship	Properties
LIKES	None

5.2.1 Artists Facebook pages

The initial graph model for the Artist Facebook pages dataset is shown in the Figure below. An Artist node can be linked to another or the same Artist node with the LIKES relationship type. An artist has two properties; the artist's id, which is unique, and their name. The graph is directed, with the relationship "LIKES" showing if a page likes another one or is liked by it.

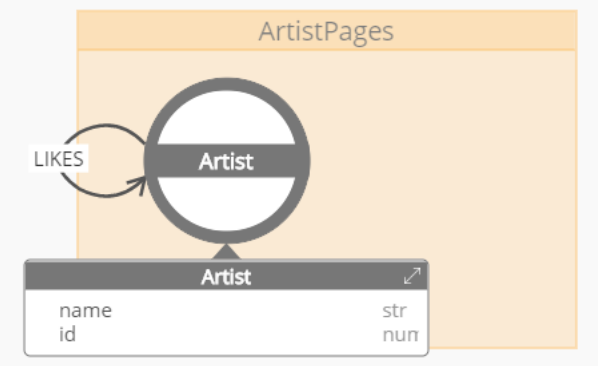

Figure 3: The Artist Facebook pages graph model

The data is imported in Neo4j with the following Cypher commands that read the CSV files which are uploaded in Neo4j's import folder. The first command creates the nodes and the second one creates the edges that link them.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM "file:///fb-pages-artist.nodes.csv" AS row
WITH row WHERE row.new_id IS NOT NULL
MERGE (:Artist {id: row.new_id, name: row.name})
```

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM "file:///fb-pages-artist.edges.csv" AS row
MATCH (source:Artist {id: row.src})
MATCH (destination:Artist {id: row.dst})
MERGE (source)-[:LIKES]->(destination)
```

The graph consists of 50492 nodes connected with 819096 edges. To further understand the graph and draw better conclusions later, it is examined with Cypher queries. First, the existence of self-linking nodes is checked. Such an existence means that we should count them while calculating the maximum number of relationships the graph could have, in order to have a view at the graph's density.

```
MATCH (a)-[r:LIKES]->(a) RETURN a,r, count(a)
```

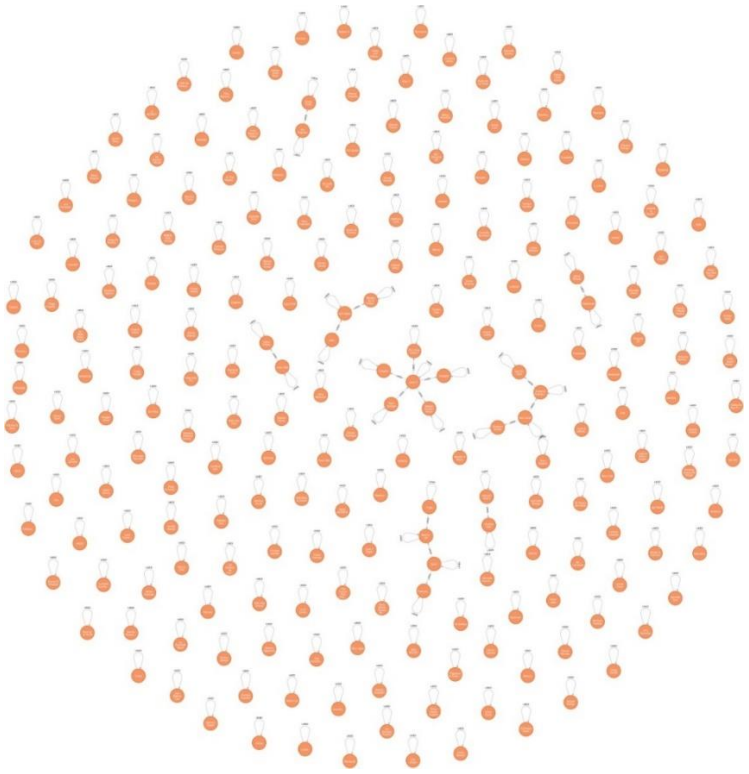

Figure 4: Artists-216 nodes that link to themselves

There are 216 "self-liking" nodes, so the maximum number of relationships between all artists would be 50492^2.

The node with the highest number of incoming relationships has 1290 likes, whereas the highest number of pages liked by a single node is 1237. These high numbers are far from the low average values, and the low standard deviations show that most nodes do not have so many relationships among them. For the purpose of this study, only the number of incoming relationships matters and only this will be referred as "likes". The reason is that a piece of information in a page is directly visible only to the pages that like them.

Minimum Likes	0
Maximum Likes	1290
Average Likes	16.22
Standard Deviation Likes	37.84

Table 1:The artist nodes' in-degree distribution

As shown in the pie chart below, there are many nodes with 0 likes. Most nodes receive up to 50 likes and there are less than 10 nodes that are liked by more than 1000 pages.

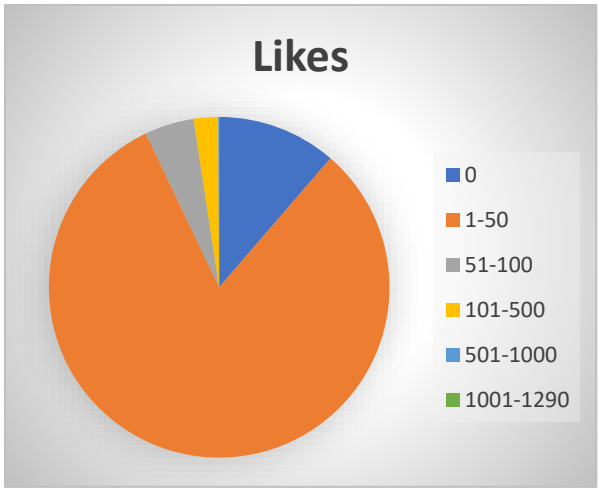

Figure 17: Likes grouped by ranges

The nodes that have 0 in-degree are 2340. This means that 4.63% of the graph's nodes cannot spread the information at all.

According to the frequency histogram in Figure 18, the graph can be assumed to be scale free, meaning that it follows a power law. This means that the characteristics of the graph are regardless of the number of nodes. This is plausible as a social graph grows over time and pages that are liked by many tend to attract more than pages with small numbers of likes. This is the idea of preferential attachment, that is particularly observed in social networks.

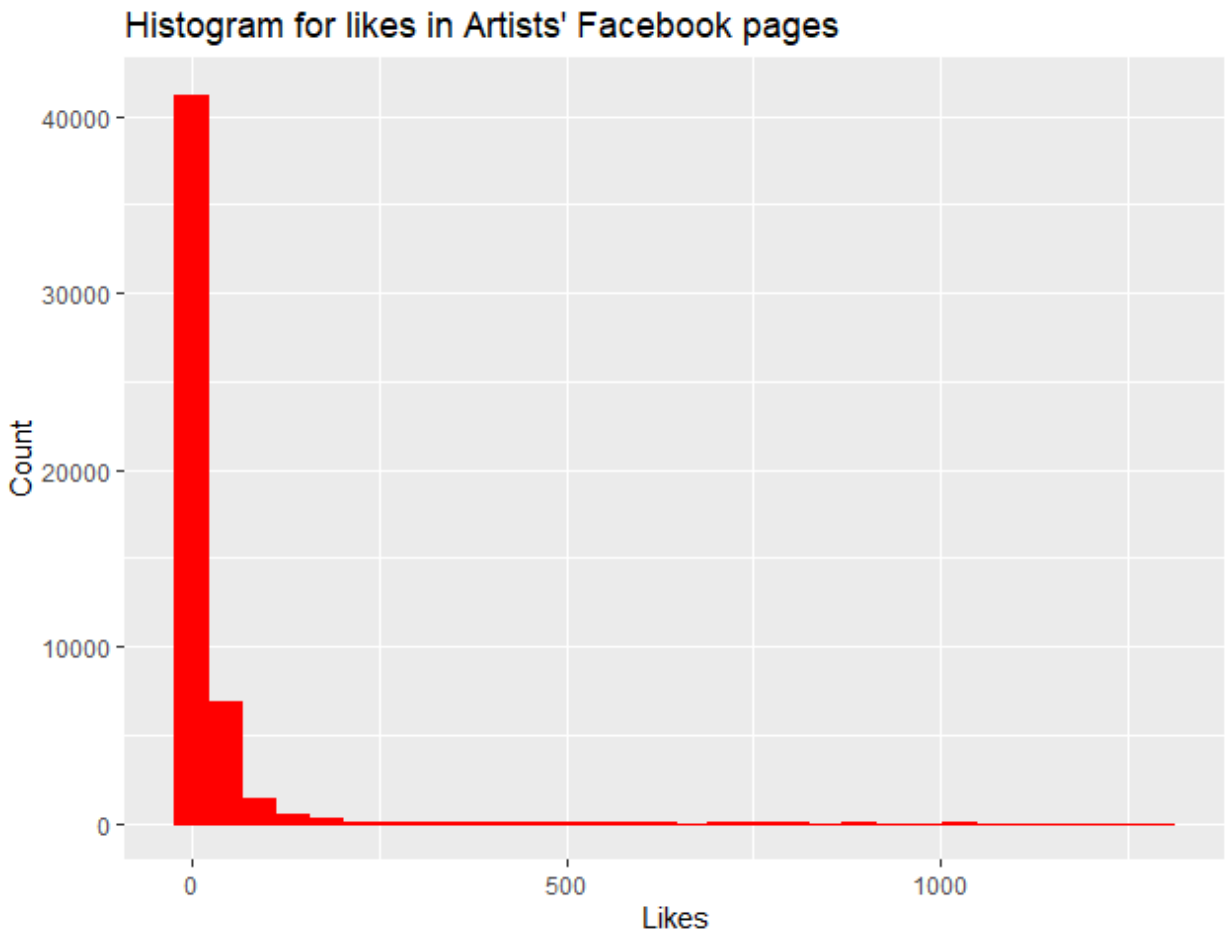

Figure 18: Frequency histogram for number of likes per artist page

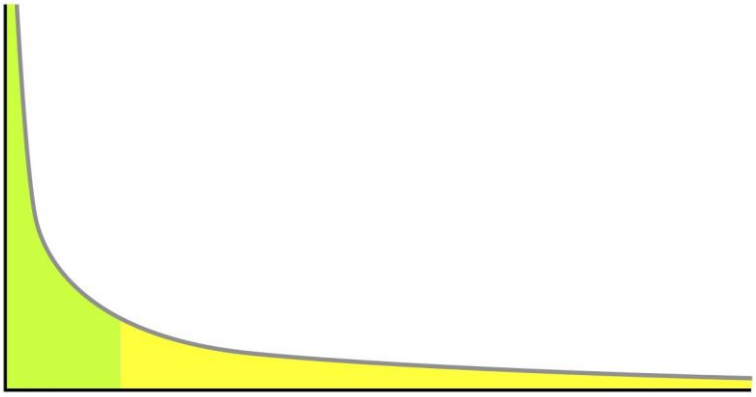

Figure 19: An example power-law graph. By User:Husky - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=1449504

With the use of GDS library, a graph “myGraph” is created, as required, to run the Triangles Count, Local Clustering Coefficient, Weakly Connected Components and Strongly Connected Components algorithms.

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

```
CALL gds.triangleCount.stats('myGraph')
YIELD globalTriangleCount
```

```
CALL gds.localClusteringCoefficient.stats('myGraph')
YIELD averageClusteringCoefficient
```

```
CALL gds.wcc.stream({
  nodeProjection: 'Artist',
  relationshipProjection: 'LIKES'
})
YIELD nodeId, componentId
RETURN gds.util.asNode(nodeId).name AS Name, componentId AS Component
```

```
CALL gds.alpha.scc.stream({
  nodeProjection: 'Artist',
  relationshipProjection: 'LIKES'
})
YIELD nodeId, componentId
RETURN gds.util.asNode(nodeId).name AS Name, componentId AS Component
```

Finally, the graph’s stats are collected and shown below.

Nodes	50492
Edges	819096
Density	0.0003
Average Degree	32.4
Minimum Degree	1
Maximum Degree	1468
Global Triangle count	1423840
Average Clustering Coefficient	0.097
Weakly Connected Components	1
Strongly Connected Components	50492

Table 2: Artist Facebook pages graph stats

5.2.2 Public Figures Facebook pages

By running the same Cypher queries as for the first dataset, with nodes labeled with "PublicFigure", the new graph is created, and its stats are gathered in Table 3.

Like in the first dataset, the existence of self-linking nodes is checked, and 76 self-liking nodes are discovered.

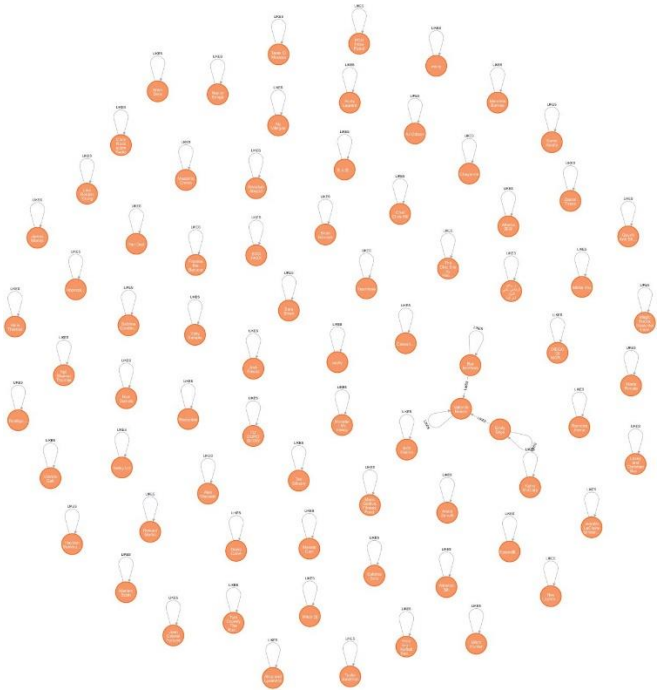

Figure 20: Public Figures-76 nodes that link to themselves

In public figures graph, 1926 nodes have 0 in-degree. This means that 16.67% of the graph nodes cannot spread the information at all.

Minimum Likes	0
Maximum Likes	274
Average Likes	6.054
Standard Deviation Likes	12.93

Table 3: The public figure nodes' in-degree distribution

In the frequency histogram below, it is observed that public figures distribution is similar to that of artists but with lower values due to the graph's size. The average in-degree of the graph is much smaller than the maximum. The graph, same as with the previous one, can be assumed to be scale free.

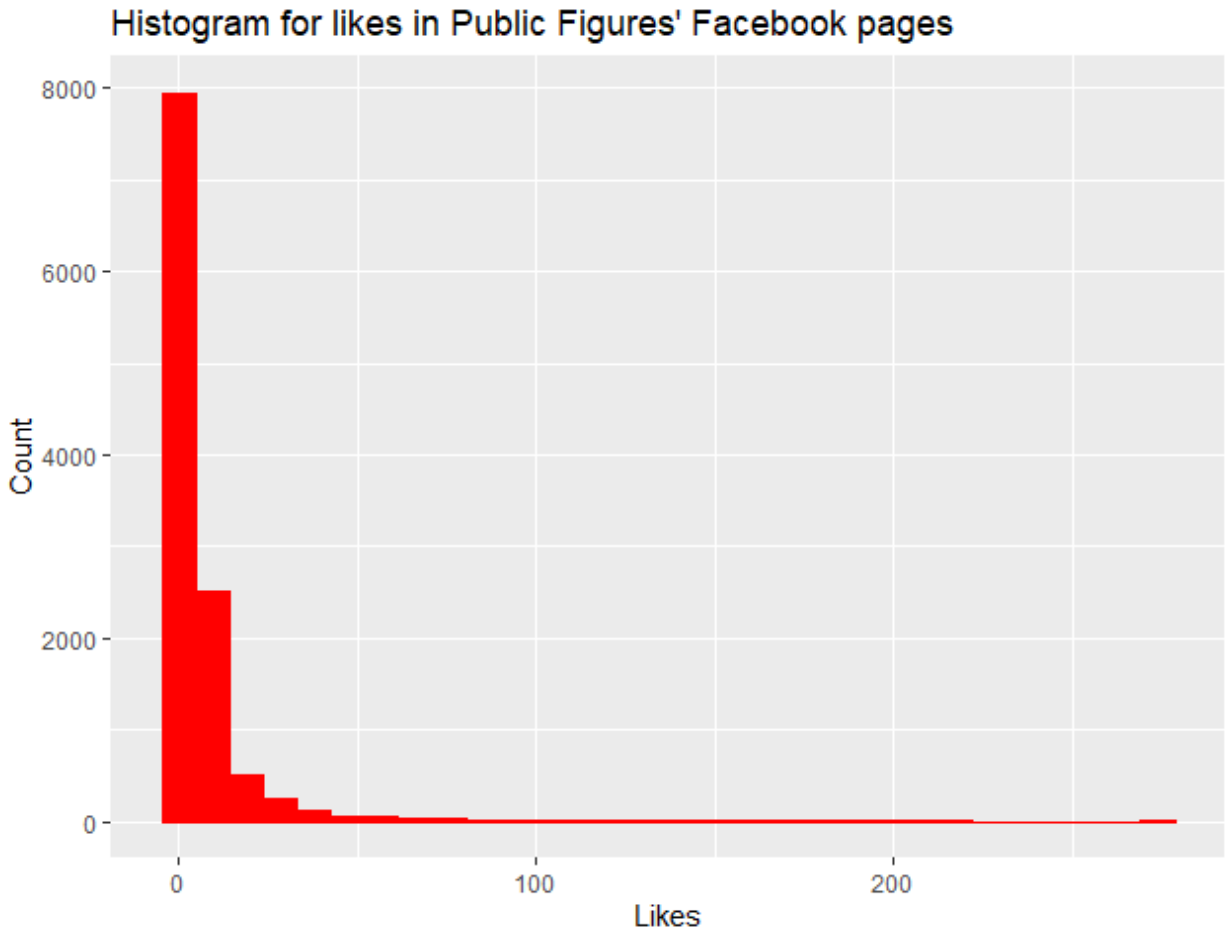

Figure 21: Frequency histogram for number of likes per public figure page

After running the Weakly Connected Components and Strongly Connected Components algorithms, the results show that the graph is connected and 9185 strongly connected components were discovered. This actually means that there is one strongly connected component of 2375 nodes and the remaining 9184 nodes form single node components with paths of 0 length.

Nodes	11559
Edges	67099
Density	0.0005
Average Degree	11.6
Minimum Degree	1

Maximum Degree	326
Global Triangle count	13706
Average Clustering Coefficient	0.129
Weakly Connected Components	1
Strongly Connected Components	9185

Table 4: Public Figure Facebook Pages graph stats

5.2.3 Politicians Facebook pages

The third graph created, which consists of Facebook pages of politicians, has 5908 nodes connected with 41729 edges. Again, there are 23 self-liking pages and 1011 pages have 0 likes, meaning that 17.11% of pages cannot spread information.

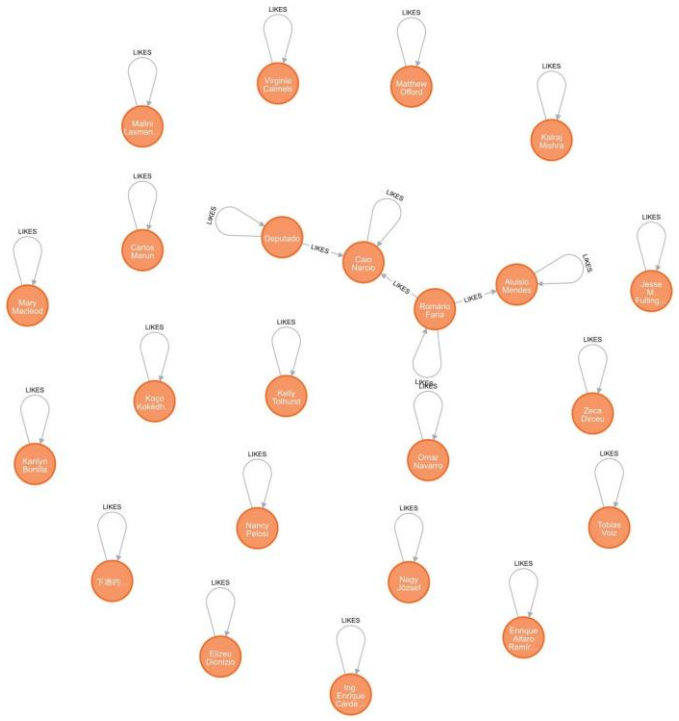

Figure 22: Politicians-23 nodes that link to themselves

Minimum Likes	0

Maximum Likes	225
Average Likes	7.06
Standard Deviation Likes	12.75

Table 5:The politician nodes' in-degree distribution

As with the previous graphs, the network seems to be scale free. 17 pages have more than 100 likes and only two of them are liked by more than 200 other pages.

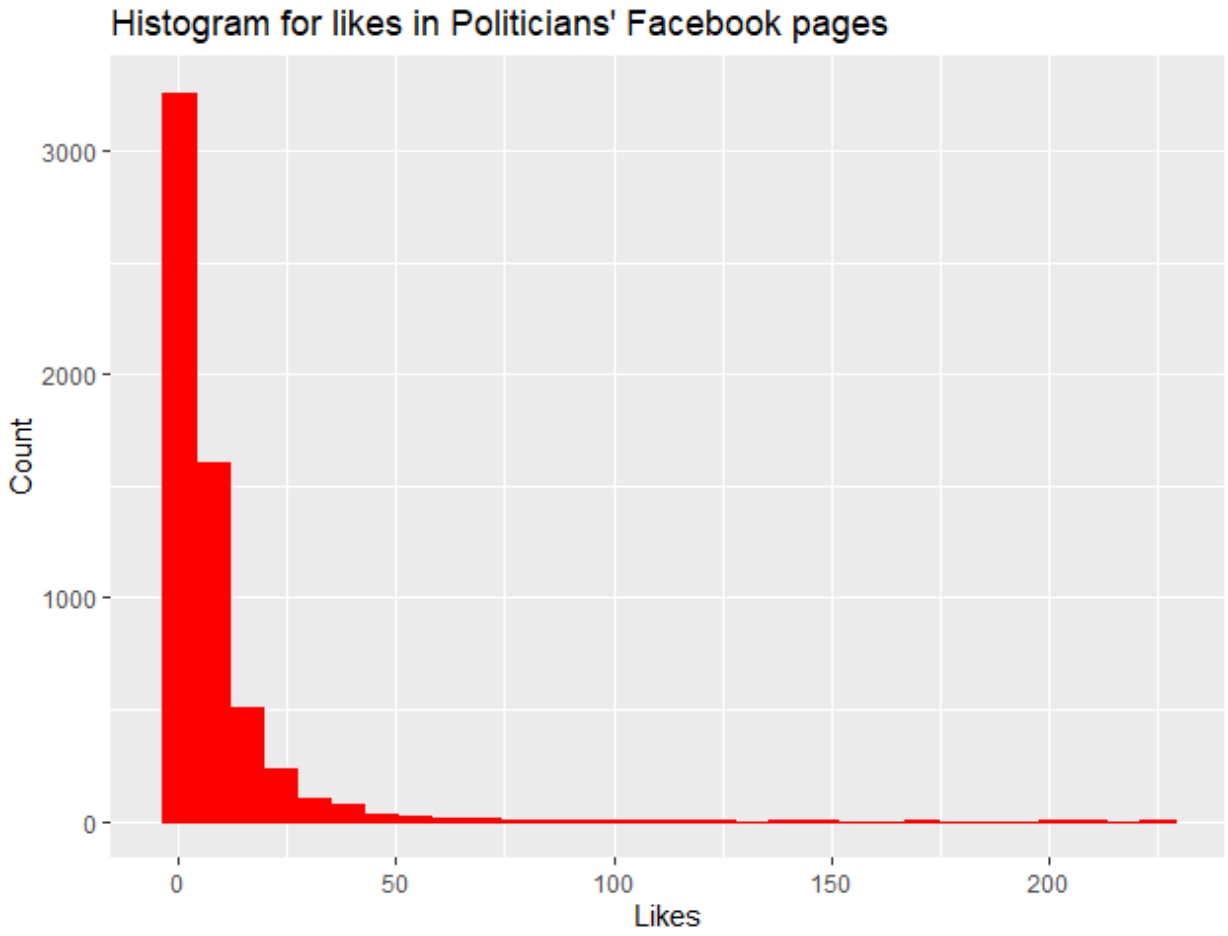

Figure 23: Frequency histogram for number of likes per politician page

Finally, as shown in the graph's stats below, the graph is connected, with practically no strongly connected components, as the 5908 strongly connected components are single nodes.

Nodes	5908

Edges	41729
Density	0.0012
Average Degree	14.1
Minimum Degree	1
Maximum Degree	323
Global Triangle count	127858
Average Clustering Coefficient	0.289
Weakly Connected Components	1
Strongly Connected Components	5908

Table 5: Politicians Facebook Pages graph stats

5.3 Data processing

In this phase, the graph models are being enriched with new properties that will hold the values of the centrality scores as computed with each centrality algorithm.

5.3.1 Degree Centrality

The following command sets a new property on the Artist nodes. The "likes" property of a node holds the number of pages that like the node.

```
MATCH (p:Artist | PublicFigure | Politician)
with p, size((p)<-[:LIKES]-()) AS likes
set p.likes = likes
```

5.3.2 Closeness Centrality

In this command, GDS library's closeness centrality algorithm is used which writes on each node its centrality score as a property named "closeness_centrality".

```
CALL gds.alpha.closeness.write({
  nodeProjection: 'Artist | PublicFigure | Politician',
  relationshipProjection: 'LIKES',
  writeProperty: 'closeness_centrality'
}) YIELD nodes, writeProperty
```

5.3.3 Betweenness Centrality

Here, GDS library's betweenness centrality algorithm is used for calculating the score for each node and writing it in a new node property "betweenness_centrality".

```
CALL gds.alpha.betweenness.write({
  nodeProjection: 'Artist | PublicFigure | Politician',
  relationshipProjection: 'LIKES',
  writeProperty: 'betweenness_centrality'
}) YIELD nodes, minCentrality, maxCentrality, sumCentrality
```

5.3.4 Eigenvector Centrality

In the following command running the eigenvector centrality algorithm, the scores are calculated with maximum normalization, meaning that all scores are divided by the maximum score. The scores will be written in node properties called "eigenvector".

```
CALL gds.alpha.eigenvector.write({
  nodeProjection: 'Artist | PublicFigure | Politician',
  relationshipProjection: 'LIKES',
  normalization:'max',
  writeProperty: 'eigenvector'
})
YIELD nodes, iterations, dampingFactor, writeProperty
```

5.3.5 Page Rank

The last algorithm chosen for evaluation, PageRank, is run with the following command with damping factor set to 0.85 and maximum number of iterations set to 20. The PageRank scores are stored in "pagerank" node properties.

```
CALL gds.pageRank.write('myGraph', {
  maxIterations: 20,
  dampingFactor: 0.85,
  writeProperty: 'pagerank'
})
YIELD nodePropertiesWritten AS writtenProperties, ranIterations
```

5.3.6 Final data model

After calculating the centrality scores and writing them on the nodes as properties, the data model in its final version is shown in Figure 5.

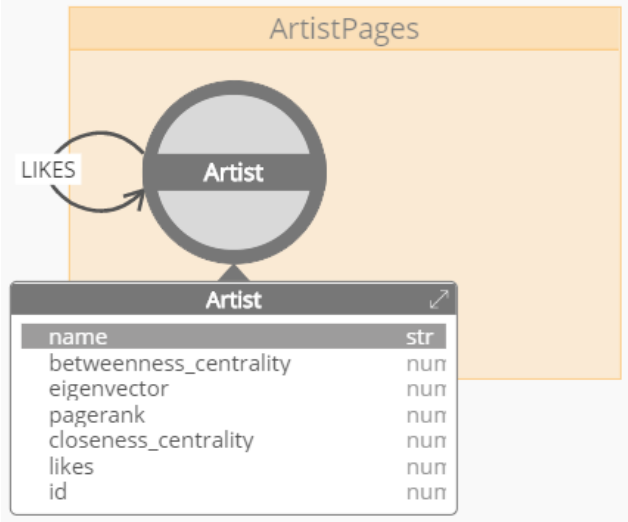

Figure 24: The Artist data model with the centrality scores added as node properties

5.3.7 Centrality Scores

	Degree	Closeness	Betweenness	Eigenvector	PageRank
1	Yuna Alvara (1290)	Beyoncé (0.3987)	Justin Bieber (14005427.15)	Calvin Harris (1.0)	The Police (84.60)
2	Beyoncé (1230)	Lady Gaga (0.3970)	Michael Jackson (13061464.76)	Yuna Alvara (0.81)	Robert Downey Jr (73.52)
3	PLASTIKO (1053)	Yuna Alvara (0.3967)	Adele (11043352.61)	F4ST (0.74)	Joseph Capriati (72.20)
4	The Beatles (1018)	Adele (0.3961)	Coldplay (9199130.96)	Diplo (0.72)	Excision (71.26)
5	Diplo (1013)	Taylor Swift (0.3955)	Rihanna (8869733.01)	Joseph Capriati (0.70)	Yuna Alvara (66.64)

Table 6: Top artist nodes by centrality score

	Degree	Closeness	Betweenness	Eigenvector	PageRank
1	Oprah Winfrey (274)	Daniel Amos (0.348)	Ross Mathews (13869310.54)	成語蕎 Jenny Cheng (1)	Dwayne The Rock Johnson (25.69)
2	煎熬弟 鍾明軒 (221)	Dwayne The Rock Johnson (0.3479)	Susie Meister (13795055.96)	熊熊 Bear Genie (0.65)	Oprah Winfrey (25.28)
3	Sergio Carlo (207)	Hugh Jackman (0.346)	Tony Robbins (12872679.63)	白吉勝&徐小可 Love 白宮這一家 (0.62)	Hugh Jackman (20.39)
4	Lulu 黃路梓茵 (204)	Oprah Winfrey (0.342)	Kris Carr (10076673.25)	小小瑜 張芯瑜 (0.62)	Daniel Amos (19.04)
5	白吉勝&徐小可 Love 白宮這一家 (203)	Sergio Carlo (0.339)	Latrice Royale INC. (9736722.34)	蘇哥哥 蘇銘翔 (0.55)	貝兒 Joannabelle (18.06)

Table 7: Top public figure nodes by centrality scores

	Degree	Closeness	Betweenness	Eigenvector	PageRank
1	Barack Obama (225)	Barack Obama (0.3588)	Manfred Weber (207104.25)	Johannes Schraps (1)	Sir Peter Bottomley MP (27.77)
2	Katarina Barley (208)	Mariya Gabriel (0.3235)	Hillary Clinton (158919.74)	Klaus Mindrup (0.79)	Barack Obama (23.89)
3	Joachim Herrmann(200)	Michael Roth (0.3205)	Michael Roth (143227.16)	Simone Raatz (0.77)	Manfred Weber (21.45)
4	Katja Mast (174)	Niels Annen (0.3174)	Terri Butler MP (135783.27)	Frank Junge (0.68)	Joachim Herrmann (15.6)
5	Heike Baehrens (149)	Mariano Rajoy Brey (0.3125)	Angela Merkel (123823.37)	Hubertus Heil (0.62)	Angela Merkel (12.58)

Table 7: Top politician nodes by centrality scores

5.4 Results and Discussion

In order to be able to draw better conclusions, it was decided to start the spread from the top 0.05% of the population in each graph. For the artists' graph, top 25 nodes according to each centrality algorithm were chosen to post the information. For the other two graphs, 0,05% of all nodes means 6 and 3 nodes, respectively.

The following command uses an APOC procedure, which starting from given nodes, expands to subgraph nodes that are reachable following the specified relationship with the specified direction with a maximum set distance. More specifically, the specific command below returns the number of distinct nodes that are related to the 25 nodes with the highest in-degree with direction towards them and are maximum 2 hops away, plus the 25 starting nodes, as the minimum level is set to 0.

```
MATCH (p:Artist)
WITH p ORDER BY p.likes DESC LIMIT 25
CALL apoc.path.subgraphNodes(p, {
   relationshipFilter: "<LIKES",
   minLevel: 0,
   maxLevel: 2
})
YIELD node
RETURN count(distinct node)
```

This procedure was run several times for each centrality measure with the maximum level starting from 1 and increasing by one in each step, until the number of nodes reached stopped increasing. The results collected for each graph are shown in the tables below.

Hops	Degree Centrality	Closeness Centrality	Betweenness Centrality	Eigenvector Centrality	PageRank
1	10805	7043	261	8180	7557
2	32911	29079	3891	31551	28946
3	38178	36563	12981	38749	36490
4	39152	37879	21039	40032	37851
5	39445	38169	25229	40335	38164
6	39565	38246	26372	40414	38246

7	39632	38263	26760	40438	38263
8	39679	38270	26955	40439	38270
9	39701	38270	27063	40439	38270
10	39723	38270	27112	40439	38270
11	39739	38270	27126	40439	38270
12	39743	38270	27130	40439	38270
13	39743	38270	27131	40439	38270

Table 8: Number of artist nodes reached per maximum number of hops for each centrality measure

As expected, the centrality measure with the best direct reach of nodes to spread information is the degree. In 12 hops the potential spread is maximized, and the information can reach up to 78.7% of the graph. Eigenvector Centrality achieves to have a slightly higher potential spread percentage (80%) which it achieves in 8 hops. However, the initial information transmission to the starting nodes' neighbors is significantly lower. Eigenvector starts prevailing over Degree in 3 hops' distance. The lowest performance in this case is observed by Betweenness Centrality which spreads information in significantly less nodes and achieves its highest spread in 13 hops. Closeness Centrality and PageRank have very similar performance, as they both need 8 hops for the maximum spread and the numbers of nodes reached in each hop are very close to each other. This is because the sets of 25 nodes each are very similar as they share the same 20 artists.

As seen in the chart below, Degree, Closeness, Eigenvector and PageRank centralities have similar rates of spread. Degree and Eigenvector are very close in their performances, with Degree Centrality transmitting directly the information to as many nodes as possible and Eigenvector achieving the highest possible spread.

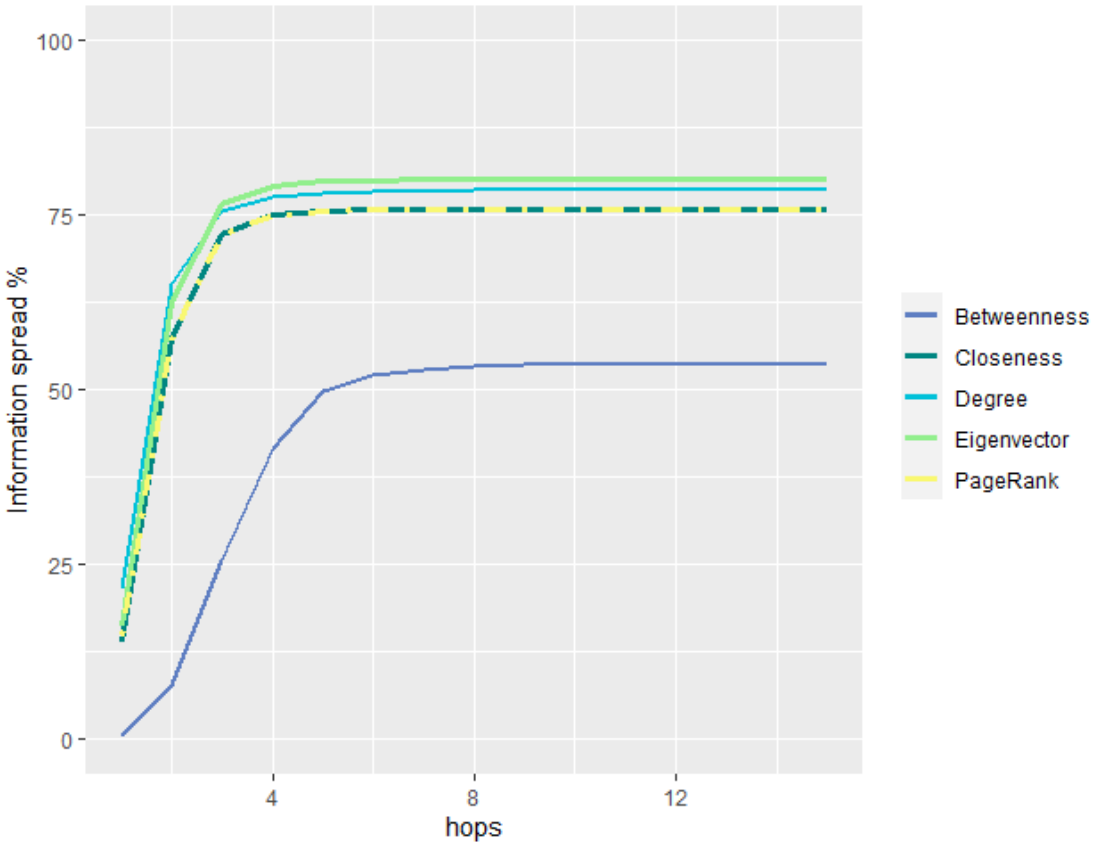

Figure 25: Centrality algorithms' performances in information spread for artists' graph

In Public Figures' graph, the spreads are very different for each centrality measure. When promoter nodes are selected by their high betweenness score, the performance is again not satisfying compared to the other centrality measures. PageRank seems to have a clear leverage as a centrality measure used to achieve high spread (up to 59%), although Degree and Closeness Centralities achieve to start with higher numbers of direct information transmission. By selecting promoters according to their PageRank score, more nodes can be reached from 3 hops distance and on than when selecting any other centrality measure. Regardless of the measure selected, spread in this network with this percentage of promoting nodes can be maximized in 8 hops.

Hops	**Degree Centrality**	**Closeness Centrality**	**Betweenness Centrality**	**Eigenvector Centrality**	**PageRank**
1	1055	792	87	483	617
2	3463	3038	524	1148	3247
3	4952	4651	1812	2502	5501

4	5546	5273	2875	4083	6479
5	5717	5447	3332	4844	6745
6	5746	5472	3510	5075	6805
7	5751	5480	3572	5132	6818
8	5752	5480	3585	5142	6822

Table 9: Number of public figure nodes reached per maximum number of hops for each centrality measure

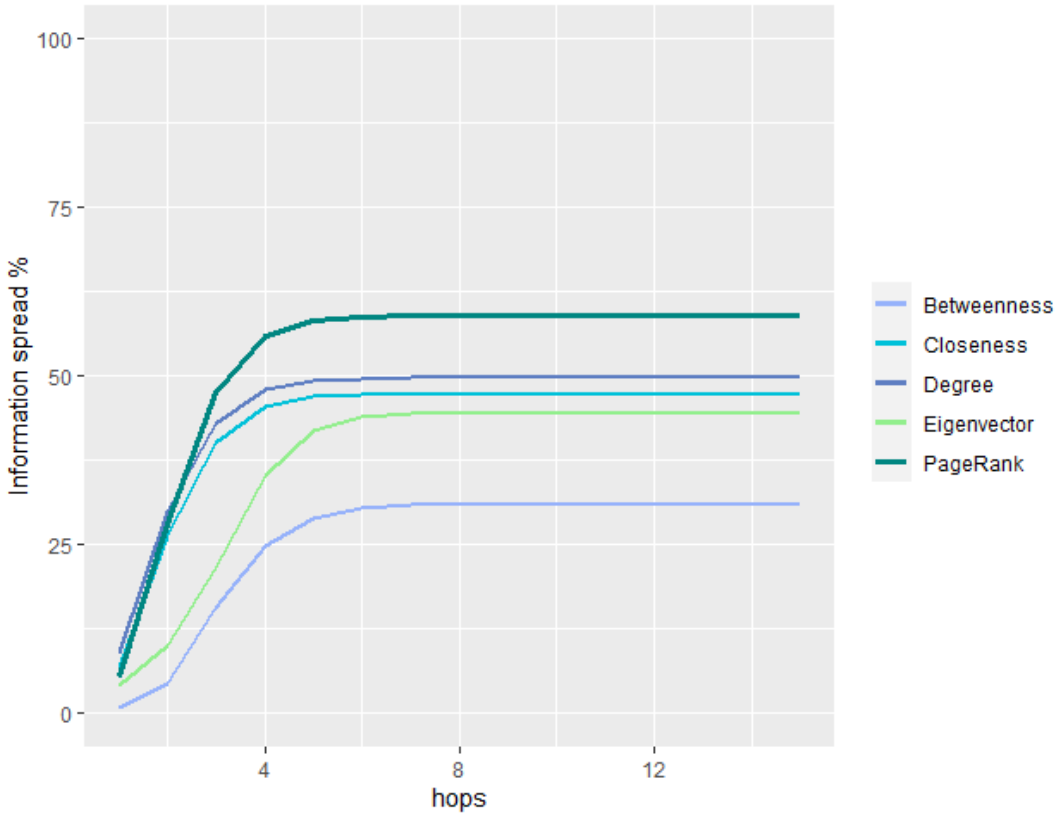

Figure 26: Centrality algorithms' performances in information spread for public figures' graph

For the last case of politicians' pages graph, Degree Centrality is the dominant centrality measure for starting nodes selection. With Degree Centrality, a maximum spread of 56.99% can be achieved with 9 hops. The nodes with the highest Betweenness Centrality scores can spread information to only up to 1.64% of the graph with four hops. Closeness centrality and PageRank follow Degree Centrality with slightly lower performances, both able to achieve up to 56.53% with 9 hops.

Hops	**Degree Centrality**	**Closeness Centrality**	**Betweenness Centrality**	**Eigenvector Centrality**	**PageRank**

1	623	360	5	192	412
2	1823	1566	85	494	1494
3	2802	2645	94	1150	2674
4	3162	3106	97	1824	3166
5	3318	3280	97	2205	3282
6	3352	3320	97	2346	3320
7	3360	3332	97	2411	3332
8	3366	3339	97	2439	3339
9	3367	3340	97	2440	3340
10	3367	3340	97	2441	3340

Table 10: Number of politician nodes reached per maximum number of hops for each centrality measure

As seen in the chart below, the centrality measures that worked better in this network are Degree, Closeness and PageRank. Their rates of spread seem very similar and their performances are very close. Betweenness Centrality is proved to be a bad choice for selecting the promoting nodes.

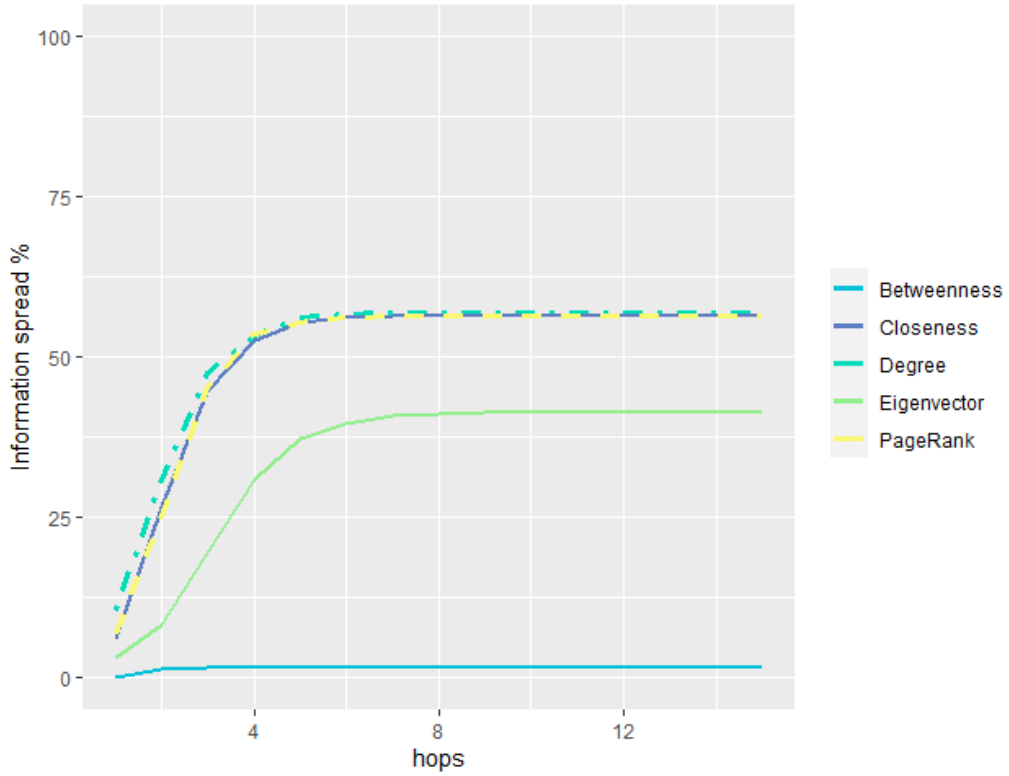

Figure 27: Centrality algorithms' performances in information spread for politicians' graph

The percentage of spread achieved in the first network is much higher than those of the other two, although the artists graph has the lowest density of the three. Also, its average clustering coefficient is 0.097, the lowest of the three.

Selection of promoters based on their Eigenvector score seems to work well for this network. In the first hop, the information has reached the 16,2% of the whole network. In the next hop, an impressive 62,5% have received it. This is already much higher than the final percentage of information spread in the other two networks.

The 25 nodes with the highest Eigenvector scores have all high in-degrees, with the lowest one having 148 likes, which is much higher than the average in-degree. Also, the 11.33% of the graph nodes have 0 likes, so if the information reaches them, they disrupt the spread.

In the next graph, the centrality measure with which the best promoter nodes were chosen was PageRank. In this graph, there is a strongly connected component of 2375 nodes and the average clustering coefficient is higher than that of the previous graph. The density is also slightly higher. However, the maximum spread that could be achieved was 59%.

The 6 nodes with the highest PageRank scores are not those with the highest in-degrees too, as in the previous graph. Three of them are also in the top 6 of degree centrality, but the remaining have relatively low in-degrees.

In the last graph, the politicians' Facebook pages, the centrality measure that achieved the best results was again the Degree Centrality. However, the percentage of the spread it achieved was the lowest comparing to the previous two. The graph is the densest of the three, and its average clustering coefficient, although low, it is also the highest.

Closeness and PageRank Centralities had similar performances to the Degree, achieving slightly lower spreads. However, Degree was clearly a better choice in this case, as it dominated at every hop reaching more nodes than the others. Betweenness Centrality was once again the poorest choice of centrality measure.

To sum up, all three graphs, regardless of their sizes, could achieve maximum spread within 8-9 hops. Degree Centrality, although being a simple count of nodes' incoming connections, was the centrality measure that had the most success in spreading information in more nodes, as it had the best results in two graphs and the second best, close to the first's results, in the other one. Degree is, by definition, the best in direct dissemination of information, but, surprisingly, it achieved to maximize the spread comparing to other "smarter" centrality measures. The amount of spreads was not predictable, as despite the graphs' similarities the spread percentages were quite different.

6 Conclusions and Future Work

The objective of this research was to evaluate the use of common existing centrality algorithms for selecting promoter nodes to spread information on social networks efficiently. The centrality algorithms that were tested are Degree, Closeness, Betweenness, Eigenvector, and PageRank. Each of the algorithms uses a different approach in scoring centrality, and for each one of them, node "importance" is defined differently.

The tests were run on three datasets containing Facebook pages linked among them with relationships of type "LIKES". The three graphs created are of different sizes, but all three of them were hard to visualize efficiently. Thus, in the phase of exploratory analysis, an attempt was made to understand the characteristics and structure of each graph. Apart from the graphs' sizes, the other characteristics that were calculated are the triangles count, the average clustering coefficient, the density, the minimum, maximum, and average degree. Also, regarding the in-degree of the nodes, which is the degree type that is important in the information spread problem, the minimum, maximum, and standard deviation values were calculated, along with the numbers of self-linking nodes and nodes with 0 in-degree. Finally, the Weakly Connected Components and Strongly Connected Components algorithms were used to identify clusters in the graphs. It appeared that there were no such groups, except for one large Strongly Connected Component in the second graph.

The small number of nodes with high in-degrees that are much higher than the average and the frequency histogram of node in-degrees indicate that the three graphs are scale-free, meaning that they follow the power-law distribution. This is quite common in social networks, as the popular nodes tend to attract many other nodes. Thus, if new nodes are added in the graph, there is a significant probability that they will connect with the popular nodes. This is the principle of preferential attachment, as it is commonly known in social networks.

The top nodes, according to each centrality measure, were often repeated in all three datasets. For example, a node with a high degree could also have high Eigenvector and PageRank centrality scores. This indicates that in social networks, the graph areas around popular nodes are often "crowded" resulting in other types of high centrality scores too.

The results of the information spreads did not allow for definite conclusions. There was not one single centrality algorithm that worked best on all three graphs, neither the spreads achieved were the expected.

Betweenness Centrality was the centrality algorithm with the worst performance on all three graphs. The spreads achieved in the three graphs were very low. This can be justified by the fact that betweenness centrality does not consider the relationships' direction. It is highly likely that nodes with high betweenness centrality score in a social network, are unpopular nodes that like more popular nodes. This assumption is most apparent in the third graph, where the top 3 nodes managed to reach only two other neighbor nodes, meaning that at least one of them had 0 in-degree. Nodes that are so unpopular are hard to be involved in popular circles; thus, they cannot spread information efficiently.

Closeness Centrality did not have the best results for any of the three graphs. However, its performance was close to the best in all three cases. Closeness Centrality does not consider the direction of relationships, either. For a node being close to many other nodes doesn't mean that it is popular in social graphs. However, it seems that in dense graph parts, there is a big probability that the information can spread easily, though this is not necessarily the case. The closeness centrality measure could have undesired results, as a node could have a high closeness score due to its high out-degree and the high connectedness of its neighbors even though its in-degree could be 0. Such a node would not be able to spread information at all.

Eigenvector centrality had, in general, a mediocre performance, although it achieved the best spread in the first graph. This is surprising, though, as PageRank, with its use of in-degrees, was expected to outperform Eigenvector in all cases. Besides the consideration of relationships' directions, the only other difference that could lead to these results is the damping factor. Eigenvector had a damping factor set to 1, by default, while for the PageRank, it was set to 0.85.

PageRank was the most suitable centrality measure for the second graph, and, generally, its performance was good and close to that of Closeness. As mentioned above, PageRank considers the direction of relationships, as well as the nodes' connectivity when scoring the nodes, thus its results were above average in every case. However, it was expected that it could achieve higher spreads and outperform the other centrality algorithms.

Degree Centrality, which is the simplest in concept and usually used as a baseline method, had the best performance overall. Although it is normal that nodes selected with this measure reached the most neighbors in one hop, the final total spread was also the highest or the second-highest in all three graphs.

Although there is not a clear answer to which centrality algorithm is more suitable to be used as a measure for selecting nodes for information spread, the results of the experiment raise important questions. Also, after studying the centrality algorithms and exploring the graphs' structure, there are conclusions drawn as to what needs attention in finding solutions for such problems. There are results in this experiment that led to new questions and need more investigation.

In the beginning of this study, there was the intent to visualize the graphs and draw conclusions about their structures through the visualizations. This was proved to be impossible in graphs of this scale, even for the smallest dataset. There is a technique that helps, where connected components are treated as single nodes, making it possible for the researcher to visually check the graph's structure. However, not all graphs have large connected components, and, more specifically, this was not the case in this study.

This obstacle made it difficult to explore further the connectivity of nodes that appeared to be important in most centralities. It would be interesting to discover more about these nodes that are important in many different ways and see if they provide more than other nodes in the final spread.

Another finding that needs more investigation is the case where Eigenvector centrality outperformed the PageRank algorithm unexpectedly, although the graph was directed. An assumption was made that it could be due to the different damping factors. More testing is required to score the nodes with different damping factors and check how the results change.

Also, regarding the good performance of Degree Centrality, more analysis is needed to see if its success is somehow connected with the social graphs' structures. It would be desired to calculate the probability of popular nodes being liked by other popular nodes. This could explain the high spreads achieved without having considered the transitivity.

Another aspect that needs to be investigated further is the maximum spread that could be achieved in each network. The specific graphs are all connected, but that doesn't consider the direction of

the relationships. The spread stops after some hops, and it would be interesting to investigate what happens in the ending nodes and which nodes were left unreached.

Much work has been left for the future due to lack of time. Besides the aforementioned further investigations, there are some ideas for improvement of the results.

It is clear that degree is very important in the selection of nodes. It is also clear that the direction of relationships must always be considered, as the important nodes for this problem are the popular ones. It could be interesting to select nodes combining both PageRank and Degree centralities. In this way, all three requirements would be met. The selection could be made by finding a set of the top *n* PageRank scored nodes and order them by their in-degrees to select those with the highest numbers of likes.

Another idea is to re-run the experiment with increased sets of promoter nodes and see when and if the spreads will keep stopping at some nodes while the starting nodes increase. As the goal in information spread problems is to maximize the nodes reached, it would be interesting to attempt to implement an algorithm that calculates the minimum number of nodes required as starting nodes to spread information across the whole graph.

7 References

Acemoglu, D., Ozdaglar, A., & ParandehGheibi, A. (2010). Spread of (mis)information in social networks. *Games and Economic Behavior*, 194-227.

Ahmed, R., Rossi, A., & K., N. (2015). The Network Data Repository with Interactive Graph Analytics and Visualization. *Twenty-Ninth AAAI Conference on Artificial Intelligence.*

Bonacich, P. (1987). Power and Centrality: A Family of Measures. *American journal of sociology*, 1170-1182.

Bonacich, P. (2007). Some unique properties of eigenvector centrality. *Social networks*, 555-564.

Bonér, J., Farley, D., Kuhn, R., & Thompson, M. (2014, September 16). *The Reactive Manifesto*. Retrieved from https://www.reactivemanifesto.org/

Brin, S., & Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*.

Doerr, B., Fouz, M., & Friedrich, T. (2012). Why rumors spread fast in Social Networks. *Communications of the ACM*, 70-75.

Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., & al., e. (2018). Cypher: An Evolving Query Language for Property Graphs. *Proceedings of the 2018 International Conference on Management of Data*, (pp. 1433-1445). Houston, United States.

Freeman, L. (1977). A set of measures of centrality based on betweenness. *Sociometry*, 35-41.

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry,* 35-41.

Freeman, L. C. (1978). Centrality in Social Networks: Conceptual Clarification. *Social Networks, 1(3),* 215-239.

Hambrick, M. E. (2012). Six degrees of information: Using social network analysis to explore the spread of information within sport social networks. *International Journal of Sport Communication*, 16-34.

Lerman, K., Ghosh, R., & Surachawala, T. (2012). Social Contagion: An Empirical Study of Information Spread.

Li, Y., Fan, J., Wang, Y., & Tan, K.-L. (2018). Influence Maximization on Social Graphs:. *IEEE Transactions on Knowledge and Data Engineering 30, no. 10*, 1852-1872.

Maharani, W., Adiwijaya, & Gozali, A. A. (2014). Degree Centrality and Eigenvector Centrality in Twitter., (pp. 1-5).

Marchiori, M., & Latora, V. (2000). Harmony in the small-world. *Physica A: Statistical Mechanics and its Applications*, 539-546.

Merrer, E. L., & Tredan, G. (2009). Centralities: Capturing the Fuzzy Notion. *In Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, (pp. 33-38).

Products. (n.d.). Retrieved from neo4j: https://neo4j.com/product/

Srinivas, A., & Velusamy, R. L. (2015). Identification of Influential Nodes from Social Networks based on Enhanced Degree Centrality. *IEEE International Advance Computing Conference (IACC)*, (pp. 1179-1184).

Wasserman, S., & Faust, K. (1994). *Social network analysis: Methods and applications.* Cambridge University press.