# GAD Graph

Devyanshi Chandra, Riya Gurnani, Kirti Magam, Sreevatsa Nukala, Ben Tunney

Northeastern University, Boston, MA, USA

Github Repository

## Abstract

We created a gene-disease association graph to visualize the data stored in DisGeNet. We pulled the data using SQL to store in a neo4j graph database. We wanted to create a novel way to view this data in order to better understand and interpret the association data. This can further the gene-disease study by making it more easy to analyze through our user interactive program in Python that calls the Cypher queries and returns NetworkX visualizations to answer the user's questions.

## Introduction

Our motivation for this project was to visualize the gene-disease association data from DisGeNet in a more novel way that is more user-friendly and interactive. We felt that it was difficult to truly understand the data in its current form of tables and decided that creating graph visualizations would make it much easier to analyze. Displaying the associations in this way is significant in understanding how varying gene types may impact the occurrence of certain diseases. It would also aid potential solutions and interventions for those who struggle with the diseases within this dataset. Creating efficient data pipelines and complex, intricate graphs would ideally work towards making the information and analyses we've completed more accessible to those impacted by the diseases, as well as those who would like to potentially learn more about its impact on their communities. This is made easier through our user interactive program to choose which queries they would like to execute.

## Methods

The first step in our project was acquiring our data. We intended to take gene-disease association data from DiGeNet in order to convert into a graph database format. We first acquired a csv of genes and their qualities and also a list of diseases and their qualities present in their database. In order to maintain a level of quality to our data, we filtered the list of genes to include only those that had at least fifty publications about them. Our next step was to make calls to the DisGeNet API [1], so that we could receive all of the gene-disease associations based on the genes we sent in. For each association between a gene and a disease, their database provides additional values such as their own calculated score and evidence index level. They calculate a score ranging from zero to one based on the types of sources of data that show the association and the number of publications that show a positive association [2]. Their evidence index is used to determine the overall consensus on an association by calculating the number of publications supporting an association divided by the total number of publications reporting about the association in general. When making our API calls, we filtered our results to only receive data that had a minimum score of 0.55 and a minimum evidence index of 0.75. It was essential to set some thresholds on the data placed into the graph database so that we ensured a relatively important association between a gene and a disease that was validated by numerous publications.

When we received our association data from the API, we noticed that there were diseases in that new data that were not present in the original diseases CSV we downloaded. To address this issue of missing data, we made calls to the diseases API using the ID's of the missing diseases to retrieve their extended details.

In SQL we created a gene-disease association schema consisting of three tables: genes and their qualities, diseases and their qualities, and genes and diseases associated with each other with their association scores. Using SQL allowed us to organize our nodes and relationships in a format that would be efficient for inserting into neo4j. We then saved the data from these three tables on SQL as CSV files.

## Analysis

Once we had our data in three CSV files from SQL, we began our analysis work in Neo4j and Python. We created a user interactive program so the user could call certain actions that invoke the cypher queries written in the Python driver. For example, the user could input 'subgraph' and then be prompted to enter a node and a minimum score threshold. Then, a subgraph would be displayed with that node and all of the nodes it has an association of at least the given score with. This subgraph and our other visualizations are created using NetworkX. We felt that the visualizations in Neo4j could be much improved upon if we implemented them using NetworkX. This was done by parsing through the return of the Cypher query and creating a graph in NetworkX from the edges to then display using Matplotlib. The output of the 'subgraph' command and passing in the ABCA4 gene results in the graph shown in Figure 1. From this visualization, we can see that ABCA4 is linked to eight diseases.

There is additional functionality the user can access including finding the common diseases between two inputted genes, finding the common genes between two inputted diseases, and finding statistics about the graph database such as a graph of the genes and diseases with the most associations. Figure 2, below, is an example of this command in which we can see many genes and diseases with a high number of associations. This visualization illustrates a sub-network of genes and diseases with the most associations in the total network. We can see how the most connected genes and diseases might interact with each other themselves. The construction of this sub-network utilizes a depth-first search algorithm and is displayed with NetworkX.

Other functionality in our program includes identifying the central (most connected) genes or diseases in the network. In Figure 3, we can see an example of the output from the usage of the 'central genes' command and requesting ten genes to be shown. We can see that the three genes with the most disease associations include 'TP53', 'KRAS', and 'PTEN'. In the future, we plan to add further analysis functionality of finding genes similar to an inputted one, utilizing a depth-first search algorithm to find other genes connected to the diseases associated with the given gene.

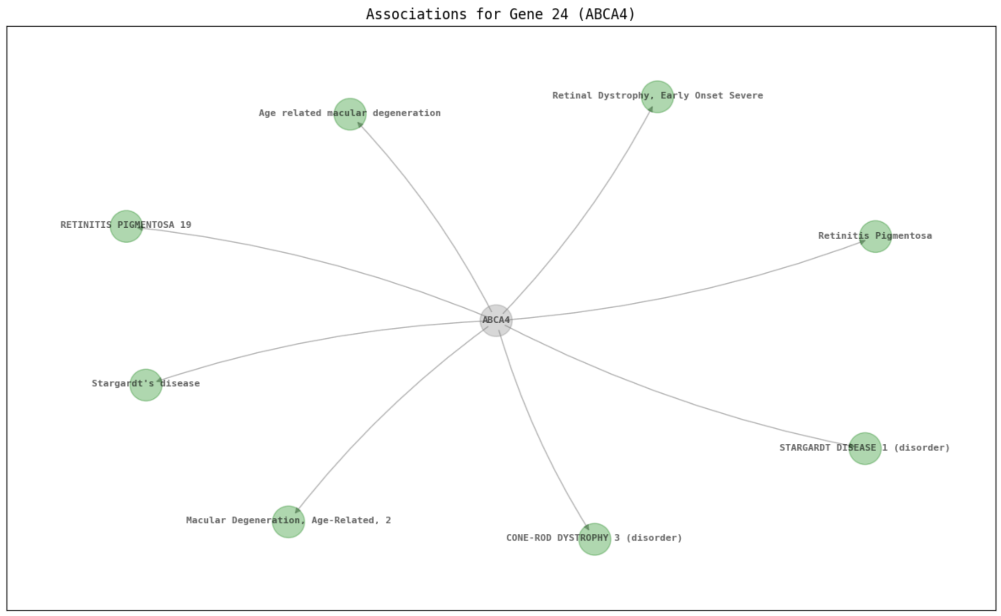
Associations for Gene 24 (ABCA4)
ABCA4
Age related macular degeneration
Retinal Dystrophy, Early Onset Severe
RETINITIS PIGMENTOSA 19
Retinitis Pigmentosa
Stargardt's disease
STARGARDT DISEASE 1 (disorder)
Macular Degeneration, Age-Related, 2
CONE-ROD DYSTROPHY 3 (disorder)

*Fig 1. Subgraph of ABCA4 gene and its disease associations*

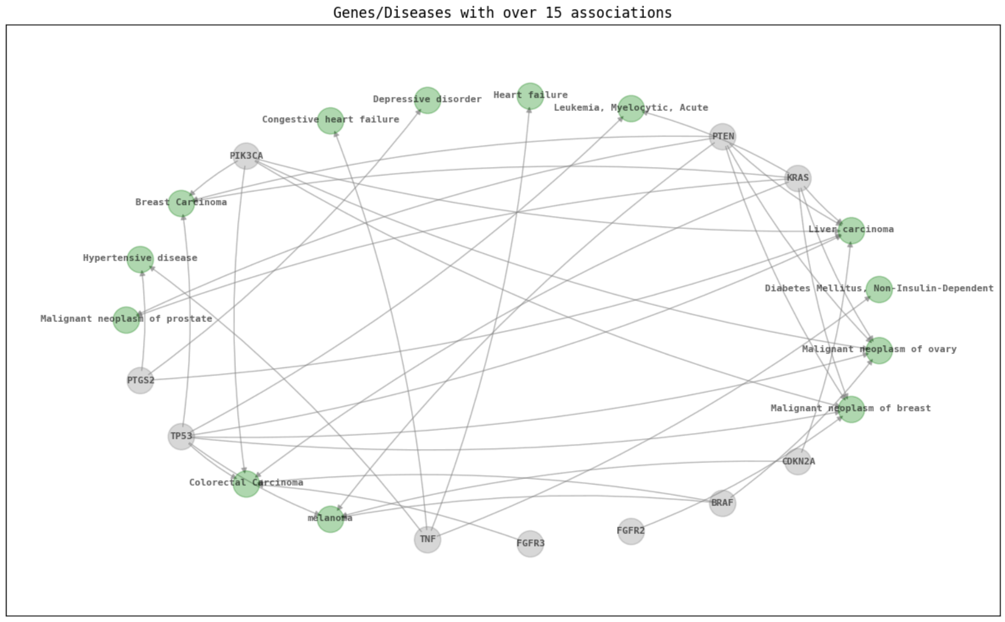
Genes/Diseases with over 15 associations
Depressive disorder
Heart failure
Congestive heart failure
Leukemia, Myelocytic, Acute
PTEN
PIK3CA
KRAS
Breast Carcinoma
Liver carcinoma
Hypertensive disease
Diabetes Mellitus, Non-Insulin-Dependent
Malignant neoplasm of prostate
Malignant neoplasm of ovary
PTGS2
Malignant neoplasm of breast
TP53
CDKN2A
Colorectal Carcinoma
BRAF
melanoma
TNF
FGFR3
FGFR2

*Fig 2. ‘High Association Graph’ of genes and diseases with at least 15 associations*

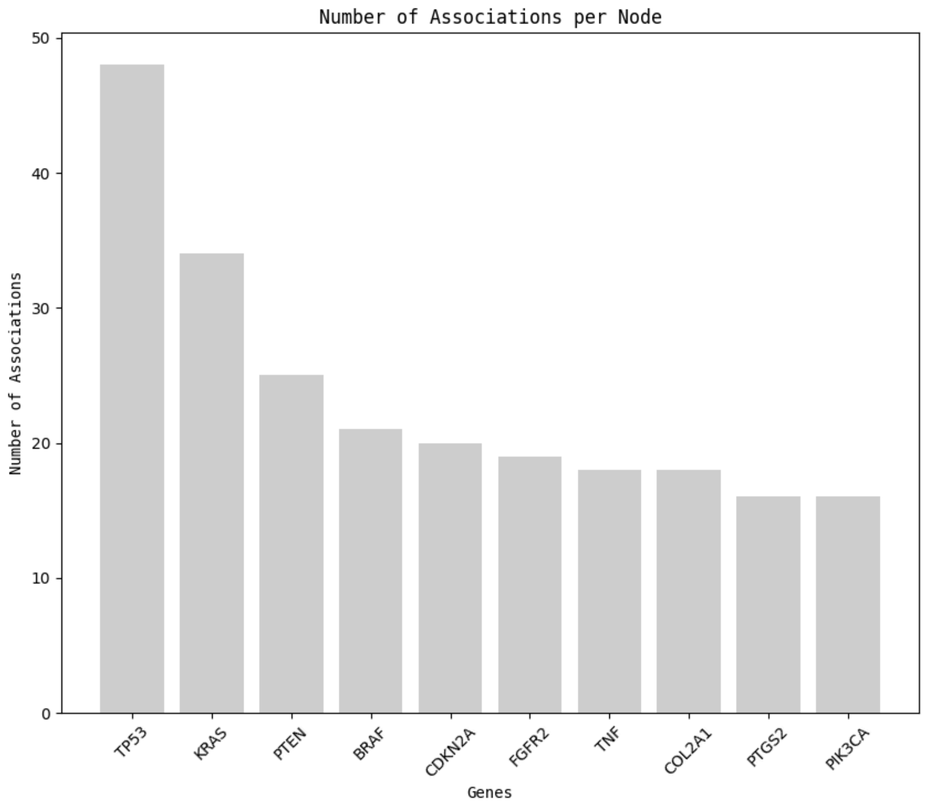

*Fig 3. Bar chart of the ten most connected genes*

## Conclusions

We found that diseases including heart failure, leukemia, depressive disorder, and genes like TNF, BRAF, and PTEN were highly connected, central nodes from our work with the recursive breadth first search. This insight could be expanded when stemming the search from other highly connected nodes, and looking into highly connected subgraphs of genes and diseases would be an extremely fruitful place for research. It may be the purview of a life scientist, but this application of the graph analysis could be useful for this research.

In terms of technologies we learned that networkX is a great Python package to work concurrently with Neo4j. It has lots of functionality for graph formatting in visualization, and can represent small graph subsets from cypher queries well.

In terms of future work we would be interested in integrating Spark or SparkSQL. We had a relatively large dataset, and working with this newer technology might have made loading and creating views of the data faster. We would also be curious to look more into developing analysis and UI functionalities with edge weights. The dataset had multiple metrics the strength of an association between a gene/disease, and these metrics could be applied for different edge weight queries and filters.

## Author Contributions

Devyanshi and Kirti focused on pulling the data from the DisGeNet API into SQL as explained in the methods section. Sree, Riya, and Ben worked on the python code to create the Cypher queries and visualize their results in NetworkX. We all contributed to the final report.

## References

1. "A Database of Gene-Disease Associations." *DisGeNET*, https://www.disgenet.org/.

2. Original data sources. DisGeNET. (n.d.). Retrieved April 19, 2023, from https://www.disgenet.org/dbinfo#section31