Building Biomedical-Domain Knowledge Graph

Yuekun Guo, Yuchen Liu
Fu Foundation School of Engineering and Applied Science
Columbia University
yg2519@columbia.edu, yl3733@columbia.edu

Abstract—**Biomedical domain has been a new cutting-edge research, and knowledge organization is necessary in this area. In this project we built up a biomedical domain knowledge graph with user interface to query the knowledge relationship. We first use Wikipedia and DBpedia APIs to build the basic graph, then crawling data from different universities and google scholar to enhance the graph. Datasets related to this domain is also used and stored in HDFS.**

Keywords-biomedical; knowledge graph; data crawling

I. INTRODUCTION

Biomedical science and biomedical engineering are research-oriented subjects that developed from biology. Since 1970s, with the rapid development of physics, information technology, medicine and material science, scientists have been finding solutions for using engineering techniques and mixing knowledge to solve problems in biology and medicine domain.

A traditional insight for organizing knowledge in specific subject is using expert system. This kind of system is emulating the decision making ability of a human expert, and it has been used in lots of well-developed but sophisticated areas[1]. However, expert system has its limits; it's using pre-defined if-else logics to make predictions, that make it unable to combine knowledge in different areas, or update new knowledge to existed ones. Biomedical domain has been owning these characters, so people turn to other possible choices. With the development of AI and big data techniques, knowledge graph provides a perfect solution.

Knowledge graph is a graph that contains different types of entities and relationships where the structured information of entities, relations are encoded in the graph. The size of the knowledge graph could be as large as millions of nodes that is related to the fields. The goal of a typical knowledge graph is to find the underline relationships using machine learning techniques, which is feasible if entities database in the graph is large enough. The most recent popular knowledge graph is the Google Freebase (Bollacker et al.2008), which is transferred to media wiki now. However, as the knowledge graph containing millions of nodes, it would take some time to find the result going through. Therefore, sometimes representation is used to simplify the query.

The existing biomedical domain knowledge graphs such as Disease Ontology, the National Drug File has the limitation that they are too specified in one single area in biomedical domain and have not enough relationship between entities. The main relationship between research, researchers and medical is still not clear to people, thus it could be hard to find the researching process of different people in different field.

Hence in our project, we mainly focused on first building the knowledge graph using multiple APIs and crawling data from different websites includes university websites, google scholar. We aim at building a knowledge graph with fundamental knowledges, finding relationship between researcher, people and fields. To combine knowledge that we got from multiple resources into the graph, we are using specific crawlers for different resource pages, and structure analysis to get uniform names and knowledge relationships. For those websites with anti-crawling techniques, agent pools and time-changed cookies are implemented. Then all the information are embedded together and saved to the graph with labels and classes that could be used to find the detailed information saved in the mongDB or HDFS.

II. RELATED WORKS

1) Data Crawling

There have been multiple mature solultions for data crawling. Although with the development of anti-crawling implemented by websites such as google scholar, existing website analyzing and fetching techniques can't be used directly, they have been providing important trains of thought in our algorithm designing.

Cho[2] provided an efficient crawling method through URL ordering. With the thought of BFS and obtaining more 'important' pages first, it provides a useful solution when the crawler cannot visit the entire web in a reasonable time.

Shkapenyuk[3] designed and implemented a high performance distributed web crawler, which runs on a network of workstations. It's a intuitive insight of distributed crawling, and could scales to several hunderd pages per second, with strong ability in dealing with system crashing or thread quitting, but the weak point is that it's hard to deal with data form processing and storing.

Python is a powerful tool in data crawling, for it's supporting open source web structure analyzing and parsing libraries. Beautifulsoup[4] is a python library for pulling data out of HTML files, and it provides idiomatic ways of navigating, searching and modifying the parse tree. Webpages such as wikipedia are very organized in context and URL links, which could be easily crawled using beautifulsoup.

Another open source crawling framework, Scrapy[5], provides even more powerful functions in those websites that URL links can not be analyzed directly. The whole scraping framework can automatically analyze and add the links in the page into queue, and they crawl them in a self-defined manner.

2) Knowledge Graph

There have been several knowledge graphs in biomedical domain such as Gene Ontology, the Disease Ontology.Gene Ontology (GO)[6] is a biology domain knowledge graph that is focused on relationship between the representation of gene and gene product attributes across all species. Goal of this graph is to maintain the knowledge related to gene and its product and provide tools to access to the annotation data. Disease Ontology[7] is developed by Institute for Genome Sciences at the University of Maryland School of Medicine mainly focuses on the disease concepts of human to meets the needs of community.

All these knowledge graph only focused on the result of research in a very specific area. Thus, they have little relationships and are not able to predict the future results.
Different embedding algorithms is another related research area in knowledge graph area. Learning Entity and Relation[8] is one basic method to embedding the graph, that is to build entity and relation in separated spaces. Another way is Dynamic Mapping Matrix[9] where two vectors are used to represent a named symbol object. Locally Adaptive Translation[10] is also used attempts to find the optimal loss function by adaptively determining its margin over different knowledge graphs.

III. System Overview

1) Crawling

The data crawling framework could be divided in to two parts: the one that used in wikipedia data crawling, and the one in google scholar data crawling. The techniques that we are using for each framework will be introduced by the next chapter.

For wikipedia data, the crawling framework is shown below in figure 1. In this framework, for crawling efficiency, we are using thread pool to scrape data from different URLs with same cookies of browser, they store the data to MongoDB if the data is usable, or drop the data if the data is unusable. We are also using a singlular name list to store website URLs and title names being crawled, so that we won't go through the same page repeatly.

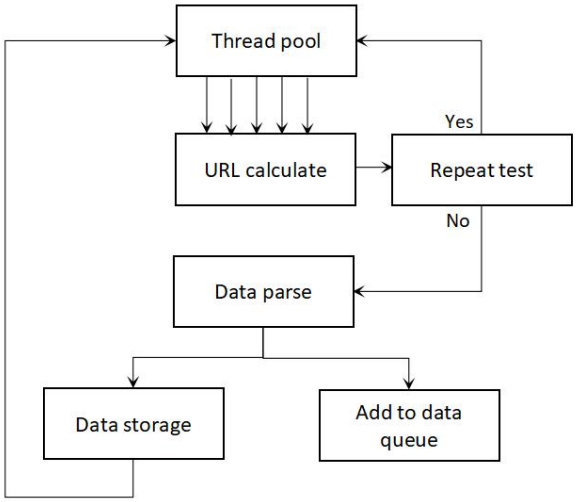

Figure 1. Flow chart for wikipedia data crwaling

Google scholars is a search engine, so the webpage organization and anti-crawling strategy is totally different from wikipedia, so we are using another framework to crawl data from it, which is shown in figure 2. Agent pool is used to change the ip address, as well as we are using different cookies of browsers, so that acts like multiple users are visiting google scholar anonymously. Also, we are using random sleeping time between visits, which makes our crawler works better.

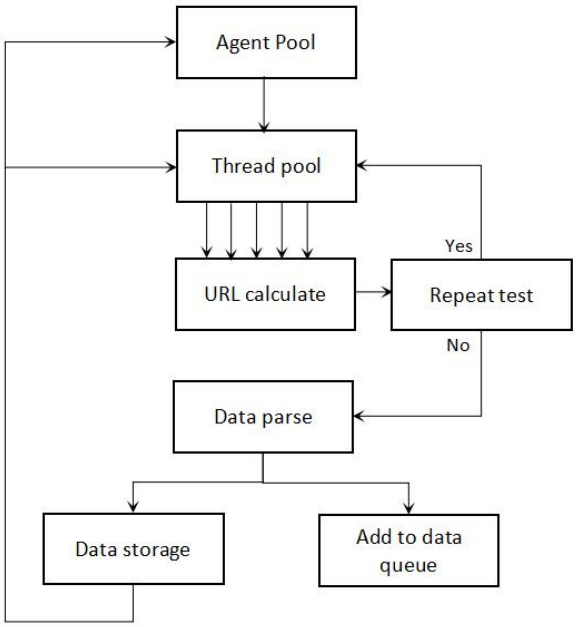

Figure 2. Flow chart for google scholars data crawling

2) Knowledge Graph

This knowledge graph consists of three parts. One is to build the knowledge graph embedding different knowledge together, the second part is to store the entities and relationships in the graph while other information is stored in another data base. The last part is while people querying to the knowledge graph, it will return and visualize the query result with links to find more about what they have found.

The sketch of the knowledge graph system is shown below in figure 3 In this system, information from different resources are used to build the graph and is embedding together. The graph only contains critical messages about entities and relationship, while the huge detail information is stored in the HDFS system. This structure is used to accelerate the research speed with the potential of finding more knowledge. Also, the usage of HDFS system can reduce the storage pressure because of the limitation of space.

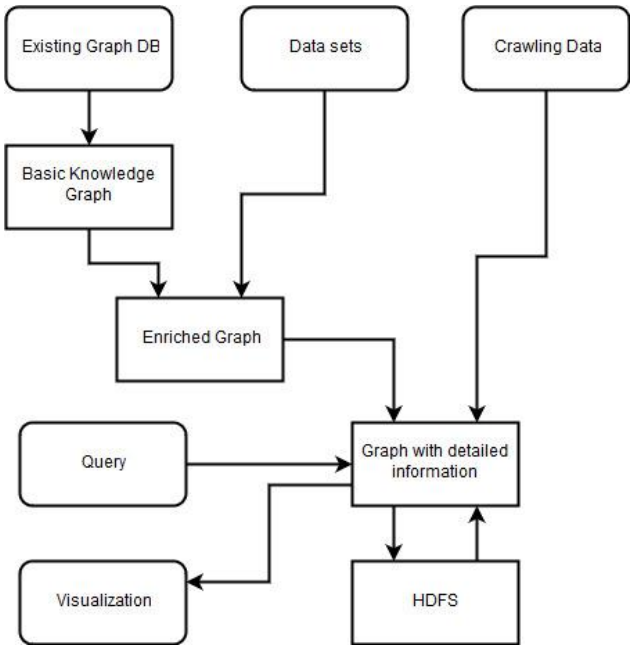

Figure 3. Flow chart for knowledge graph building

IV. ALGORITHM

1) Data Crawling

As we are using frameworks like beautifulsoup and scrapy to parse data from HTML or XML pages, so the algorithms that we designed is mainly for data processing and storage, as well as avoid being banned from specific pages. When we begin a new thread, we randomly select a request header as well as cookies we are going to use, then calculate or load URL that we are going to scrape. We then compare it with URLs that we have already have in our queue to find out whether it's the first time that we are visiting the website; if not, we drop this thread to pick a new thread or wait for next crawling process begin. If it's a new page, then we parse data from the webpage, and use string algorithm to get information that we want (title, writer names, publication names, etc.) and store them in matrices. Then we add a unique sign, make pair with its URL, and add this information to our data queue. After all threads in one round of processing are done, we sleep the crawler for a random time, then begin our next round.

2) Knowledge Graph

The algorithms that the knowledge graph used is to embedding all different kind of things together. In this project, we basically used the Learning Entity and Relation[8], that is one basic method to embedding the graph, that is to build entity and relation in separated spaces, and Locally Adaptive Translation[10] which attempts to find the optimal loss function by adaptively determining its margin over different knowledge graphs. For example, when new relationship about a researcher is coming, we first need to find out which specific node it is as people could have same name or symbol. Therefore, we need to check the birthday, institution, degrees and so on to guarantee that the relationship is added to the right person.

V. SOFTWARE PACKAGE DESCRIPTION

1) Data Crawling

We are using python script in the data crawling part, and a lot of libraries and packages are used to avoid being blocked from google scholar, which contains spyder, requests, threadpool, selenium, and other techniques such as Tor. The libraries that we are using to parse data contains beautifulsoup and scrapy.

(1) Spyder

The spyder is used to build the coding environment. It has integrated scientific libraries, as well as stable environment for multi-thread processing.

(2) Requests

Requests is the most powerful tool for getting request head and cookies, as well as build up an agent pool with multiple agent browsers. Figure 4 shows how we use specific agent in our request message.

```
agent = 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko
url = "https://en.wikipedia.org"
```

Figure 4. We still need to deal with ip address and cookies we are going to use before send the request

(3) Threadpool

Threadpool is a library that allow users to run multiple threads to improve efficiency. Users can set the value of thread number, and multiple attempts will run on the same agent.

(4) Selenium

Selenium is a library that automates browsers. It's a tool that allows user to virtualize a browser and simulate its behavior, just like what real browsers behave - and we are using it to run anonymous web agents. The agent will be blocked if it runs for multiple times, but usually we are using the same information for only one times inside one round of process.

(5) Tor

Tor is short for 'The onion route', and it's an open source tool for users to hide their communications; it also allows users to visit network anonymously. The user of Tor is running a onion proxy in local machine, which periodically connect with other Tor users, so that a virtual circuit is built among Tor network (see figure 5). It's encrypting the communication in application payer, so that ensures the safety of connection.

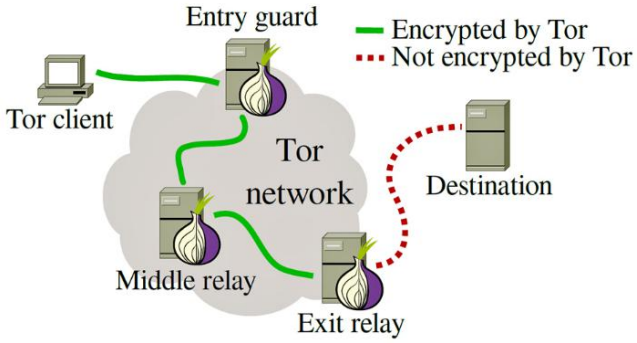

Figure 5. A work diagram for Tor

As we have been using the agent pool and selenium, Tor's characters make it hard to cooperate with other packages, so we just used it in the test.

(6) Beautifulsoup

Beautifulsoup is the mostly used crawling tool around the world, and it provides powerful functions such as navigating, searching and encoding. The way that beautifulsoup analyze XML files is shown in figure 6, and as long as we indicate how to process the data the we fetched according to names of subtitles, beautifulsoup can do all other works.

```
print(soup.prettify())
# <html>
#  <head>
#   <title>
#    The Dormouse's story
#   </title>
#  </head>
#  <body>
#   <p class="title">
#    <b>
#     The Dormouse's story
#    </b>
#   </p>
#   <p class="story">
#    Once upon a time there were three little sisters; and their names were
#    <a class="sister" href="http://example.com/elsie" id="link1">
#     Elsie
#    </a>
#    ,
#    <a class="sister" href="http://example.com/lacie" id="link2">
#     Lacie
#    </a>
#    and
#    <a class="sister" href="http://example.com/tillie" id="link2">
#     Tillie
#    </a>
#    ; and they lived at the bottom of a well.
#   </p>
#   <p class="story">
#    ...
#   </p>
#  </body>
# </html>
```

Figure 6. How beautifulsoup parse HTML and XML files

(7) Scrapy

Scrapy is a well developed crawler based on python, and it's built up a complete framework including data transmitting and data storage; users only need to define the parsing rules based on needs (figure 7), so we used it to crawl parts of data from university BME webpages, and store the information of professors in .csv files.

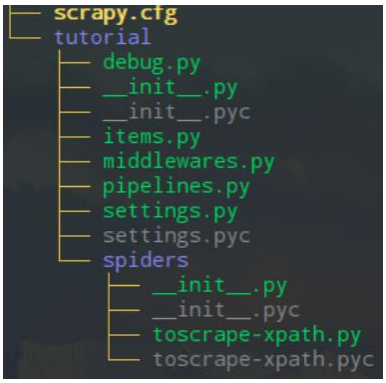

Figure 7. A framework for scrapy

2) Knowledge Graph

To build this knowledge graph system, a lot of software as well as packages are used. In the knowledge graph part, the software contains python, anaconda, neo4j, MongoDB and hadoop.

(1) Anaconda

The anaconda is used to deploy the jupyter notebook server on remote ec2 machine to use python more easily. The following figures are the starting of jupyter notebook and UI.

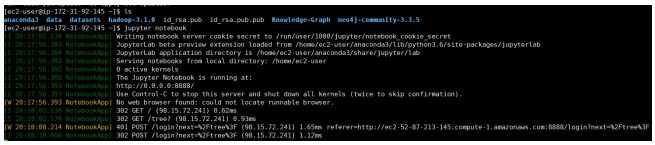

Figure 8. Start page of Jupyton notebook console

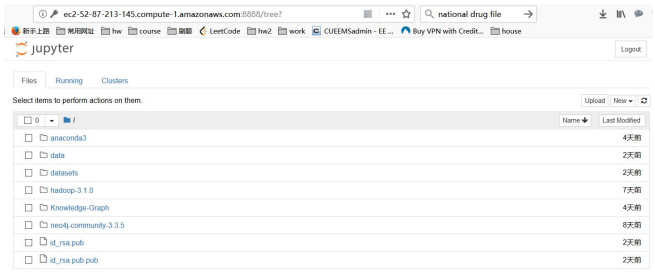

Figure 9. Start page of Jupyton notebook UI

The python packages used this part are wikipediaapi, sparQL, pymongo, py2neo, pandas, numpy and hdfs.

(2) Neo4j

Neo4j is a great graph management database containing visualization function and query system.

The following images are the starting of neo4j and UI.

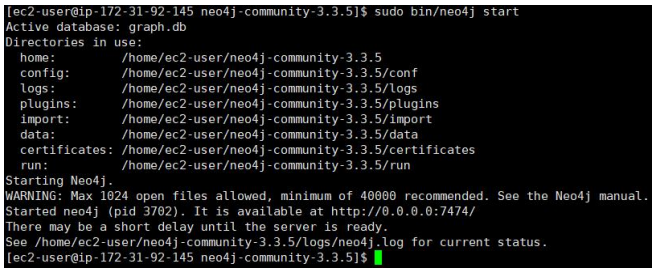

Figure 10. Start page of neo4j console

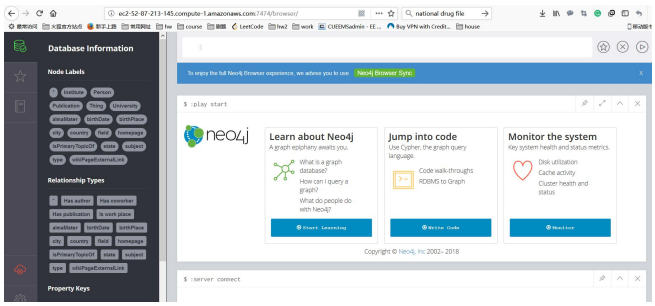

Figure 11. Start page of neo4j UI

(3) MongoDB

MongoDB[11] is a database program classified as a NoSQL database program, who uses JSON-like documents with schemas. The following is a mongoDB desktop that can be used to view the data.

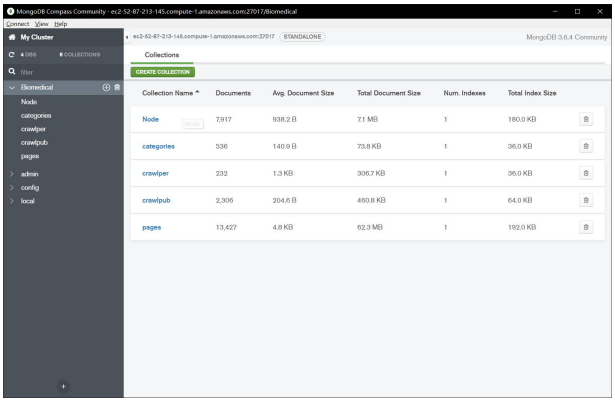

Figure 12. MongoDB compass

(4) Hadoop

The file system of hadoop is Hadoop distributed file system. The HDFS is a distributed, scalable, and portable file system written in Java for the Hadoop framework. The following is the start of the hadoop system and the usage is to use command on the command line or using python packages.

Figure 13. Start page hadoop system

VI. Experiment Results

1) Data Crawling

We have been getting data from different sources:

(1) From organized dataset

The data we got from organized dataset is mainly from DBpedia API, National center for health statistics, and Kaggle BME dataset. With the data size of over 2GB, we are able to do data processing and choose the node that we want and add them to the graph.

(2) From web (crawled)

In the crawling part, our data resource is wikipedia and google scholar. For wikipedia, we begin our crawling from Biomedical Engineering and Biomedical Science page, with eventually over 600MB data. For google scholar, we first found names from Top 20 and 50 universities in the U.S. on area of BME, then add 10-20 professors in each school to our crawling list; according to those names, we get their publications as well as co-writers from google scholar, and that gives us over 12000 nodes (Top 20) and over 27000 nodes (Top 50) with full relationships between those professors and publications.

2) Knowledge Graph

Using the data we got above, the knowledge graph built are shown below.

(1) The data stored in MongoDB

Firstly, we use the MongoDB to store all the data from APIs. From Wikipedia, we have all the name of nodes related to the biomedical domain, that has 13427 nodes.

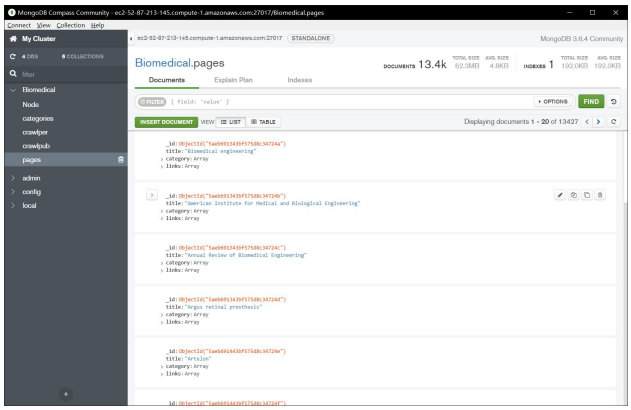

Figure 14. Data from APIs

Then we use the DBpedia to specify those entities and the result is shown below. There are 8000 reasonable names about the biomedical domain as shown below.

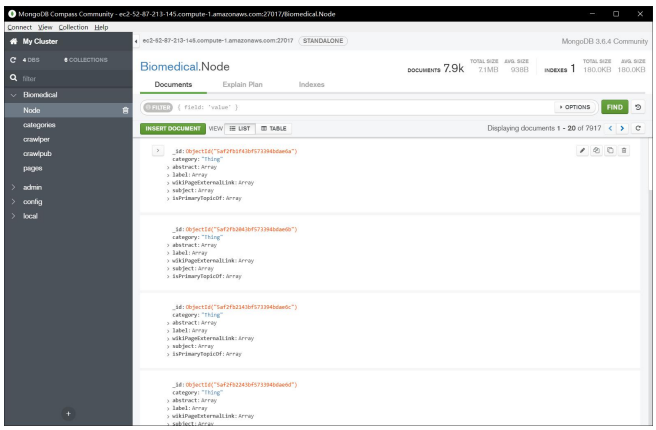

Figure 15. Data from DBpedia

(2) The crawling data stored in MongoDB and HDFS

We also transfer the form of crawling data to the form of graph-like data with nodes and relationship. We use first letter of first name and whole last name to match names from google scholar and wikipedia, and a total number of around 9000 nodes (Top 20) are added to the graph after relationship calculating.


```
_id: ObjectId("5af3de9a43bf572a202aaac5")
category: "Person"
label: "Danino T"
publication: Array
    0: "A synchronized quorum of genetic clocks"
    1: "A sensing array of radically coupled genetic 'biopixels'"
    2: "Entrainment of a population of synthetic genetic oscillators"
    3: "Hepatitis C virus RNA functionally sequesters miR-122"
    4: "Queueing up for enzymatic processing: correlated signaling through cou..."
    5: "Programmable probiotics for detection of cancer in urine"
    6: "Synchronized cycles of bacterial lysis for in vivo delivery"
    7: "Streaming instability in growing cell populations"
    8: "Genetic circuits in Salmonella typhimurium"
coworker: Array
    0: "Mondragón-Palomino O"
    1: "Tsimring L"
    2: "Prindle A"
    3: "Samayoa P"
    4: "Mondragón-Palomino O"
    5: "Selimkhanov J"
    6: "Luna J M"
    7: "Scheel T K"
    8: "Cookson N A"
    9: "Mather W H"
    10: "Prindle A"
    11: "Kwong G A"
    12: "Din M O"
    13: "Mather W"
    14: "Mondragón-Palomino O"
    15: "Prindle A"
    16: "Selimkhanov J"
institute: "Columbia University"
```


Figure 16. Data from google scholar

We also store the crawling data and related datasets into HDFS system and access them when needed through python package.

(3) The graph stored in neo4j

Finally, we save the simplified entities and relations into the neo4j graph data base for quick search and visualization.

The relationship between biomedical domain names are shown below. From this graph, we can see that all the names are connected with each other by their categories and relations.

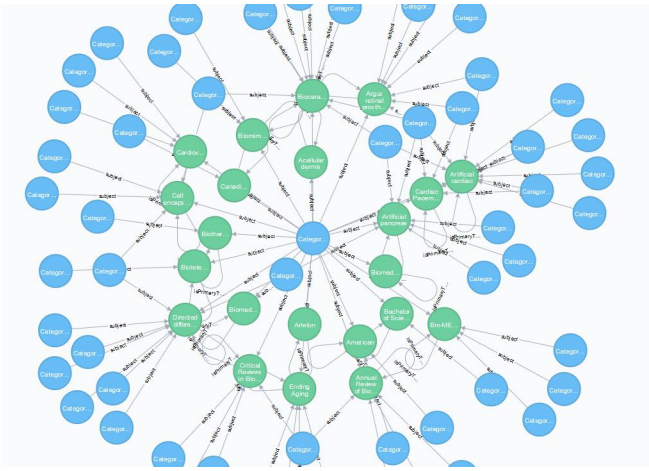

Figure 17. Graph between names (part example)

The staff of a university is shown below. This is the structure of people in this domain of one university that has relations between each other in other fields such as their publications.

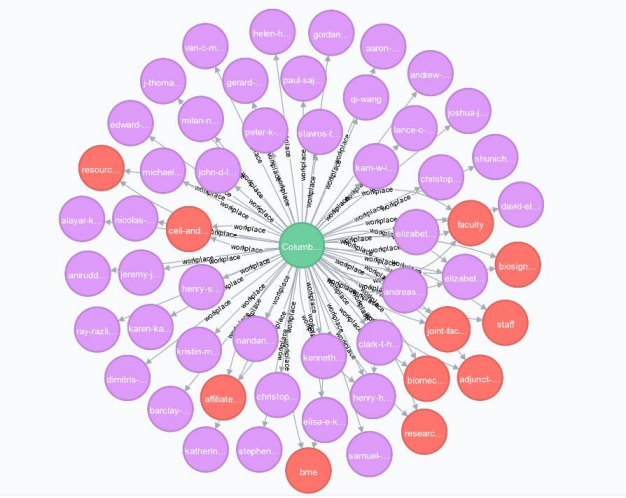

Figure 18. Graph of a certain university (part example); with top 20 and 50 universities, a even larger co-operate graph could be built

The relationship between researchers and publications are shown below. We can tell the closeness of people by telling how many times they have cooperate with each other and how close their researches are. From top 20 and 50 universities, we are getting about 4000 researchers; the calculation of the whole map may take decades of times, but we can easily find the relationship of a specific node with the query system.

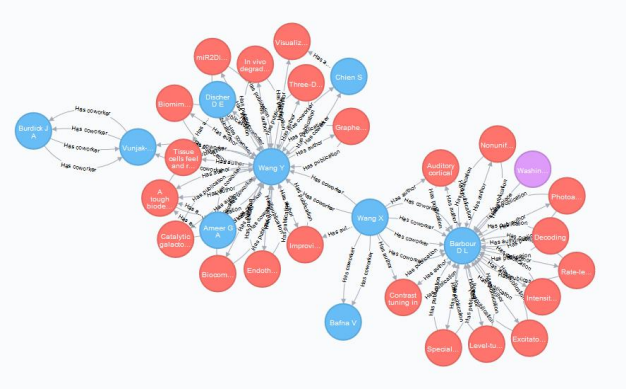

Figure 19. Graph between researchers (part example)

From these images of our storage data and visualization knowledge graph, we can see that the graph is in great structure and is fully connected. Every aspect of this graph is reasonable and we can search for the expected related knowledge by going through this graph. All the embedding algorithm is well functioned.

VII. Conclusion

In our project, we have built up a new knowledge graph that focuses on relationship between biomedical domain researchers and their publications, as well as the specific areas they are making contribution to. The relationship could be easily find out through the generated graph or the query system, and we can easily generate a part of the knowledge graph so that to get information that we are interested in. Also, the knowledge graph is easy to be updated; with adding nodes and relationships to the graph, previous works and nowadays works could be gathered and matched. This method of knowledge organizing will lead to an impact on those cutting-edge, rapidly changing researching areas.

In our project, Yuchen Liu mainly focuses on data crawling as well as algorithm designing and data processing, and Yuekun Guo mainly focuses on knowledge graph building, includes using APIs and embedding new data to the graph.

In the future, more data would be crawled and other existing knowledge graph in biomedical domain could be embedded into this one. Also, a more functional query page will be developed, so that users outside of the specific areas can also do searching in this graph.

Acknowledgment

The authors would like to thank professor Ching-Yung Lin, and TA Vishal Anand.

Appendix

Github Links:

Project Link: https://github.com/Sapphirine/Biomedical-Domain-Knowledge-Graph

YuchenLiu:https://github.com/YuchenLiuColumbia

YuekunGuo:https://github.com/ykwen

Video Link: https://youtu.be/sLnF-YUjLJM

References

[1] Nakai, K. and Kanehisa, M., 1991. Expert system for predicting protein localization sites in gram-negative bacteria. Proteins: Structure, Function, and Bioinformatics, 11(2), pp.95-110.

[2] Cho, J., Garcia-Molina, H. and Page, L., 1998. Efficient crawling through URL ordering. Computer Networks and ISDN Systems, 30(1-7), pp.161-172.

[3] Shkapenyuk, V. and Suel, T., 2002. Design and implementation of a high-performance distributed web crawler. In Data Engineering, 2002. Proceedings. 18th International Conference on (pp. 357-368). IEEE.

[4] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[5] https://scrapy.org/

[6] Gene Ontology Consortium, 2007. The gene ontology project in 2008. Nucleic acids research, 36(suppl_1), pp.D440-D444.

[7] http://do-wiki.nubic.northwestern.edu/do-wiki/index.php/Main_Page#Disease_Ontology_Wiki

[8] Torkestani, J.A., 2012. An adaptive focused web crawling algorithm based on learning automata. Applied Intelligence, 37(4), pp.586-601.

[9] Olson, M., 2010. Hadoop: Scalable, flexible data storage and analysis. IQT Quart, 1(3), pp.14-18.

[10] Lin, Y., Liu, Z., Sun, M., Liu, Y. and Zhu, X., 2015, January. Learning entity and relation embeddings for knowledge graph completion. In AAAI (Vol. 15, pp. 2181-2187).

[11] https://en.wikipedia.org/wiki/Mongodb