International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 5 Issue 4, May-June 2021 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Graph Databases and Graph Data Science in Neo4j

Akanksha Junawane[1], **Y. L. Puranik**[2]

[1]PG Student, Department of MCA (Engineering), [2]Guide
[1,2]P.E.S.'s Modern College of Engineering, Pune, Maharashtra, India

ABSTRACT
The contents include what graph databases are, their uses, notations, structure, what is neo4j, its components, what are Graph Data Science and GDS algorithms and their types in Neo4j. It contains an overview of all the features provided by neo4j like querying, visualization, remote access, etc. It will also include information about Neo4j Aura, Sandbox, Desktop, Browser and Bloom. The various tiers of maturity of GDS algorithms and their types will also be explained along with an example of each of the type of algorithms.

How to cite this paper: Akanksha Junawane | Y. L. Puranik "Graph Databases and Graph Data Science in Neo4j" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-5 | Issue-4, June 2021, pp.771-777, URL: www.ijtsrd.com/papers/ijtsrd42358.pdf

IJTSRD42358

GRAPH DATABASES

A graph database is a type of database that uses graphical structures with nodes, edges, and properties to represent the data and for its storage with the help of queries. Graph databases are built with the purpose of storing and navigating the graph through its relationships. Relationships are the most important aspect in graph databases, and maximum value of graph databases with respect to querying, analysing and visualizing is derived from these relationships present between the nodes. In graph databases nodes are used to store data entities and edges store the relationships between entities. An edge always has a start node, end node, type, and direction. An edge can describe hierarchical or inheritance relationships, actions performed, ownership type and so on. There are unlimited ways and limitless numbers of relationships a node can have.

A graph in graph databases can be traversed along specific relationship types or across the entire graph. In graph databases, traversing the joins or relationships is very fast because the relationships between nodes are not calculated at query times but are persisted in the database. Graph databases have advantages for use cases such as social networking, recommendation engines, and fraud detection, when you need to create relationships between data and quickly query these relationships.

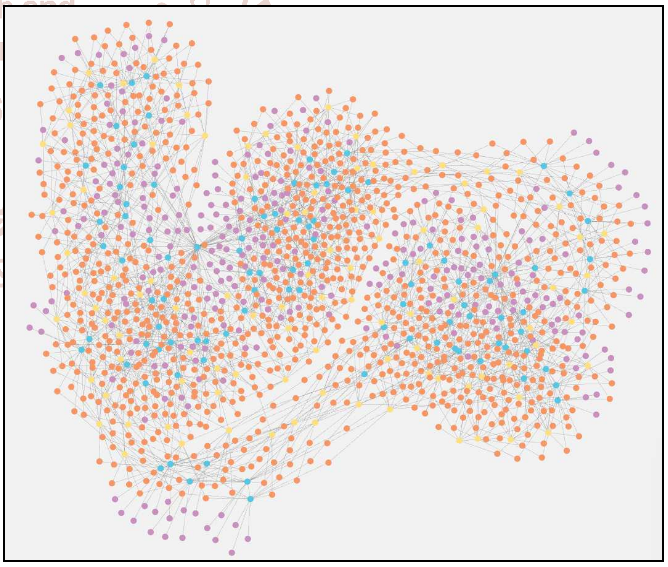


Fig.1

Figure 1depicts how a completely loaded graph database looks along with its nodes connected with edges or relationships. This graph will be used in this article as an example to show all the applications and functionalities of Neo4j, GDS and the algorithms. A main component of the system is the graph(that is relationships or edges). The graph relates the data items in the store to a collection of nodes and edges, the edges representing the relationships between the nodes. The relationships allow data in the store to be linked together directly and, in many cases, retrieved with one operation. Graph databases hold the relationships between data as a priority. Querying relationships is fast because they are permanently stored in the database.

Nodes:
They represent entities such as people, organizations, credit cards, etc. to be tracked. They are roughly the equivalent of a tuple or row in a relational database, or a document in a document-store database.

Edges:
They are also called as relationships. They connect nodes to one another and represent the relationships between them. Examining the connections and interconnections of nodes, properties and edges gives rise to meaningful patterns. The edges can either be directed or undirected.

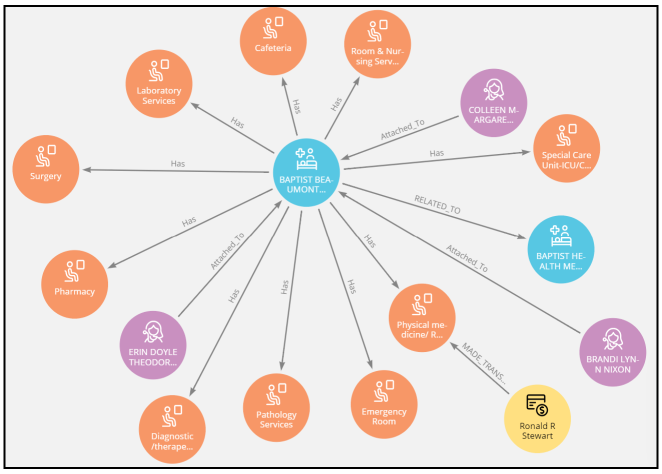
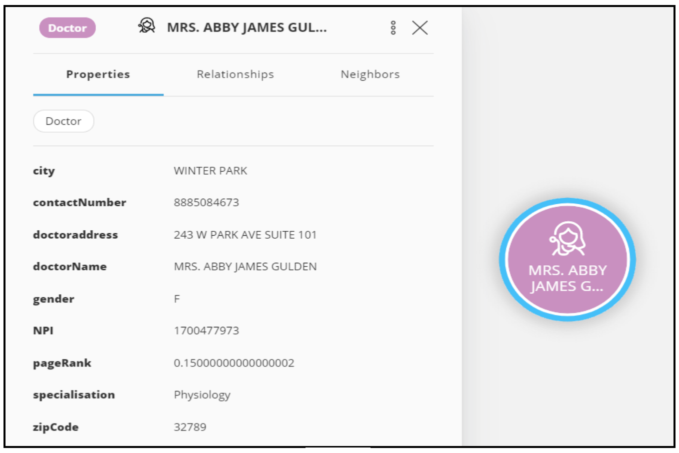


Fig. 2

In figure number 2, the nodes are of types doctor, hospital, register and credit card and the relationships are 'has' between the hospital and register nodes , 'made_transaction' between register and credit card nodes, 'related_to' between two hospital nodes and 'attached_to' between doctor and hospital nodes which are represented by the edges.

Properties:
It is metadata associated to a node or property. For example here for the Doctor node the properties are city, contactNumber, doctorName, doctorAddress, Gender, etc. and the property values are winter park, 8885084673, Mrs. Abby James Gulden, 243 west park ave suite 101, F and so on.



city	WINTER PARK
contactNumber	8885084673
doctoraddress	243 W PARK AVE SUITE 101
doctorName	MRS. ABBY JAMES GULDEN
gender	F
NPI	1700477973
pageRank	0.15000000000000002
specialisation	Physiology
zipCode	32789

Fig. 3

When should you use a graph database:
1. When you have highly related data.
2. You need a flexible schema.
3. You want to have a structure and build queries that are more similar to way people think.

NEO4J
Neo4j (Network Exploration and Optimization 4 Java) is a graph database management system developed by Neo4j, Inc. Neo4j is implemented in Java and accessible from software written in other languages using the Cypher query language through a transactional HTTP endpoint, or through the binary "bolt" protocol.

Neo4j comes in 2 editions: Community and Enterprise. It is dual-licensed: GPL v3 and a commercial license. The Enterprise Edition is available under a closed-source Commercial license.

Notable features of Neo4j are:
1. Data model with flexible schema
2. Scalable and reliable
3. Based on ACID properties
4. Provides built in browser web application.

SERVICES PROVIDED BY NEO4J:
1. Neo4j Aura:
Neo4j Aura is an automated graph database service. Aura enables you to focus on creating rich, data-driven applications rather than waste time managing the databases. Neo4j Aura is fast, reliable, scalable and makes the power of relationships available in a cloud-native environment, enabling fast queries for real-time analytics and insights.

2. Neo4j Sandbox:
Neo4j Sandbox is a great way to try out Neo4j for free without having to download or install anything on your machine. It gives you access to Neo4j database, Neo4j Bloom, and plugins like Neo4j Graph Data Science, all hosted online and private to you.

3. Neo4j Desktop:
Neo4j Desktop is a Developer IDE or Management Environment for Neo4j instances. You can manage as many projects and database servers locally as you like and also connect to remote Neo4j servers. Neo4j Desktop comes with a free Developer License of Neo4j Enterprise Edition.

4. Neo4j Graph Applications:
A Graph app is a Single Page Application (SPA) built with HTML and JavaScript which interact with Neo4j databases through Neo4j Desktop. A developer could take an existing SPA and package it into a graph app or start from scratch with a new idea.

5. Neo4j Browser:
Neo4j Browser is a tool for developers to interact with the graph. It is the default interface for both Enterprise and Community Editions of the Neo4j database. You can query your databases using cypher queries from the neo4j browser.

6. Neo4j Bloom:
Neo4j Bloom helps the developer visualize the graph and its nodes and relationships better which in turn helps understand it for further use. It also provides features for selective visualization of data and lets you create dynamic queries to help you visualize the database based on the dynamic input provided by the user.

GRAPH DATA SCIENCE
Graph Data Science is a science-driven approach to gain knowledge from the relationships and structures in data, typically to power predictions. It contains various techniques that help data scientists answer questions and explain outcomes using graph data. Graph data science lets you leverage the power of relationships, that is, the connections

between your data points to improve model prediction and answer previously obstinate questions.

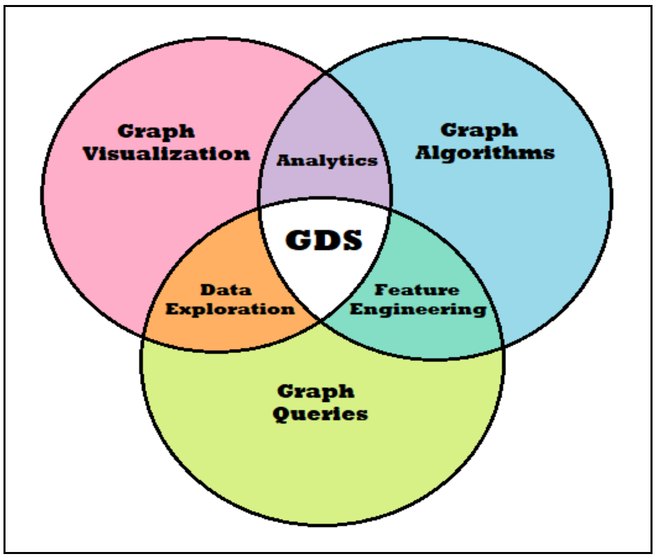


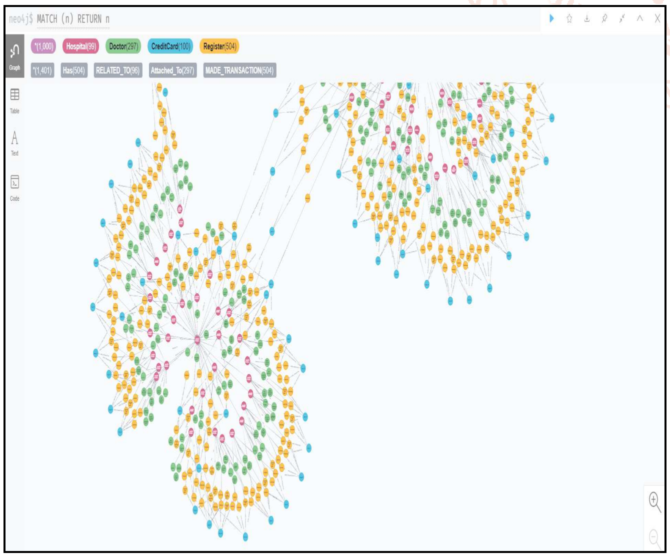
Fig. 4

GDS consists of three main parts, namely graph visualization, graph algorithms and graph queries.

Graph visualization helps in understanding the nodes and relationships and the overall structure and patterns of the graph better and using graph algorithms along with this, data analysts run analytics on the graph for predictive purposes. Graph queries are used to create the nodes and relationships in a graph and manipulate them as per the developer's requirement and these together with the Graph algorithms are used for feature engineering in graph data science. Graph algorithms are used to run various types of analysis on data and these along with graph visualization aid in data exploration.

1. Graph Queries:

In neo4j cypher queries are used to query the database. This querying of the database is done through the neo4j browser.



```
neo4j$ MATCH (n) RETURN n
```

Fig. 5

Through this neo4j browser you can query your database and view the results in the tabs below. Here we are taking a simple 'MATCH (n) RETURN n' query as an example which returns all the nodes and relationships in the graph. The GDS algorithms are also run on the graph database through the browser. Using the cypher queries you can create new nodes, relationships and properties and also modify and update them as and when necessary. Complex queries can be used for more detailed and use case specific results.

2. Data Visualization:

In neo4j the Bloom perspective is used for the visualization of data.

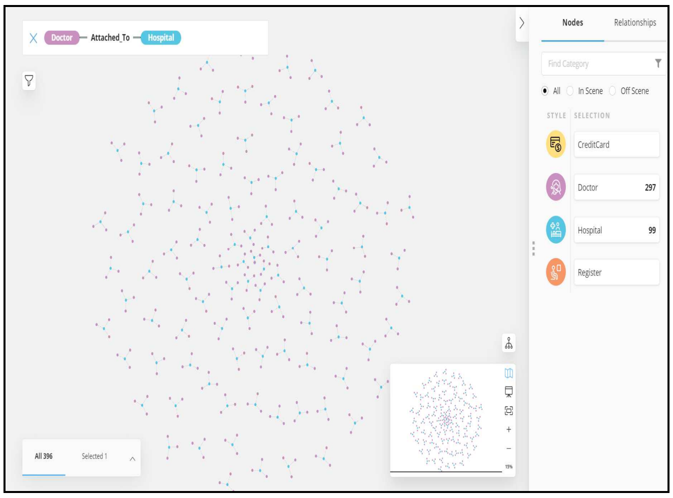
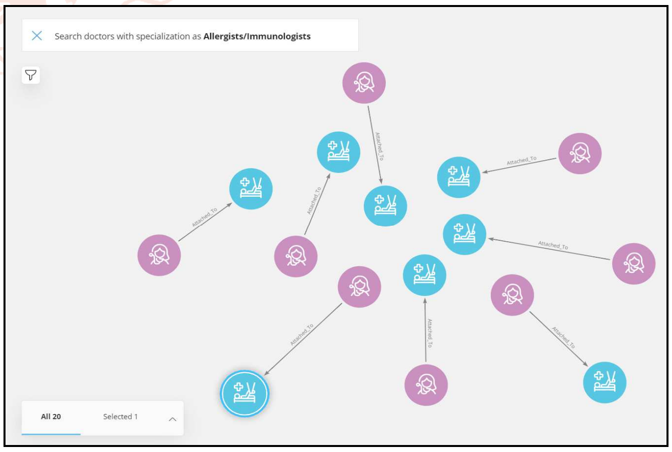


Fig. 6

Bloom helps us visualize data in a very easy and attractive way. It helps the user understand the database better. As we can see here the pane on the right shows the type of nodes and relationships. Here, we have also added appropriate icons to the nodes for better understanding and to be able to identify the type of node in the graph. The search box above shows suggestions as you type in the search phrase which makes it very user friendly. Here, we have selected the doctor and hospital nodes with an 'attached_to' relation between them. This is how we can selectively visualize certain aspects of the graph as per requirement. We can also create dynamic search phrases so that the output changes according to user input and we don't have to write the queries over and over again.

For example let's see a dynamic query that returns the doctors with specialization as Allergists/Immunologist.



Search doctors with specialization as **Allergists/Immunologists**

Fig. 7

Here, we can see that with the use of dynamic search phrase we have given an input of our choice as the specialization and it has returned all the doctors with specialization as Allergists/Immunologists along with the hospitals that they are attached to.

3. GDS Algorithms:

Neo4j offers various types of graph data science algorithms are made available in the Graph Data Science library. These algorithms provide unsupervised machine learning methods which help you understand the structure of your graph and also help in deterministic and predictive analysis. The algorithms in neo4j exist in 3 tiers of maturity,

➢ **Alpha tier:** The algorithms in the alpha tier are completely experimental in nature and can be removed at any point in time.

The prefix 'gds.alpha' is used before the name of the algorithms in this tier during implementation.

➢ **Beta tier:** The algorithms in the beta tier can be candidates for the production quality tier. That is these algorithms can be moved to the production quality tier in the future if requires and deemed fit.

➢ The prefix 'gds.beta' is used before the algorithm names in this tier during implementation.

➢ **Production-quality tier:** If an algorithm exists in this tier, it indicated that, that specific algorithm has been thoroughly tested.

The algorithms present in this layer are stable and scalable in nature.

The prefix 'gds' is used before the algorithm names in this tier during implementation.

Types of algorithms:
We will be using the Hospital Database shown in figure 1 as an example to demonstrate the application and results of the algorithms. Following are the types of algorithms provides by the GDS library in neo4j,

1. Centrality Algorithms:
These algorithms are used to find out which nodes are important in a graph based on its topology. The position of nodes in the graph determines the influence of each of the node over the database. These algorithms help determine group dynamics such as the ripple effects caused by the vulnerability of nodes, credibility of nodes and the bridges between nodes. Following are the centrality algorithms provided by neo4j,

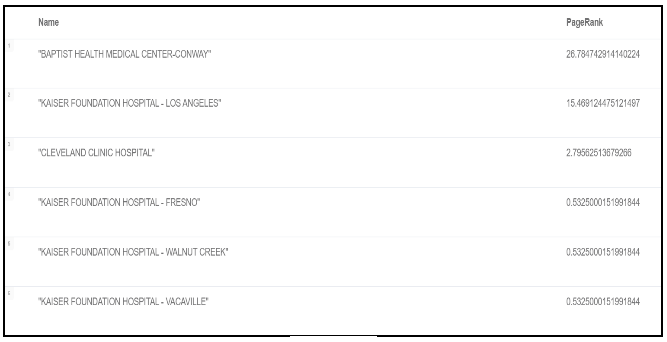
➢ Alpha:
- HITS
- Closeness Centrality
- Article Rank
- Harmonic Centrality
- Degree Centrality
- Eigenvector Centrality

➢ Production-quality:
- Page Rank
- Betweenness Centrality

Let's see the application of one of the algorithms to understand the functionality:

Page Rank Algorithm: This algorithm measures the importance of each of the nodes within the graph based on the number of incoming relationships and the importance of the corresponding nodes. It means that a node is only as important as the nodes connected to it.

Eg. The figure 8.1 shows the output after running the page rank algorithm on the Hospital database. It return the page ranks of all nodes in the graph in a descending order.



	Name	PageRank
1	"BAPTIST HEALTH MEDICAL CENTER-CONWAY"	26.784742914140224
2	"KAISER FOUNDATION HOSPITAL - LOS ANGELES"	15.469124475121497
3	"CLEVELAND CLINIC HOSPITAL"	2.79562513679266
4	"KAISER FOUNDATION HOSPITAL - FRESNO"	0.5325000151991844
5	"KAISER FOUNDATION HOSPITAL - WALNUT CREEK"	0.5325000151991844
6	"KAISER FOUNDATION HOSPITAL - VACAVILLE"	0.5325000151991844

Fig. 8.1

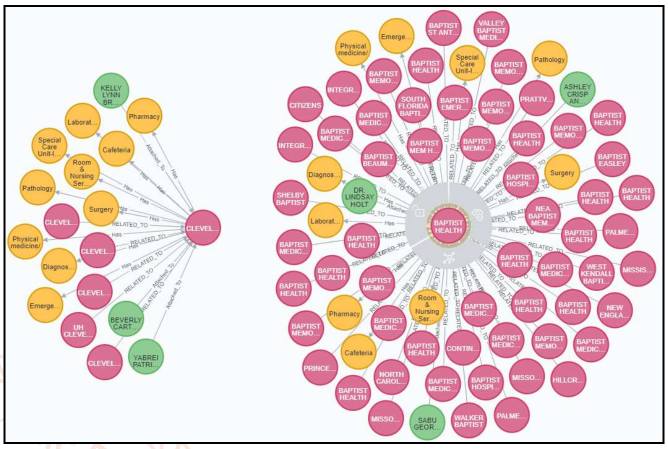


Fig. 8.2

Here we take Baptist Health Medical Center and Cleveland Clinic Hospital to see how the algorithm works. As we can see here the Baptist Health Medical Center has a huge amount of incoming nodes and hence has page rank of 26.7 and Cleveland Clinic has comparatively very few incoming nodes and hence its page rank is 2.7.

2. Similarity Algorithms:
Similarity algorithms compare the nodes with each other to see how alike they are based on their neighbours or properties. They produce likeness scores which can be used for developing categorical hierarchies or to personalize recommendations. Following are the similarity algorithms provided by neo4j,

➢ Alpha:
- Jaccard Similarity
- Cosine Similarity
- Approximate Nearest Neighbors
- Euclidean Similarity
- Overlap Similarity
- Pearson Similarity

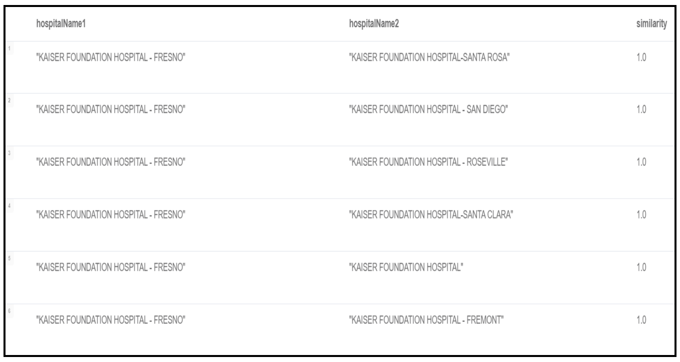
➢ Beta:
- K-Nearest Neighbors

➢ Production-quality:
- Node Similarity

Let's see the application of one of the algorithms to understand the functionality:

Node Similarity Algorithm: This algorithm measures the similarity of a set nodes based on their neighbours. If 2 nodes have more shared nodes their similarity score will be higher. This algorithm uses Jaccard similarity to compute the similarity scores.

The figure 9.1 shows the output after running the node similarity algorithm on the Hospital database.



	hospitalName1	hospitalName2	similarity
1	"KAISER FOUNDATION HOSPITAL - FRESNO"	"KAISER FOUNDATION HOSPITAL-SANTA ROSA"	1.0
2	"KAISER FOUNDATION HOSPITAL - FRESNO"	"KAISER FOUNDATION HOSPITAL - SAN DIEGO"	1.0
3	"KAISER FOUNDATION HOSPITAL - FRESNO"	"KAISER FOUNDATION HOSPITAL - ROSEVILLE"	1.0
4	"KAISER FOUNDATION HOSPITAL - FRESNO"	"KAISER FOUNDATION HOSPITAL-SANTA CLARA"	1.0
5	"KAISER FOUNDATION HOSPITAL - FRESNO"	"KAISER FOUNDATION HOSPITAL"	1.0
6	"KAISER FOUNDATION HOSPITAL - FRESNO"	"KAISER FOUNDATION HOSPITAL - FREMONT"	1.0

Fig. 9.1

Here we can see that each node is being compared to every other node in the graph and similarity scored is being computed.

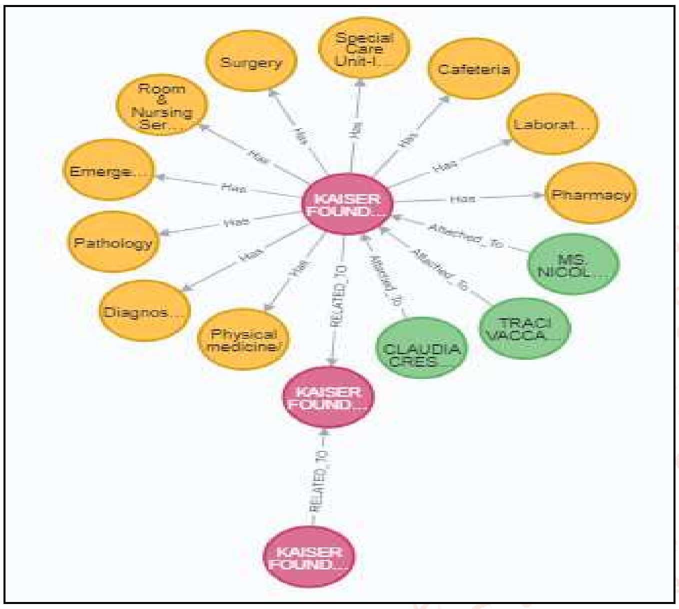


Fig. 9.2

Here we can see that since 'Kaiser Foundation Hospital - Fresno' and 'Kaiser Foundation Hospital-Santa Rosa' have one shared neighbour their similarity score is 1.

3. Pathfinding Algorithms:

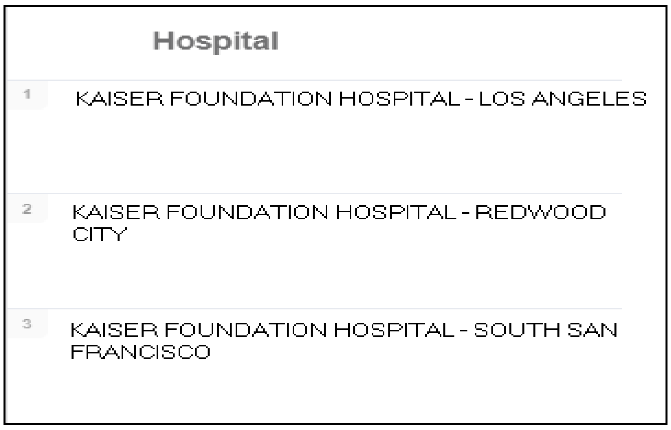
These algorithms are used for graph analytics and find the most efficient or shortest paths to traverse between nodes. Some of these algorithms are used to understand complex dependencies and evaluate routes for uses such as physical logistics and IP routing.

➢ Alpha:
- Breadth First Search
- Depth First Search
- Minimum Weight Spanning Tree
- All Pairs Shortest Path
- Single Source Shortest Path
- Random Walk

➢ Beta:
- A*
- Yen's algorithm
- Dijkstra Source-Target
- Dijkstra's Single-Source

Let's see the application of one of the algorithms to understand the functionality:

Random Walk: This algorithm provides random paths in a graph between the nodes. A random walk means that we specify a start node, randomly choose a neighbour to reach or choose based on a provided probability distribution, and then do the same from that node, keeping the resulting path in a list.



	Hospital
1	KAISER FOUNDATION HOSPITAL - LOS ANGELES
2	KAISER FOUNDATION HOSPITAL - REDWOOD CITY
3	KAISER FOUNDATION HOSPITAL - SOUTH SAN FRANCISCO

Fig. 10.1

Eg. In figure 10.1 we can see the result of random walk algorithm when run on the hospital database while considering the start node as 'Kaiser Foundation Hospital-Los Angeles' and destination as 'Kaiser Foundation Hospital-South San Francisco'.

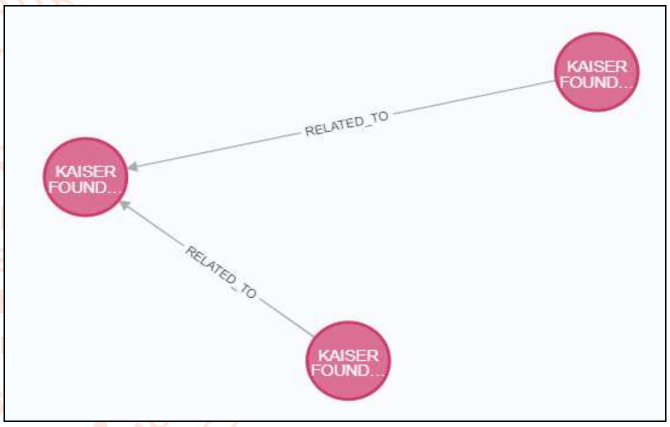


Fig. 10.2

4. Link Prediction Algorithms:

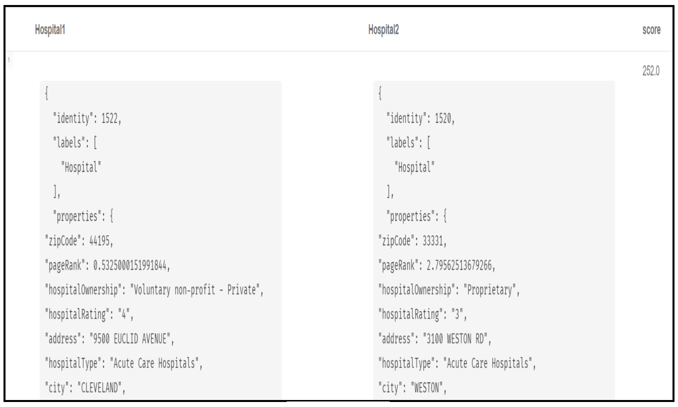
Link prediction algorithms are used to predict the likelihood of forming relations between nodes that are not connected at present. It considers the closeness of nodes in the graph and also structural elements of the graph. The following algorithms are provided by neo4j for this type,

➢ Alpha
- Preferential Attachment
- Total Neighbors
- Adamic Adar
- Common Neighbors
- Resource Allocation
- Same Community

Let's see the application of one of the algorithms to understand the functionality:

Preferential Attachment: Preferential Attachment is a measure used to calculate the closeness of nodes, based on the number of shared neighbours they possess.

Eg. If we run this algorithm on our Hospital Database considering two hospital, namely 'Cleveland Clinic' and 'Cleveland Clinic Hospital' we get the following result as shown in figure 11.1.



Hospital1	Hospital2	score
{ "identity": 1522, "labels": ["Hospital"], "properties": { "zipCode": 44195, "pageRank": 0.5325000151991844, "hospitalOwnership": "Voluntary non-profit - Private", "hospitalRating": "4", "address": "9500 EUCLID AVENUE", "hospitalType": "Acute Care Hospitals", "city": "CLEVELAND",	{ "identity": 1520, "labels": ["Hospital"], "properties": { "zipCode": 33331, "pageRank": 2.79562513679266, "hospitalOwnership": "Proprietary", "hospitalRating": "3", "address": "3100 WESTON RD", "hospitalType": "Acute Care Hospitals", "city": "WESTON",	252.0

Fig. 11.1

As we can see here the algorithm has returned the information of the two hospitals along with its preferential attachment score. As we can see the hospitals are directly connected and are direct neighbours. Therefore their score is 252.0 which is a very high closeness score.

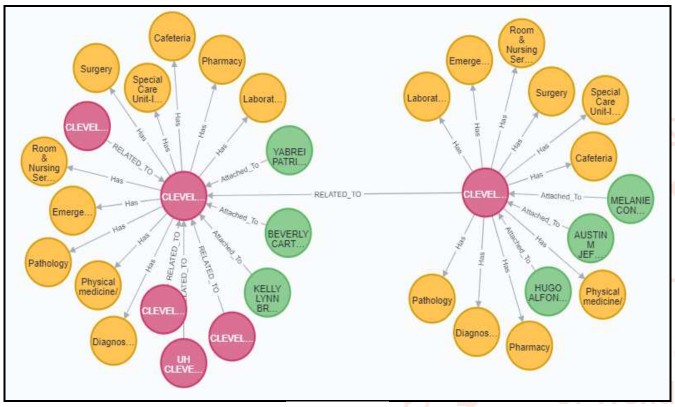


Fig. 11.2

5. Community Detection Algorithms:

These algorithms find clusters in the graph based on relationships to find the communities where members have more significant interactions. Detecting communities help us in predicting similar behaviours between the nodes, find duplicate entities or prepare data for various analytical purposes.

➢ Alpha:
- Strongly Connected Components
- Speaker-Listener Label Propagation

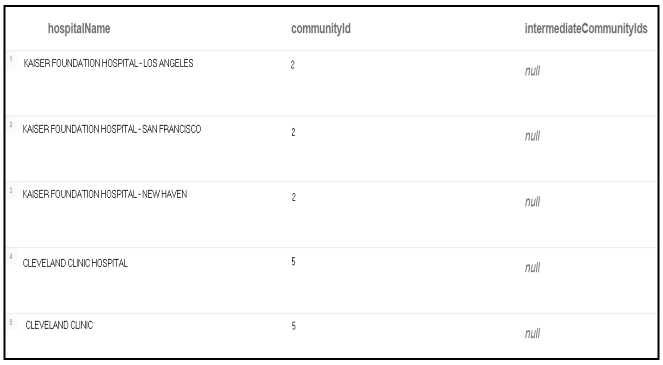
➢ Beta:
- Modularity Optimization
- K-1 Coloring

➢ Production-quality:
- Louvain
- Triangle Count
- Local Clustering Coefficient
- Label Propagation
- Weakly Connected Components

Let's see the application of one of the algorithms to understand the functionality:

Louvain: The Louvain method is used to detect communities in large networks. It maximizes a modularity score for each community based on the quality of assignment of nodes to various communities. It is a hierarchical clustering algorithm and it recursively merges communities into a single node and executes the modularity clustering on the condensed graphs.

Eg. When we run this algorithm on our graph after creating communities based on certain property and relationship type, we get the following results shown in figure 12.1 which includes the name of the hospital and the community id of the node.



	hospitalName	communityId	intermediateCommunityIds
1	KAISER FOUNDATION HOSPITAL - LOS ANGELES	2	null
2	KAISER FOUNDATION HOSPITAL - SAN FRANCISCO	2	null
3	KAISER FOUNDATION HOSPITAL - NEW HAVEN	2	null
4	CLEVELAND CLINIC HOSPITAL	5	null
5	CLEVELAND CLINIC	5	null

Fig. 12.1

6. Node Embedding Algorithms:

Graph embeddings are very powerful as even while reducing the dimensionality for being able to decode the graph, they preserve all the key features. They transform the structure and features of the graph into fixed length vectors which uniquely represent each node. In simpler terms they compute low-dimensional vector representations of nodes in the graphin order to be useful in machine learning processes by capturing the complexity and structure of the graph and transforming it.

➢ Alpha
- Node2Vec

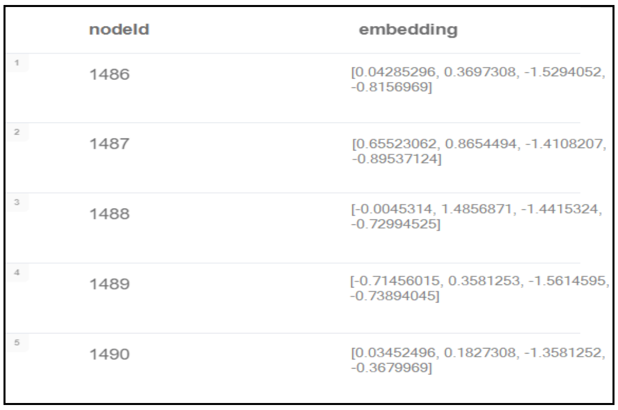
➢ Beta
- GraphSAGE

➢ Production-quality
- FastRP

Let's see the application of one of the algorithms to understand the functionality:

FastRP: FastRP stands for Fast Random Projection. It is a node embedding algorithm which belongs to the family of random projection algorithms. These algorithms are theoretically backed by the Johnsson-Lindenstrauss lemma according to which one can project *n* vectors of *arbitrary* dimension into *O(log(n))* dimensions and still approximately preserve pair wise distances among the points.

Eg. When we run the FastRP algorithm on the Hospital graph we get the following results shown in figure 13.1 which returns the node ids and embeddings of the nodes in the graph based on properties and orientation.



	nodeId	embedding
1	1486	[0.04285296, 0.3697308, -1.5294052, -0.8156969]
2	1487	[0.65523062, 0.8654494, -1.4108207, -0.89537124]
3	1488	[-0.0045314, 1.4856871, -1.4415324, -0.72994525]
4	1489	[-0.71456015, 0.3581253, -1.5614595, -0.73894045]
5	1490	[0.03452496, 0.1827308, -1.3581252, -0.3679969]

Fig. 13.1

CONCLUSION:

The main purpose of this paper is to help you understand how graph databases and gds works in neo4j and also how the graphs are visualized and queried. The functionality of the algorithms types is also explained in simple terms along with an example algorithm of each type to see how the results are generated along with the graphical representation.

REFERENCES:

Books:

[1] Graph Algorithms - Mark Needham

[2] Learning Neo4j - Jérôme Baton, Rik Van Bruggen

[3] Graph Data Science - Amy Hodler and Mark Needham

Links:

[1] https://neo4j.com/docs/graph-data-science

[2] https://neo4j.com/blog/why-graph-databases-are-the-future