San José State University

A Protein to Protein Interaction Database in Neo4j: Generation and Features

Aarohi Chopra and Daniel Quintana

CS 257 – Database Systems Principles

Ramin Moazeni, PhD

Abstract - This research project developed a Neo4j-based graph database system to analyze complex protein-to-protein interactions, addressing the challenges of managing and interpreting extensive proteomic data. The database system is designed for ease of use, enabling users to effectively store, query, and analyze large datasets, with a focus on understanding biological interactions within these datasets. By efficiently processing protein dataset annotations into a network of nodes, edges, labels, and weights, the system enhances the analysis capabilities in proteomics. This advancement is particularly significant given the complexity of proteins. The use of a graph database approach, exemplified by Neo4j, offers a dynamic and interconnected framework for representing and analyzing biological data, marking a pivotal step forward in proteomic research.

Index Terms - Neo4j, Protein-to-Protein Interactions, Graph Databases, Centrality Analysis

PROJECT GOALS AND MOTIVATION

The aim of the project is to create a database system that is specifically designed to analyze complex protein datasets to understand the interactions between the data points. The database system should not only be able to store and manage large databases but also identify the biological interactions represented by the dataset. Additionally the database needs to be user friendly and easy to use for researchers in fields of proteomic, clinical research, biology, etc so enabling them to easily query, analyze and gather meaningful insight from the data. It should be able to understand annotations provided in protein datasets and archives to create nodes, edges, labels, and weights based on the provided data. This tool will help in advancement in the field of proteomics by providing a strong resource with high compute, important data point analysis features, etc. With the emergence of applications of graph databases over different domain applications in proteomics will be an interesting domain to explore.

Proteins are long sequences of amino acid blocks translated from RNA transcript. They contain genetic code required for all necessary biological processes. Single strand of proteins representing the primary structure is combined to form Alpha helix or B pleated sheets which form the secondary structure of proteins. These secondary structures interact with different proteins to form stable bonds which result in tertiary structure of proteins which are also bound to changes due to post translational mutations that occur in protein.

When amino acids combine to form proteins, they create unique sequences that fold into specific three-dimensional structures. These structures, in turn, determine the function of the proteins. The progression from a linear sequence of amino acids to a complex, functioning protein is a leap in complexity. For instance, simple sequences can form structures like alpha helices or beta sheets, which are the secondary structures in proteins. These secondary structures further interact and fold into more complex tertiary structures, often stabilized by various types of bonds and interactions. The vast diversity in protein structures arises from these folding patterns and the sequence of amino acids.

Moreover, post-translational modifications, which occur after protein synthesis, add another layer of complexity. These modifications can alter the protein's function, activity, location, and interactions with other molecules, thereby increasing the diversity of the protein functions and interactions.

Given this complexity, traditional methods of data analysis and storage struggle to efficiently manage and interpret the vast array of protein data. This is where graph databases like Neo4j come into play, offering a more dynamic and interconnected way to represent and analyze these complex biological systems. By enabling researchers to create nodes, edges, and labels that represent various aspects of proteins and their interactions, such databases facilitate a deeper and more comprehensive understanding of the proteome.

In summary, the leap from a limited number of amino acids to a seemingly endless diversity of protein structures and functions underscores the need for advanced computational tools like Neo4j. These tools enable researchers to tackle the intricacies of proteomics, turning a vast and complex data landscape into a navigable and informative resource.

ADDRESSED DATABASE SYSTEM CHALLENGE

In the realm of protein-to-protein interactions, traditional database systems often fall short due to the inherent complexity and voluminous nature of proteomic data. These conventional systems typically rely on extensive join operations, which become increasingly cumbersome as data scales up, leading to significant performance bottlenecks. Our approach addresses these challenges by adopting Neo4j, a graph database system that excels in managing complex, interconnected data. Unlike traditional relational databases, Neo4j represents protein interactions as direct relationships within a graph structure. This method facilitates a more intuitive and efficient handling of the dense network of protein interactions, eliminating the need for complex joins and significantly reducing system overhead. By leveraging the graph-based model, our database mirrors the natural interconnectivity of proteomic data, enhancing both the performance and the utility of the system for comprehensive proteomic analysis.

RELATED WORK

The foundational work for our project involved straightforward guidelines for importing protein interactions into Neo4j, although this process was hindered by numerous errors and misspellings, requiring significant troubleshooting. This primary reference, a study by Kumar and Mukhtar titled "Building Protein-Protein Interaction Graph Database Using Neo4j," is crucial in understanding the practical aspects of building a Protein-Protein Interaction (PPI) graph database using Neo4j. This approach in protein-related research is relatively new yet highly promising, as detailed in their work[1].

The utility and effectiveness of graph databases in the domain of protein research are further highlighted by Santos et al. in "A knowledge graph to interpret clinical proteomics data." This study illustrates the groundbreaking application of knowledge graphs in interpreting complex clinical proteomics data, demonstrating the significant advantages that graph databases offer in managing and understanding intricate protein interactions[2].

Furthermore, Lehne and Schlitt's extensive comparison of six major PPI databases in their work, “Protein-protein interaction databases: Keeping up with growing interactomes,” focuses on content and annotation. They shed light on the challenges associated with expanding protein databases, the integration of diverse data sources, and the overarching need for comprehensive data collection. This research underscores the importance of effective data management and curation in PPI databases, crucial for advancing the field of proteomics[3].

Our project is also informed by Farooq et al.’s research, which delves into the methods, databases, and applications of PPIs in virus-host studies. Their work emphasizes the efficacy of graph databases in delineating complex networks of protein interactions, offering valuable insights for our database development[4].

Additionally, the insights from BioGRID's themed curation projects, which focus on assembling core genes and proteins for specific biological processes, have enriched our understanding of annotations and metadata in protein databases. This approach is essential for precise and relevant data curation in PPI

research, ensuring our database's relevance and usability for specific research areas[5].

Moreover, Armingol et al.'s study on deciphering cell-cell interactions from gene expression opens new avenues for understanding PPIs within a broader biological context. Their integration of spatial mapping and transcriptomics to unravel complex cellular interactions provides a new perspective for our database's functional scope[6].

A pivotal study by Lee highlights the transformative impact of deep learning in PPI analysis from 2021 to 2023. This comprehensive review of recent methodologies in PPI analysis serves as an invaluable resource, guiding our database's approach to incorporating advanced computational techniques[7].

Additionally, a study utilizing the Galaxy platform for genome-wide predictions and structural modeling of PPIs by Guerler et al. underscores the importance of accurate PPI identification. Their insights into cellular processes and novel drug targets are highly relevant to our project, influencing our methodological approach[8].

These collective studies have significantly contributed to our approach in developing a Neo4j-based PPI database. They underscore the importance of not just gathering data, but curating, visualizing, and understanding it in a way that truly benefits the scientific community's ongoing research in proteomics.

METHODOLOGY, ANALYSIS, AND IMPLEMENTATION

The objective was to procure protein data, inclusive of Gene Ontology annotations, from specialized databases available online. Additionally, we sought to incorporate a dataset outlining the interactions between proteins, identifiable through their unique protein IDs. This data was integrated into the Neo4j graph database, whereupon we employed Cypher queries to extract various graph features—such as centrality metrics, community detection, and shortest paths. The goal was to map out the interactions (edges) based on similarity and to apply vector embeddings to capture the multi-dimensional structure of the protein interaction network.

I. Database Key Design Goals

The objective of utilizing a graph database was to generally simplify and visualize the analysis of protein to protein interactions. To achieve this we focused on creating a system that has the following qualities.

- Easy Query and Information Retrieval: Construct a user-friendly interface for researchers to efficiently query and extract data.
- Visualization of Protein Interactions: Develop visualization tools within the database to map and understand complex protein interactions.
- Advanced Analysis Features: Implement advanced graph analysis features to explore the intricate network of protein interactions.

II. Key Components and Algorithms

Our methodology integrates a suite of computational algorithms and components essential for the analysis of protein interaction networks. These include various centrality metrics to assess the influence of individual proteins, community detection algorithms for uncovering modular structures, shortest path calculations for understanding connectivity, and similarity measures for evaluating resemblances between proteins. We also leverage graph embedding techniques to distill the complex network topology into a form amenable to machine learning applications. This enumeration is not exhaustive but highlights the principal tools and techniques employed in our analysis.

- Betweenness Centrality: A centrality measure that quantifies the number of times a node acts as a bridge along the shortest path between two other nodes. In the context of protein interactions, it helps identify key proteins that may regulate or impact various biological pathways due to their strategic position within the network.
- Degree Centrality: A basic yet powerful metric that counts the number of direct connections a node has. Proteins with high degree centrality are often critical, as they interact with many other proteins, suggesting a pivotal role in cellular functions.
- PageRank: An algorithm initially designed to rank web pages, which assesses the importance of nodes in a network based on the quantity and quality of their links. When applied to protein networks, it can reveal proteins that are central to the network's integrity and function.

- Louvain Method: A community detection algorithm that groups nodes into modules or communities by optimizing the modularity score of the network. For protein interaction databases, this method is useful for identifying potential functional groupings or complexes of proteins that work together.
- Node2Vec: An algorithm for learning continuous feature representations for nodes in networks, which effectively captures the network's topology. In proteomics, Node2Vec can uncover patterns in protein interactions, aiding in the classification and prediction of protein functions.
- Fast Random Projection: A dimensionality reduction technique that projects high-dimensional data into a lower-dimensional space. It simplifies the protein interaction data to accelerate computational tasks while still maintaining the essential structure of the network for analysis.
- k1 Coloring: A graph coloring method where adjacent nodes are assigned different colors to minimize the number of colors used. This technique could be employed to visually differentiate proteins in the interaction network, potentially correlating with different functional types or groups.
- Strongly Connected Components: Refers to the subgraphs where there is a directed path from each node to every other node within the component. Identifying such components in protein interaction networks could highlight proteins involved in strong interaction loops, possibly indicating feedback mechanisms or essential signaling pathways.
- Triangle Count: An algorithm that counts the number of triangles in a network, which is a small motif consisting of three interlinked nodes. In protein interaction networks, a high triangle count can be indicative of tightly interconnected protein complexes, suggesting structural stability or functional significance.
- Semantic Similarity: Measures the similarity of nodes based on the attributes and annotations associated with them, often derived from ontologies. This is particularly valuable in proteomics for comparing proteins based on their Gene Ontology terms, inferring functional similarities or relationships.
- Local Clustering Coefficient: A measure of the likelihood that nodes in a network will cluster together. In the realm of protein-to-protein interactions, a high local clustering coefficient can indicate proteins that are part of a dense network, such as a molecular complex or a biochemical pathway.

III. Architecture.

This segment of the "Architecture" section maps out the pathway of data as it transitions from initial raw formats into a dynamic, query-able model within the Neo4j graph database environment. It outlines the transformation of protein metadata and interaction data through a sequence of precise steps: data preprocessing using Python to refine and structure the information, the importing of this processed data into Neo4j, and the execution of Cypher queries to weave a rich tapestry of protein interactions. This narrative will guide you through the essential stages and tools that form the backbone of our protein interaction database, detailing the systematic conversion of data into a functional and insightful repository.

1. Protein Metadata and Protein Interaction Data: These are the initial data sources. The metadata includes descriptive information about proteins, while the interaction data details how proteins interact with one another.
2. Python Data Preprocessing Script: This script is responsible for cleaning and formatting the raw data into a structured format. It reads the gene association data from a given source, often a CSV or Excel file, and processes it to extract relevant information such as gene identifiers and associated Gene Ontology (GO) terms.
3. Protein & Gene Ontology CSV and Protein Interactions CSV: The output from the preprocessing script. The Protein & Gene Ontology CSV contains processed data ready for import into Neo4j, including the association between proteins and their functions or characteristics as defined by GO terms. The Protein Interactions CSV file outlines the relationships between proteins, which will be represented as edges in the Neo4j graph database.
4. Neo4j Database Import Scripts: These scripts import the preprocessed CSV files into Neo4j,

transforming the tabular data into a graph structure. They create nodes for proteins, add labels and properties like GO annotations, and establish relationships based on the interaction data.

5. Cypher Queries: Once the data is imported, Cypher, Neo4j's query language, is used to manipulate and retrieve information from the database. Queries can range from simple look-ups to complex pattern matching and are used to extract features or insights from the protein interaction network.
6. Protein Interaction Neo4j Graph Database with Features from Cypher Queries: This is the fully formed graph database that incorporates both the protein data and the interaction data, enriched with features derived from the execution of Cypher queries.
7. Data Analysis Python Script or Machine Learning: The final step in the workflow involves analyzing the graph database to derive biological insights or predictive models. This can be done through a Python script for statistical analysis or machine learning algorithms that can predict interactions, classify protein functions, or uncover new patterns within the data.

At its core, the "Protein Interaction Neo4j Graph Database" acts as a centralized hub for analyzing and storing protein interactions.

From the central database, several calculated attributes are derived using Cypher queries:

- Protein Centrality Measures: These are metrics calculated to determine the importance of a protein within the network, such as how central a protein is in the context of all interactions.
- Protein Similarity Scores: These scores are calculated to quantify the similarity between proteins based on their attributes or interaction patterns.
- Protein Shortest Interaction Paths: This calculation finds the shortest paths of interaction between proteins, which is crucial for understanding the functional linkage and communication within the protein network.
- Protein Communities: This feature identifies clusters or groups of proteins that interact more frequently with each other than with other proteins in the network, suggesting a common function or involvement in the same biological process.

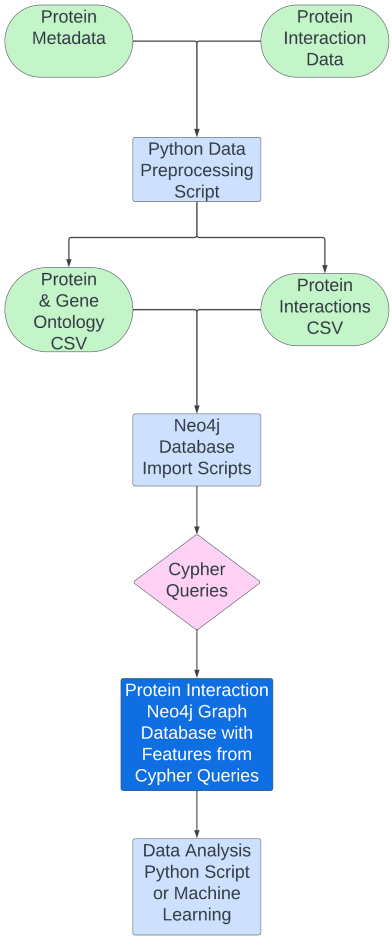

FIGURE 1. UML Diagram of Data Pipeline

- Protein Graph Embeddings: These are complex representations that encapsulate the position of a protein within the network, preserving the high-dimensional structure in a form that can be used for machine learning tasks.

The graph database is not only a repository but also a processing unit that can organize, sort, and export data based on these calculated attributes. For instance, it can export subsets of the graph where proteins have high centrality measures or are part of the same community. This capability allows researchers to conduct targeted analyses, like examining key regulatory proteins or exploring the protein-protein interactions within specific biological pathways. It facilitates a more nuanced understanding of the proteome by enabling the sorting and categorization of proteins according to various interaction metrics derived from the graph data.

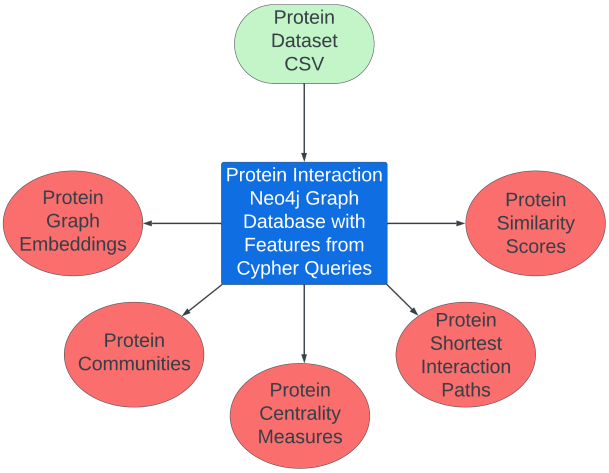

FIGURE 2. UML Diagram of GraphDB Features

In the architectural context of user interaction within the Neo4j graph database system, the UML diagram and accompanying user definitions delineate a flow where users initiate requests that may be routed through administrative processing before interacting with the database. This structure ensures a clear delineation of roles, enhancing the system's efficiency and data governance.

The diagram simplifies the user roles and actions into a clear, concise flow of tasks and responsibilities:

- Users are at the forefront, engaging with the system by providing initial data, activating the graph generation process, and conducting searches for specific protein features. This reflects a user-centric design, allowing for direct interaction with the database through a graphical user interface.
- User Admins operate behind the scenes, handling the critical tasks of loading data into the graph database, overseeing the generation of the database structure, and crafting Cypher queries to facilitate data retrieval and manipulation. They also have the capability to request additional data from the graph database to ensure completeness and accuracy of the information.
- The Graph Database acts as the repository and responds to the User Admin's requests by providing the requisite data. This component is essential for maintaining the integrity and extensibility of the database system, ensuring that the user's and admin's needs are met efficiently.

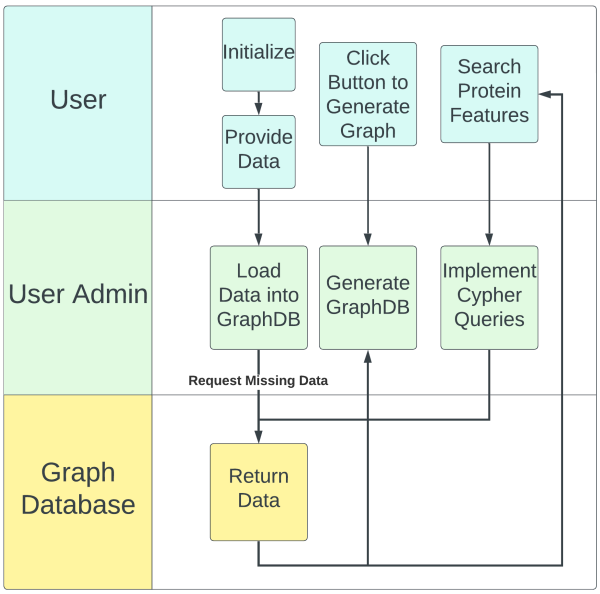

FIGURE 3. UML Diagram of User Cases

SYSTEM DEMO

I. Preprocessing Demo

The preprocessing phase begins with the extraction of gene association data, crucial for understanding the functional relationships within the protein structures. Here, we employ Python to parse and structure this data into a usable format.

```
# Load the gene association data as a pandas dataframe. Make a dictionary of AGI (Arabidopsis Genome Initiative, unique identifier), gene names, and GO terms using the following script.

data = pd.read_csv("https://www.arabidopsis.org/download_files/GO_and_PO_Annotations/Gene_Ontology_Annotations/gene_association.tair.gz", sep="\t", header=None, comment="!")

gene_association=data[[1,2,4]].values.tolist()

AGI_name=dict()
```

```
AGI_GO=defaultdict(list)
for i in gene_association:
  AGI, Name, GO = i
  AGI_name[AGI] = Name
  AGI_GO[AGI].append(GO)
```

The above script reads the gene ontology annotations and stores them in a structured dictionary, setting the stage for further analysis.

Then we address the interactions between proteins:

```
# Load interaction data as a pandas
dataframe. Keep only "main_screen"
interactions.
Network =
pd.read_excel("http://interactome.dfci
.harvard.edu/A_thaliana/doc/AI_interac
tions.xls")
Network = Network[Network.main_screen
== 1]
Network = Network[["ida", 'idb']]
# Extract unique proteins from both
'ida' and 'idb' columns.
unique_proteins =
set(Network['ida'].tolist() +
Network['idb'].tolist())
```

This segment filters the interaction data, ensuring only primary interactions are considered for the network analysis, reflecting a focus on high-confidence protein interactions.

After that, we proceed to consolidate our findings into a CSV format:

```
# Write the protein info to the
PROTEIN_neo.csv file.
fh = open("PROTEIN_neo.csv", "w")
print(":ID|name:STRING|GO:STRING[]",
file=fh)
for Protein in unique_proteins:
    ID = Protein
    name = "Unknown"
    GO_Terms = "Unknown"
    if Protein in AGI_name:
        name = AGI_name[Protein]
        if Protein in AGI_GO:
            GO_Terms =
",".join(AGI_GO[Protein])
    print(ID, name, GO_Terms, sep="|",
file=fh)
fh.close()
```

This code snippet translates the protein data into a Neo4j-friendly CSV, laying the groundwork for graph database integration.

The Protein_neo.csv is visualized here:

:ID|name:STRING|GO:STRING[]					
AT4G24540|AGL24|GO:0000977	GO:0000978	GO:0000981	GO:0005634	GO:0006357	GO:0045944
AT2G02470|AL6|GO:0000976	GO:0003712	GO:0005634	GO:0005634	GO:0005634	GO:0005739
AT5G66460|MAN7|GO:0005576	GO:0016985	GO:0071704	GO:0009845	GO:0016985	GO:0071944
AT3G07210|AT3G07210|GO:0003674	GO:0005634	GO:0009059	GO:0055085	GO:0071702	GO:0071705

TABLE 1 .Protein Gene Ontology CSV Example

Next, we document the protein-protein interactions:

```
# Write the
AI1main_PPI_Interactions_neo.csv File.
fh =
open("AI1main_PPI_Interactions_neo.csv
", "w")
print(":START_ID|:END_ID", file=fh)
for a, b in Network.values.tolist():
    print(a, b, sep="|", file=fh)
fh.close()
```

The final output captures the interaction pairs, ready for Neo4j ingestion, providing a network blueprint.

The AI1main_PPI_Interactions_neo.csv is visualized here:

:START_ID|:END_ID	
AT1G05410|AT3G10140	
AT3G54850|AT5G19010	
AT3G07780|AT5G66720	
AT1G80040|AT5G66720	
AT1G09660|AT2G38610	
AT1G15385|AT4G24840	

TABLE 2. Protein Interaction CSV Example

These steps collectively demonstrate the data transformation process, vital for the subsequent database demonstration and analysis.

II. Database Demo

For the purposes of this demonstration, the initial setup and data importation into the Neo4j graph database—though intricate and sometimes challenging due to potential bugs and troubleshooting needs—will not be the focus. These preliminary steps, which entail establishing a Neo4j database environment and importing the requisite datasets, are foundational yet relatively direct. The demonstration will instead concentrate on the subsequent stages of our database application, particularly emphasizing the execution of Cypher queries and the augmentation of the graph database with calculated attributes, to showcase the database's capabilities in facilitating protein interaction analysis.

The setup phase for the Local Clustering Coefficient (LCC) Algorithm within the Neo4j environment is outlined below. It begins with configuration parameters that dictate the scope and scale of the analysis:

```
:param limit => (1000);
:param config => ({});
:param communityNodeLimit => ( 10);
:param graphConfig => ({
  nodeProjection: 'Protein',
  relationshipProjection: {
    relType: {
      type: 'INTERACTS',
      orientation: 'UNDIRECTED',
      properties: {}
    }
  }
});
:param generatedName =>
('in-memory-graph-1702113391628');
```

These parameters define operational constraints, such as the maximum number of nodes to display (limit) and the targeted size of communities within the graph (communityNodeLimit). The graph configuration sets the stage for the projection of proteins as nodes and their interactions as edges in an undirected graph. The run in Neo4j will look like this:

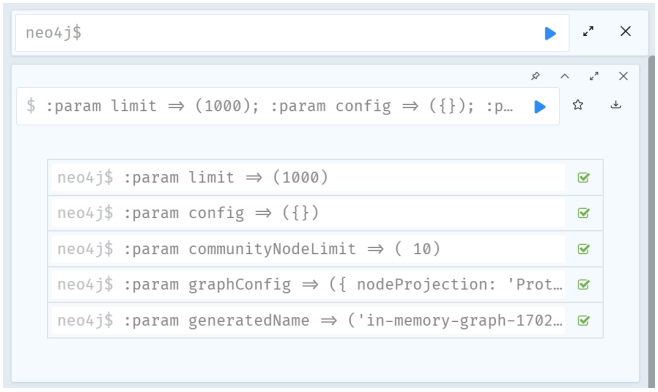

FIGURE 4. Neo4j Cypher Query Execution

Here we will then call using all of the parameters that we have just set up

```
CALL gds.graph.project($generatedName,
$graphConfig.nodeProjection,
$graphConfig.relationshipProjection,
{});
```

This operation structures the data into a graph format that the GDS (Graph Data Science) library can interact with, enabling the execution of the LCC algorithm.

The execution of the LCC algorithm is performed with:

```
CALL
gds.localClusteringCoefficient.stream(
$generatedName, $config)
```

```
YIELD nodeId,
localClusteringCoefficient AS
coefficient
WITH gds.util.asNode(nodeId) AS node,
coefficient
RETURN node, coefficient
ORDER BY coefficient DESC
LIMIT toInteger($limit);
```

This code snippet calculates the LCC for each node, which quantifies how closely a node and its neighbors are to being a complete graph, or clique. It is a measure of the degree to which nodes tend to cluster together.

The final step involves cleaning up the in-memory graph to conserve resources:

```
CALL gds.graph.drop($generatedName);
```

This command removes the temporary graph from memory after the analysis is complete. The output from the LCC algorithm provides insights into the local structure of the graph, which can be visualized or further analyzed to understand the intricacies of protein interactions.

Below is the visualization of the LCC algorithm with a 100 node limit due to rendering constraints.

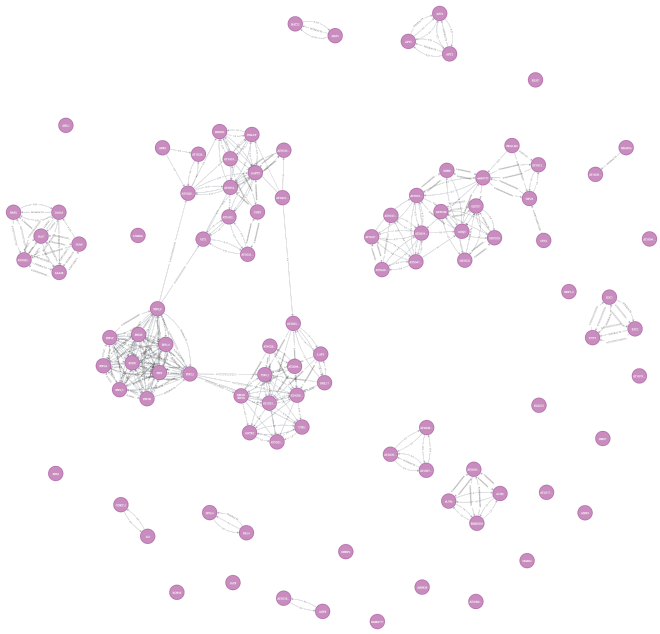

FIGURE 5.Visualization of LCC with 100 Nodes

EVALUATION METHODOLOGY AND SIGNIFICANT RESULTS

This section evaluates the methodology used in the Neo4j graph database system for analyzing protein interactions and outlines the significant results derived from our analysis. It encompasses an appraisal of the chosen approaches, the algorithms' effectiveness, and their computational complexities.

I. Methodology Evaluation

Our methodology encompassed the creation and utilization of a Neo4j graph database, complemented by Python scripting for preprocessing and data analysis. This approach was strategically chosen to effectively manage and interpret complex protein interaction data. Our evaluation focused on assessing the performance of the database across various metrics such as speed, accuracy, and scalability, which are crucial in handling diverse and large protein interaction datasets. We tested the system with different sizes of datasets and complexities of queries to ensure robustness and efficiency.

II. Analysis of Algorithms

In our database system, we incorporated several algorithms, including centrality measures, community detection, and clustering techniques, each pivotal in graph data analysis. Our evaluation concentrated on the accuracy of these algorithms in mirroring the complexities of protein interactions, ensuring they reflect the biological relationships with high fidelity. We also conducted a complexity analysis to understand the computational demands of these algorithms, particularly focusing on time and resource efficiency. This aspect of the evaluation was critical in confirming that our algorithms were not only theoretically sound but also practically implementable in a real-world context.

III. Significant Results

The use of the database yielded insightful results, particularly in identifying key proteins within the network. The PageRank and betweenness centrality measures highlighted proteins with significant roles in the network, suggesting their potential importance in biological processes. The visualizations created using histograms and scatter plots provided a clear view of the distribution and correlations among centrality

measures, offering valuable insights into the protein network's structure.

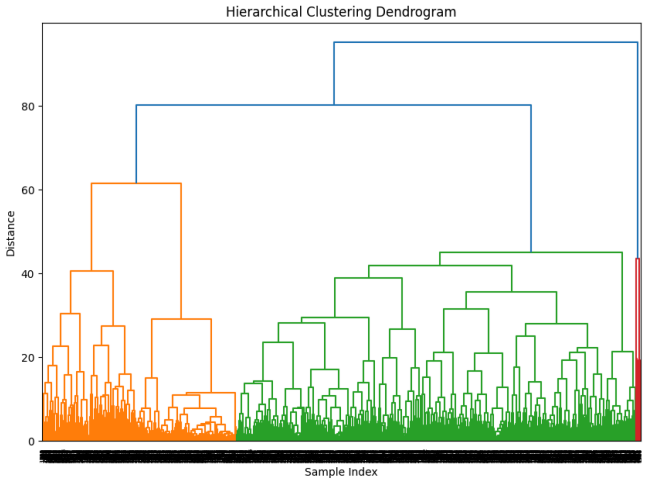

FIGURE 6. Hierarchical Clustering of Proteins

IV. Discussion of Results

These findings contribute significantly to our understanding of protein interaction networks, shedding light on key proteins and the network's underlying structure. The cluster analysis, revealing distinct protein groupings, suggests potential areas for further biological research and hypothesis formation. The pattern analysis and network property relationships opened new avenues for exploring protein functions and interactions, making this approach a potent tool for proteomics research

FIGURE 7. Protein Clusters Betweenness and Degree

This evaluation demonstrates the effectiveness of the Neo4j graph database in handling and analyzing complex biological data, making it an invaluable tool for researchers in the field of proteomics. The combination of accurate algorithms and intuitive visualizations ensures that the database is not only a repository of information but also a platform for discovery and analysis.

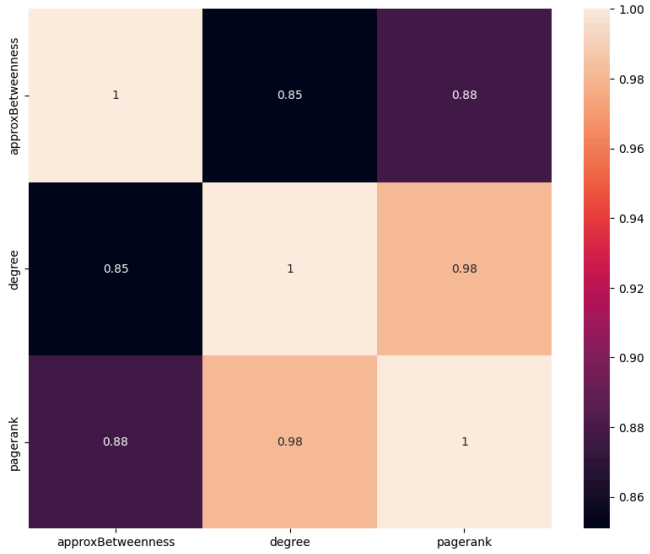

FIGURE 8. Heatmap Analysis of Protein Interactions Data Subset

CONCLUSION, LESSONS LEARNED, AND POSSIBLE IMPROVEMENTS

I. Conclusions

The project to develop a Neo4j-based database for analyzing protein interactions has been a significant step forward in the field of proteomics. Our system successfully integrated complex protein data, enabling the analysis of protein structures and interactions through advanced graph database technology. Particularly effective was the database's ability to calculate similarity measures and centrality metrics, which provided valuable insights into the protein interaction networks. These analyses proved instrumental in understanding the intricate relationships and functions of various proteins.

II. Lessons Learned

A key learning point was the realization that while graph databases like Neo4j are powerful tools for managing complex biological data, they also have limitations, especially when handling certain computational tasks. For example, we encountered challenges when attempting to calculate K-Nearest Neighbors (KNN) within the database. This highlighted the necessity of understanding the capabilities and limitations of graph databases in processing extensive and complex datasets.

III. Possible Improvements

Looking ahead, there are several areas where improvements can be made to enhance the database's functionality and user experience. First, developing consistent, executable scripts for converting processed protein data into the Neo4j graph database format would streamline the data import process. This would ensure a more efficient and error-free data transfer. Additionally, creating scripts for Cypher commands used in the analysis would be beneficial. During our project, Cypher commands were often written and executed ad hoc, leading to a loss of valuable queries since they weren't systematically recorded. By saving these commands, future users can replicate analyses, verify results, and build upon our work more effectively.

In conclusion, while the database successfully addressed many challenges in protein data analysis, there is room for improvement in computational efficiency and user workflow management. The lessons learned from this project will guide future developments in this field, contributing to more advanced and user-friendly tools for proteomic research.

REFERENCES

[1] Kumar N, Mukhtar S. Building Protein-Protein Interaction Graph Database Using Neo4j. Methods Mol Biol. 2023;2690:469-479. doi: 10.1007/978-1-0716-3327-4_36. PMID: 37450167.

[2] Santos, A., Colaço, A.R., Nielsen, A.B. et al. A knowledge graph to interpret clinical proteomics data. Nat Biotechnol 40, 692–702 (2022). https://doi.org/10.1038/s41587-021-01145-6

[3] B. Lehne and T. Schlitt, "Protein-protein interaction databases: Keeping up with growing interactomes - human genomics," BioMed Central, https://humgenomics.biomedcentral.com/articles/10.1186/1479-7364-3-3-291

[4] Farooq QUA, Shaukat Z, Aiman S, Li CH. Protein-protein interactions: Methods, databases, and applications in virus-host study. World J Virol. 2021 Nov 25;10(6):288-300. doi: 10.5501/wjv.v10.i6.288. PMID: 34909403; PMCID: PMC8641042.

[5] Oughtred, R., Rust, J., Chang, C., *et al.* (2021). The BioGRID database: A comprehensive biomedical resource of curated protein, genetic, and chemical interactions. *Protein science : a publication of the Protein Society*, *30*(1), 187–200. https://doi.org/10.1002/pro.3978

[6] Armingol, E., Officer, A., Harismendy, O. *et al.* Deciphering cell–cell interactions and communication from gene expression. *Nat Rev Genet* 22, 71–88 (2021). https://doi.org/10.1038/s41576-020-00292-x

[7] Lee M. Recent Advances in Deep Learning for Protein-Protein Interaction Analysis: A Comprehensive Review. Molecules. 2023 Jul 2;28(13):5169. doi: 10.3390/molecules28135169. PMID: 37446831; PMCID: PMC10343845.

[8] Guerler, A., Baker, D., van den Beek, M. *et al.* Fast and accurate genome-wide predictions and structural modeling of protein–protein interactions using Galaxy. *BMC Bioinformatics* 24, 263 (2023). https://doi.org/10.1186/s12859-023-05389-8