Twitter-Based Knowledge Graph for Researchers

Final Report

CS 4624: Multimedia, Hypertext, and Information Access
Virginia Tech
Blacksburg, VA 24061

April 26, 2020

Client: Prashant Chandrasekar
Professor: Dr. Edward Fox
Authors: Emma Meno, Kyle Vincent

Contents

List of Figures

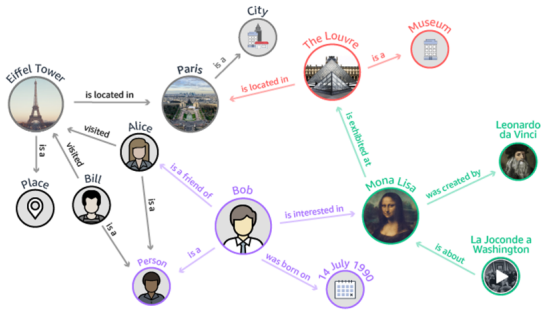
List of Tables

1 Executive Summary

The Twitter-Based Knowledge Graph for Researchers project is an effort to construct a knowledge graph of computation-based tasks and corresponding outputs to be interfaced by subject matter experts and statisticians. A knowledge graph is a directed graph of knowledge accumulated from a variety of sources. An example is shown in Figure 1. For our application, Subject Matter Experts (SMEs) are experts in their respective non-computer science fields but are not necessarily experienced with running heavy computation on datasets. As a result, they find it difficult to generate workflows for their projects involving Twitter data and advanced analysis. Workflow management systems and libraries that facilitate computation are only practical when the users of these systems understand what analysis they need to perform. Our goal is to bridge this gap in understanding. Our queryable knowledge graph will generate a visual workflow for these experts and researchers to achieve their project goals.

After meeting with our client, Prashant Chandrasekar, we established two primary deliverables. First, we needed to create an ontology of all Twitter-related information that an SME might want to answer. Secondly, we needed to build a knowledge graph based on this ontology and produce a set of APIs to trigger a set of network algorithms based on the information queried to the graph. An ontology is simply the class structure/schema for the graph. Throughout future meetings, we established some more specific additional requirements. Most importantly, the client stressed that users should be able to bring their own data and add it to our knowledge graph. As more research is completed and new technologies are released, it will be important to be able to edit and add to the knowledge graph. Next, we must be able to provide metrics about the data itself. These metrics will be useful for both our own work, and future research surrounding graph search problems and search optimization. Additionally, our system should provide users with information regarding the original domain that the algorithms and workflows were run against. That way they can choose the best workflow for their data.

Figure 1: Example of a Knowledge Graph. Adapted from [1].



After implementing our original solution on a CentOS virtual machine hosted by the Virginia Tech Department of Computer Science, we transitioned our solution to Grakn, an open-source knowledge graph database that supports hypergraph functionality. When finalizing our workflow paths, we noted some nodes depended on completion of two or more inputs, representing an "AND" edge. This phenomenon is modeled as a hyperedge with Grakn, initiating our transition from Neo4J to Grakn. Currently, our system models and produces a possible workflows (which can involve hyperedges) after a user list selects a leaf node in the knowledge graph. A majority of the work for this project involved developing the nodes and edges comma-separated values files, compiled from projects collected in a literature review.

2 Introduction

2.1 Objective

Advanced algorithms, big data analysis, data science, and heavy computation are no longer topics confined Computer Science circles. Researchers from departments all around the university and industry are asking questions about these topics now. The problem is that these Subject Matter Experts have a hard time understanding what tools they have at their disposal, which tools to use for their research, and how to use those tools. This Knowledge Graph for Researchers project seeks to span this knowledge gap. In a broader sense, this project also aids efficiency. Often times, the most time consuming part of a project can be the research required to answer such questions about the tools and databases available to the researcher. Helping these experts start their work faster can help boost productivity and advancement. We want our knowledge graph interface to be easy to use, extensible, and applicable to a variety of fields and research interests. Our solution will be primarily aimed at SMEs in non-CS fields. But, as a stretch goal, we also want to allow statisticians to gather metadata about this type of graph itself. This project supports a broader doctoral research initiative aiming to build interfaces for SMEs to conduct their research with advanced analysis.

2.2 Client Background

Our client, Prashant Chandrasekar is a fifth year PhD student working in the Digital Library Research Laboratory at Virginia Tech. His primary areas of research include digital libraries, natural language processing, and knowledge representation. This project serves as a proof of concept for his dissertation.

2.3 Project Background

Virginia Tech has been collecting a massive number of tweets since the tragic events of April 16, 2007, the Virginia Tech campus shooting claiming 32 innocent lives. This effort has been conducted by projects including Integrated Digital Event Archive Library (IDEAL) as well as Global Event and Trend Archive Research (GETAR). Tweets gathered cover a variety of areas, such as hurricanes, solar eclipses, and school shootings. Our knowledge graph would be very useful in a variety of domains. For the scope of our project, we will just be focusing on the vast amount of Twitter data available from these projects as part of the CS 5604 Information Retrieval course. The idea behind this project was the brainchild of Prashant Chandrasekar in conjunction with Dr. Edward A. Fox, who teaches this capstone course.

2.4 Organization of this Report

Following the summary and introduction, this report lays out the requirements established for the Knowledge Graph for Researchers project, followed by the initial design of the system with subheadings for specific areas of the design. After that is the implementation section which will have similar subheadings but will describe the concrete manifestation of the design. Section 6 contains testing, evaluation, and assessment information which goes into detail about how this system can be tested and judged for its solution to the few sections are the most lengthy, i.e., the user manual and developer's manual. To wrap up the report, we will discuss some lessons learned by the project team, and include acknowledgements and references.

3 Requirements

3.1 Return Path Between Two Nodes

The most essential requirement of the solution is the primary function of the query service. As input, the user will say "this is the data or dataset result I am trying to get to", and "this is the data I have", and our graph search query will need to either return a workflow path between those nodes if possible, and if not give them potential paths so they can see the point that they need to get to. The manner in which this is accomplished is flexible, whether it be simply on the command line, a user clicking manually on nodes, or the user searching through a catalog. Along with this workflow path output should be all the information associated with each node and edge, allowing the user to have as much help as possible.

3.2 Allow User Import of Data

While we have gathered a significant amount of data to build the graph, we understand that more data is out there and will definitely become available. With every semester comes a new batch of project reports, each describing either a new workflow or a new technology for a particular workflow that could process data faster. It is imperative that our design can accommodate new data coming into the system. It will be crucial to have a defined format for data imports to keep things consistent. The ability to add data from different domains other than Twitter down the line will also be necessary, so we need to create a way for users to create additional graphs on the same database.

3.3 Database Contents

While updates to our database schema in the future are almost certainly inevitable, we want to keep those changes few in number. Because of this, numerous meetings with our client have occurred to nail down a wide array of parameters that our Subject Matter Experts would be interested in. In our graph, both datasets and algorithms needed to be represented as the two primary classes of data. An algorithm would be represented as an edge between an input dataset and an output dataset result. Parameters for these tuples include the names of the three items (Input Dataset - Algorithm - Output Dataset), the data collection the operation was run on, any command line parameters used, whether the operation was manual or automatic and the report that the data was gathered from.

3.4 User Interface

The interface with which the users interact with their data needs to be as intuitive as possible. The client has given us a wide range of possibilities for how to implement this interface (command line, API, Neo4j Console, Javascript Website). The complexity of the interface will be a function of how much time the project team devotes to it, so that aspect is up in the air. No matter the implementation however, it needs to be user-friendly. The user should learn their options of datasets to choose from and either click or type in their choice. The workflow needs to be clean, rich in information, and exportable to other programs for future reference.

3.5 Security

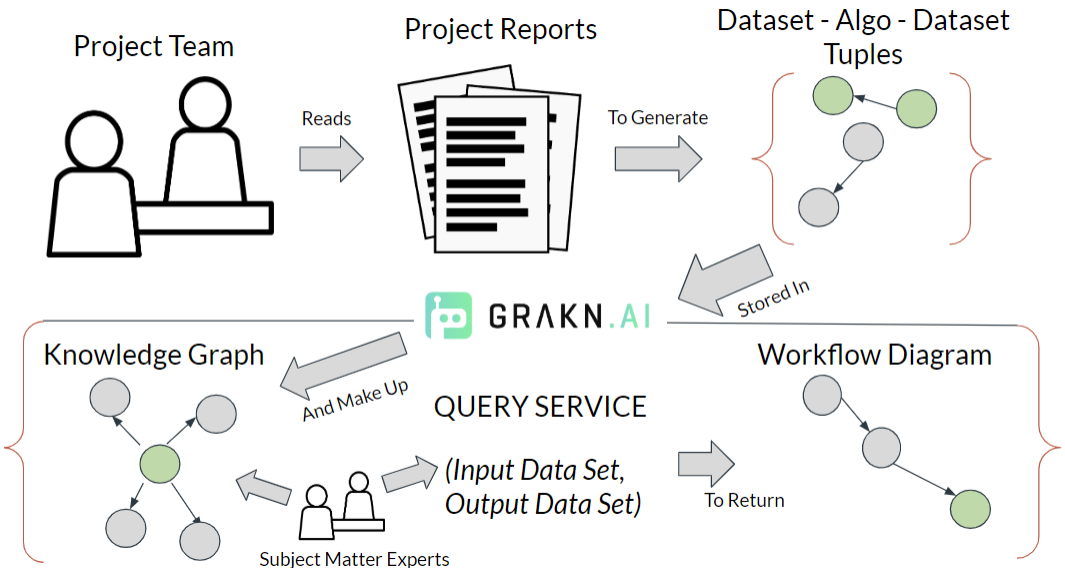
With any software comes the need for security. The data contained within the graph should only be manipulated by trusted users to ensure data loss or corruption does not occur. At the same time, the ability to add users to the system and grant permissions needs to be an easy operation from a developer perspective, as new Subject Matter Experts will have need of this service as time goes on.

4 Design

4.1 Overview

The basic architecture of the Knowledge Graph for Researchers project is illustrated in Figure 2.

Figure 2: Knowledge Graph Workflow Architecture



4.2 Data Collection

All of the data for the knowledge graph is taken from past projects dealing with Twitter data collections. The project team was responsible for reading and parsing through those reports and their accompanying PowerPoint presentations. These reports could be lengthy and often required extensive knowledge of the project goals to extract the necessary information. As a result, this was the most time consuming aspect of the project.

The reports analyzed throughout this report are those dealing specifically with Twitter data from the CS 5604 Information Retrieval course, written as part of the IDEAL and GETAR projects for 2016 and 2017.

The 12 specific project reports included in the knowledge graph produced as a result of this project are listed below:

- CS5604 Fall 2016 Classification Team Final Report [2]
- CS5604 Fall 2016 Collection Management Tweets Team Final Report [3]
- CS5604 Spring 2016 Clustering and Social Networks for IDEAL Final Report [4]
- CS5604 Fall 2016 Collection Management Webpages Team Final Report [5]
- CS5604 Spring 2016 Classification Project Final Report [6]
- CS5604 Spring 2016 Topic Analysis Final Report [7]
- CS5604 Spring 2016 Collection Management for IDEAL [8]
- CS5604 Spring 2016 Collaborative Filtering for IDEAL [9]
- CS5604 Fall 2016 Clustering and Topic Analysis Final Report [10]
- CS5604 Fall 2017 Collection Management Webpages Team Final Report [11]
- CS5604 Fall 2017 Clustering and Topic Analysis Team Final Report [12]
- CS5604 Fall 2017 Collection Management Tweets Team Final Report [13]

From each of these 2016 and 2017 CS 5604 reports, we collected information on the Twitter data analysis algorithms. We assigned a unique Task ID and Task Name to each algorithm and then collected attributes that would help us in constructing and representing the ontology. These attributes were:

- Input File(s)
- Output File(s)
- Functions/Libraries
- Manual (Y/N)
- Data Collection
- Command Line Parameters
- Project
- Report Name

We then compiled all of this information into a spreadsheet. Because the data collection is a collaborative effort, Google Sheets is recommended as a temporary data store for the tuples of datasets and algorithms.A snapshot of our Google Sheet with the raw tuple data is illustrated in Table 1.

Table 1: Google Sheet Tuple Raw Data

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Task ID	Task Name	Input File(s)	Output File(s)	Dependent Task ID	Functions / Libraries	Manual (Y/N)	Data Collection	Command Line Parameters	Project			Report Names
2	**2016 Reports (CS 5604: Information Retrieval)**												
3	0	Remove stop words and lemmatize	Cleantext from HBase (RDD of tweets)	Cleaned Lemmatized Tweets		StanfordNLP	Y		table name data type for each row-key desired columns	https://vtechworks.lib.vt.edu/handle/10919/73713			CS5604 Fall 2016 Classification Team Final Report
4	1	Manually Classify Tweets	Cleaned Lemmatized Tweets AND Set of Events Classes	Tweet Training Data	0	N/A	Y		collection ID batch size	https://vtechworks.lib.vt.edu/handle/10919/73713			CS5604 Fall 2016 Classification Team Final Report
5	2	Feature Selection	Tweet Training Data	Word Feature Representation	1	Word2Vec Model	N			https://vtechworks.lib.vt.edu/handle/10919/73713			CS5604 Fall 2016 Classification Team Final Report
6	3	Feature Selection	Tweet Training Data	Word Feature Representation	1	Association Rules Based Classifier (provided by Dr. Pereira)	N		path_to_jar_file support threshold block size cluster node base_dir_on-HDFS path_to_training_file path_to_test_file stopwords_file output_directory	https://vtechworks.lib.vt.edu/handle/10919/73713			CS5604 Fall 2016 Classification Team Final Report
7	4	Tweet Classification	Word Feature Representation	Predicted Class of the Given Tweet // Associated Probabilities of Each Class	2,3	Multi-class logistic regression classifier	N		collection number batch size --retrain --metric<training file> <test file>	https://vtechworks.lib.vt.edu/handle/10919/73713			CS5604 Fall 2016 Classification Team Final Report
8	5	Convert to CSV	Cleaned Tweets in HBase	CSV Format Tweets	37	Pig Script	N			https://vtechworks.lib.vt.edu/handle/10919/73739			CS5604: Information and Storage Retrieval Fall 2016 - CMT (Collection Management Tweets)
9	6	Record User, URL, and Tweet Relationships	CSV Format Tweets	Social Network Matrix	5		Y			https://vtechworks.lib.vt.edu/handle/10919/73739			CS5604: Information and Storage Retrieval Fall 2016 - CMT (Collection Management Tweets)
10	7	Calculate Importance Factors for Nodes	Social Network Matrix	Weighted Social Network Matrix	6		Y			https://vtechworks.lib.vt.edu/handle/10919/73739			CS5604: Information and Storage Retrieval Fall 2016 - CMT (Collection Management Tweets)

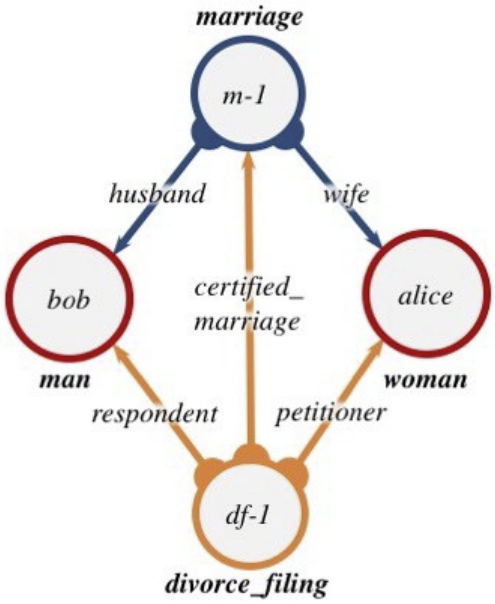
4.3 Data Storage

The project team initially decided to use a Neo4j Graph Database for the persistent storage of the data. Neo4J is an industry favorite for graph databases, and has been utilized by past projects for our client. However, after analyzing the raw tuple data, we found the presence of some "AND" edges, where the outputs of two or more algorithms fed in as input to a single algorithm. Neo4J did not have the functionality that would specifically support such configurations. Our client recommended Grakn as an alternative knowledge graph database system, which would support hypergraph modeling for queries. Hypergraphs generalize the traditional node-edge-node framework of graphs by redefining the edges between nodes as hyperedges, which are sets of vertices [14]. The Grakn hypergraph data model (HDM) is built on three premises [14] :

1. hypergraphs are built from a non-empty vertex set and a hyperedge set
2. a hyperedge represents a finite vertex set (separable by specific roles played in that hyperedge)
3. a hyperedge represents a vertex itself, possibly connected by other hyperedges

An illustration of one such HDM graph is shown in Figure 3. In this example, there are four vertices: bob, alice, m-1, df-1. But, m-1 and df-1 also act as hyperedges: m-1 describes a binary marriage relationship and df-1 represents a ternary divorce filing relationship.

Figure 3: Grakn Hypergraph Data Model. Adapted from [14].



Graph databases (including Grakn) require their data in a specific format, that being a table of nodes and a table of edges. Since this schema differs from the Google Sheets schema of raw tuples, a conversion to this new schema was required. This conversion was done manually by the project team, but future work into automating this step is necessary. Note that the intermediate data store (representing the data in its raw form in a spreadsheet similar to that shown in Table 1) is still necessary, because it is hard to conceptualize the data in terms of nodes and edges, especially for cross examination of whether the node or edge is already present in the data.

Tables 2 and 3 illustrate snapshots of the structure for the nodes and edges csv files respectively for our project application. The nodes.csv file simply contains the fileID and name for each node in the knowledge graph. The edges.csv file contains column information for: taskId, name, inputId, outputId, functionsAndLibraries, manual, commandLineParameters, reportUrl, reportName, and domainCollection. The information for each of the edge entries was manually entered using the data in the raw tuples Google Sheets file.

Table 2: Formatted Table of Nodes for Graph Database

	A	B
1	fileId	name
2	1	Cleantext from HBase (RDD of tweets)
3	2	Cleaned Lemmatized Tweets
4	3	Set of Events Classes
5	4	Tweet Training Data
6	5	Word Feature Representation
7	6	Predicted Class of the Given Tweet
8	7	Associated Probabilities of Each Class
9	8	Cleaned Tweets in HBase

Table 3: Formatted Table of Edges for Graph Database

	A	B	C	D	E	F	G	H	I	J
1	taskId	name	inputId	outputId	functionsAndLibraries	manual	commandLineParameters	reportUrl	reportName	domainCollection
2	1	Remove stop words and lemmatize	1	2	StanfordNLP	Y	table name; data type for each row-key; desired columns	https://vtechworks.lib.vt.edu/handle/10919/73713	CS5604 Fall 2016 Classification Team Final Report	Kentucky Shooting, Newton Shooting, Firefighter Shooting, Hurricane Arthur, Hurricane Sandy, Hurricane Isaac, China Factory Explosion, Texas Plant Explosion, Manhattan Explosion
3	2	Manually Classify Tweets	2	4	N/A	Y	collection ID; batch size	https://vtechworks.lib.vt.edu/handle/10919/73713	CS5604 Fall 2016 Classification Team Final Report	Kentucky Shooting, Newton Shooting, Firefighter Shooting, Hurricane Arthur, Hurricane Sandy, Hurricane Isaac, China Factory Explosion, Texas Plant Explosion, Manhattan Explosion
4	3	Manually Classify Tweets	3	4	N/A	Y	collection ID; batch size	https://vtechworks.lib.vt.edu/handle/10919/73713	CS5604 Fall 2016 Classification Team Final Report	Kentucky Shooting, Newton Shooting, Firefighter Shooting, Hurricane Arthur, Hurricane Sandy, Hurricane Isaac, China Factory Explosion, Texas Plant Explosion, Manhattan Explosion
5	4	Feature Selection	4	5	Word2Vec Model	N		https://vtechworks.lib.vt.edu/handle/10919/73713	CS5604 Fall 2016 Classification Team Final Report	Kentucky Shooting, Newton Shooting, Firefighter Shooting, Hurricane Arthur, Hurricane Sandy, Hurricane Isaac, China Factory Explosion, Texas Plant Explosion, Manhattan Explosion
6	5	Feature Selection	4	5	Association Rules Based Classifier (provided by Dr. Pereira)	N	path_to_jar_file; support threshold; block size; cluster node; base_dir_on-HDFS; path_to_training_file; path_to_test_file; stopwords_file; output_directory	https://vtechworks.lib.vt.edu/handle/10919/73713	CS5604 Fall 2016 Classification Team Final Report	Kentucky Shooting, Newton Shooting, Firefighter Shooting, Hurricane Arthur, Hurricane Sandy, Hurricane Isaac, China Factory Explosion, Texas Plant Explosion, Manhattan Explosion
7	6	Tweet Classification	5	6	Multi-class logistic regression classifier	N	collection number; batch size; --retrain; --metric<training file><test file>	https://vtechworks.lib.vt.edu/handle/10919/73713	CS5604 Fall 2016 Classification Team Final Report	Kentucky Shooting, Newton Shooting, Firefighter Shooting, Hurricane Arthur, Hurricane Sandy, Hurricane Isaac, China Factory Explosion, Texas Plant Explosion, Manhattan Explosion
8	7	Tweet Classification	5	7	Multi-class logistic regression classifier	N	collection number; batch size; --retrain; --metric<training file><test file>	https://vtechworks.lib.vt.edu/handle/10919/73713	CS5604 Fall 2016 Classification Team Final Report	Kentucky Shooting, Newton Shooting, Firefighter Shooting, Hurricane Arthur, Hurricane Sandy, Hurricane Isaac, China Factory Explosion, Texas Plant Explosion, Manhattan Explosion

4.4 Data Retrieval

The design for data retrieval can take a variety of forms. The data in the knowledge graph database could be retrieved asynchronously, so we need to ensure that multiple users can query the database at the same time. An API will have to be utilized so that the Subject Matter Experts won't need to understand Cypher graph query statements in order to interact with their data. This applies to data import was well, though a detailed user manual can alleviate confusion in this area. As stated in Section 3.1, the user will be able to enter in, either by clicking or typing, an input and output in order to get a path between the two nodes. Our revised design goal is to implement a system where users can list select a leaf and the system returns all possible paths to that leaf. The paths will be generated using a simple shortest path algorithm, but every possible path between the two nodes will be returned to the user. The result of this will be a visual workflow diagram that the SME can use as a basis for their work.

5 Implementation

5.1 Initial Implementation

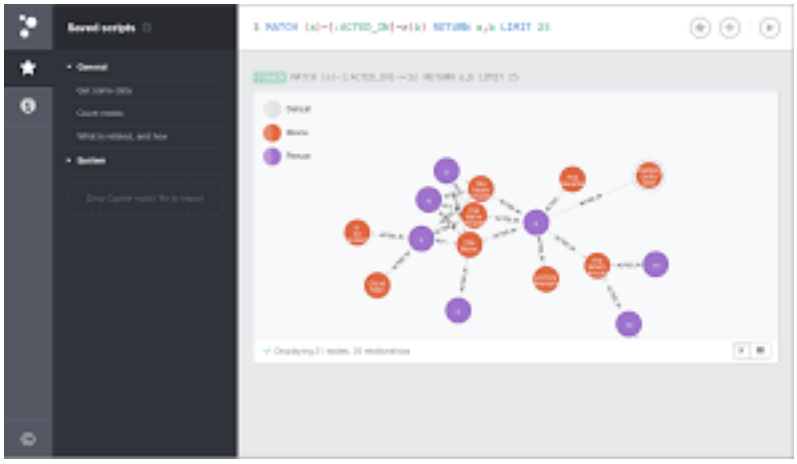
5.1.1 Overview

At the time of the interim report, the implementation of the design was still in process. We have inserted a brief description of our initial implementation here to better represent the design process and illustrate some of the main concerns with our first iteration. At the interim mark, all reports had been processed and the data gathered. We originally setup a Neo4j database for the project. The data was converted into the node/edge format and loaded into the database. We were looking into options for the visual and interface aspect of the project, specifically centered on the GRANDStack as discussed in Section 5.1.3.

5.1.2 Data Storage

Our initial Neo4j graph database was provisioned on the virtual machine (VM). Neo4J is a very powerful tool that can be used as a visual tool, and even supports its own query language: Cypher. An example of what data looks like in the Neo4j console is shown in Figure 4.

Figure 4: Neo4j Console Data View. Adapted from [15].



The data was stored as nodes and edges, inputted as two different .csv files: nodes.csv and edges.csv. The user could open the console using any browser and visualize their data. While the console view was nice, it did not offer the level of customizability that we would like to have for this project. We began exploring other alternatives for representation and settled on the implementation discussed in Section 5.2.

5.1.3 Initial Solution: The GRANDstack

One solution that we were exploring for our initial implementation was the GRANDstack, which is a development stack for graph database applications that has just been taking shape recently [16]. The GRANDstack consists of 4 parts:

- GraphQL: A relatively new paradigm for building APIs, GraphQL is a way of describing data and enabling clients to query it [16]
- React: A popular JavaScript library for building front end interfaces
- Apollo: Acts as the connection between GraphQL and both Neo4j and React. It is a suite of development tools for creating workflows in GraphQL.
- Neo4j Database: Graph Database Solution

Figure 5: Initial Implementation: The GRANDstack. Adapted from [16]

We initially felt that this combination of tools, while new to us, gave the project the greatest chance of success. There are many ways in which these tools can talk to each other efficiently, allowing us to focus less so on the connections and more so on the front end. Our API would be a GraphQL API that simplifies queries to the database. It can also use JavaScript packages to connect to our front end in React. The biggest question mark with this stack is Apollo and React, two tools neither of us on the project team have had experience with. With later stages in development, the biggest drawback to this implementation method was the lack of support for "AND" edges in Neo4J. After discussions with our client Prashant, we migrated our implementation to Grakn.

5.2 Revised Implementation

5.2.1 Overview

Our initial implementation was built on a dual-core 16 GB RAM CentOS Machine provided by the Computer Science. This revised implementation was built on one of our local dual-core 16 GB Windows machines. The instructions and manuals in the remainder of the report all pertain to Windows operating systems, which may only be a few differences in implementation. This revised implementation is way flatter, as it takes a 4 technology stack and trims it down to 1, with Grakn. As previously mentioned in this report, Grakn is a graph database technnology that is open-source, and focuses on hypergraph flexibility, allowing us to make more customizable path queries and schema definitions. We found that Neo4j did not allow us to draw edges that connected two input nodes to one output node, in cases where it requires two inputs.

We still used the same workflow for gathering the data and storing it in CSV files. Grakn is used for the data store and Graql, its own query language is used for our queries to the graph. Grakn is a very easy to install technology out of the box, but as will be explained later, can be hard to learn and interface with.

5.2.2 Workflow Metrics

Overall, the project team analyzed 12 reports between the Fall 2016, Spring 2016, and Fall 2017 CS 5604 Information Retrieval projects, as listed in Section 4.2. From those 12 project reports, we generated an initial raw ontology with 112 nodes (representing datasets) and 95 edges (representing algorithms). We found these numbers of nodes and edges to be slightly high for our application. We also noted that the number of single-length paths was high: out of 50 possible paths, 22 had length 1. After meeting with our client, involving a thorough manual review process, we pared the dataset down to 91 nodes and 89 edges. In this review, we found duplicate nodes where some teams used other teams' output datasets as input. We also eliminated a series of single-edge paths for the SOLR Team visualization algorithms. Any algorithms requiring cleaned tweets as input were also updated to start with the following, where the edge is indicated parenthetically:

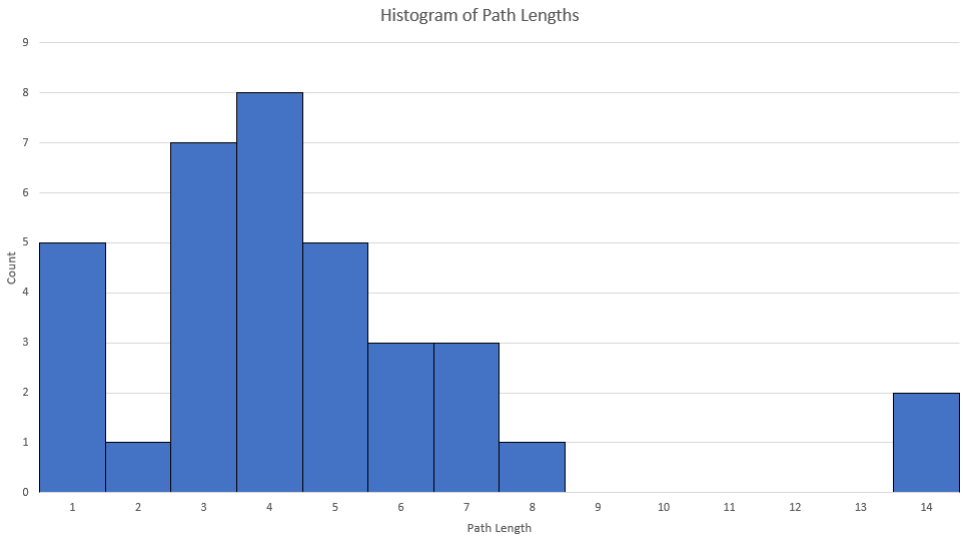
Uncleaned Tweets - (Tweet Cleanup) - Cleaned Tweets in HBase

Adding this prefix helped reduce the number of single-length paths. The new metrics for path lengths in our knowledge graph are shown in Table 4 and Figure 24.

Table 4: Path Lengths in Knowledge Graph

Path Length	Count
1	5
2	1
3	7
4	8
5	5
6	3
7	3
8	1
14	2
Total	35

Figure 6: Path Lengths Graph.



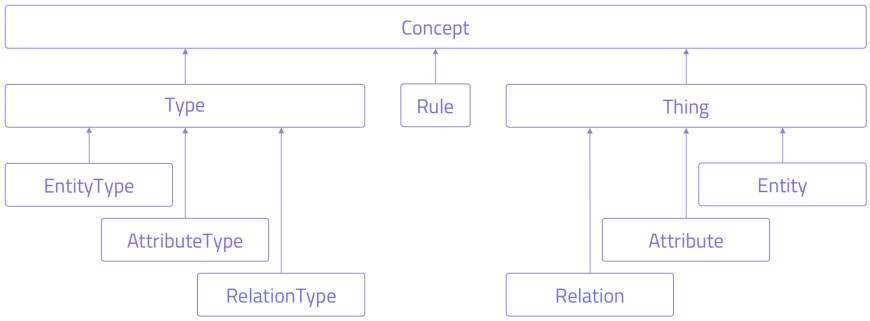
5.3 Grakn Database

Figure 7: Grakn Logo. Adapted from [17]

Grakn is the database storage that was used for this project. All interfacing with the data: reads, writes, and analysis must be done via Graql queries. Grakn is flexible in that it provides three different client api's for this interfacing : Node.js, Python, and Java, some of which were explored for this project. However, all of those transactions still revolve around strings of text in the form of the Graql Query. The database was set up on our local machine in the C:/ directory.

As opposed to Neo4j and other traditional RDF data stores, Grakn has developed it's own unique Concept Architecture. Similar to a traditional class diagram, everything within the grakn architecture is contained in the diagram in Figure 8 [18]. Everything in the database is stored as a concept: a type, rule, or thing. Things are the entities (nodes), relations (edges), or attributes. Types describe those things. Rules are inference logic that can be used to make on-the-fly categorizations. We will see in the developer manual how all these things interact, and how to build queries using the console to access each of the concepts.

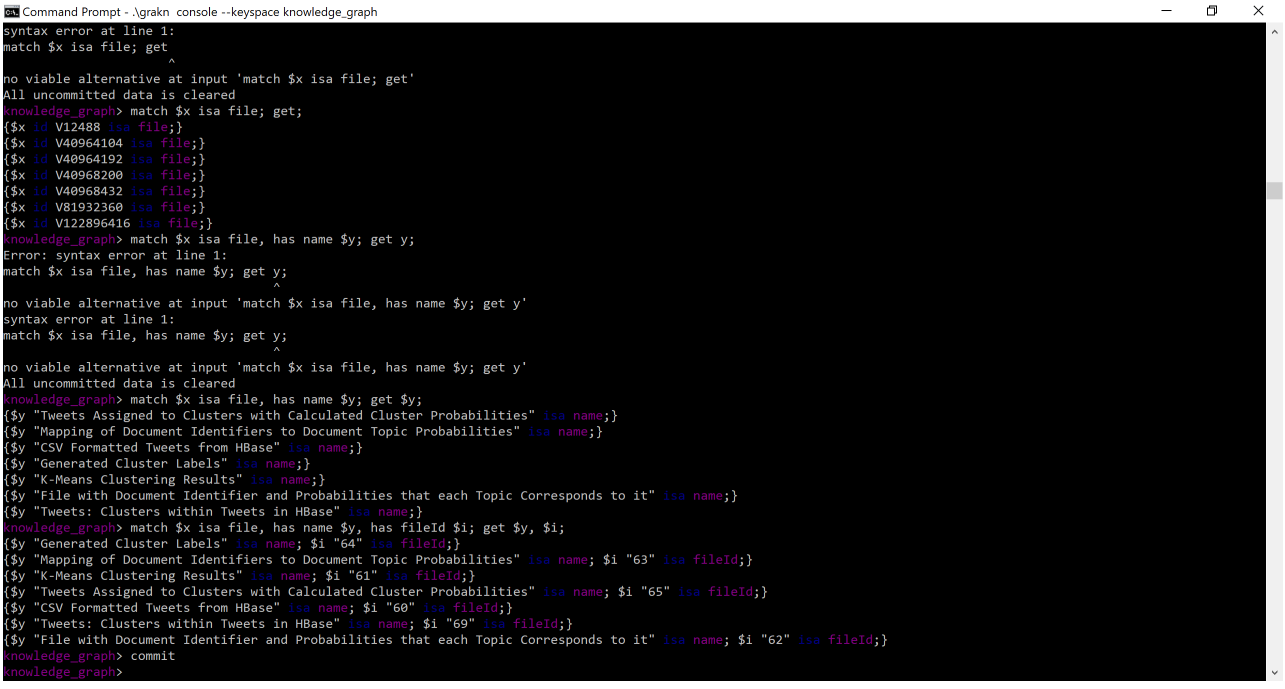
Figure 8: Grakn Concept Architecture. Adapted from [18]



5.4 Grakn Console

The Grakn Console will be how we interact with the data in the database. It can also be used to input the data manually using a series of "insert" statements. The console comes with the Grakn Core installation described in this report. We will go into further detail about sample queries to use.

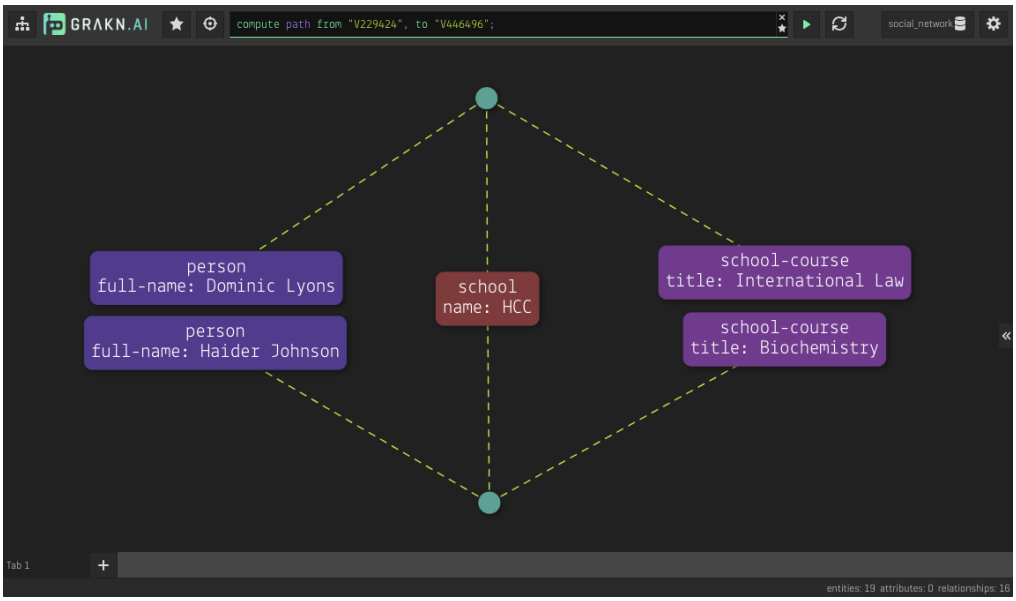
Figure 9: Grakn Console



5.5 Grakn Workbase

Grakn Workbase is Grakn's graph visualization tool. It allows users to customize how they view their data, as well as execute Graql queries using the command line at the top. It also allows users to create data schemas visually by clicking the icon in the top left. Workbase is a very new tool, and lacks sufficient documentation. As a result, we were unable to incorporate it into the project, but details for how to set it up will be included in the manual.

Figure 10: Grakn Workbase Graph Visualizer. Adapted from [19]



6 Testing/Evaluation/Assessment

At time of writing, our client is currently testing out solution. Ultimately, we were unable to establish the graph database, API, and front-end on a virtual machine. As a result, the only way for our solution to be tested would be for the person evaluating it to have control of our personal laptop. Seeing as that is not possible, the way this will be tested is whether someone can follow our Developer and User Manual and reproduce what we have created. This project has taken a turn to be research and data acquisition focused. We have gathered data to be used in a knowledge graph for researchers, and we have tried a few ways to interface with that data. We hope that eventually the project will be continued to be a more fleshed out user-friendly version.

7 Developer's Manual

7.1 About

The purpose to give developer's step by step instructions on how to set up a Grakn database, create a Grakn schema, migrate data into from .csv files, and setup a Workbase interface. This specifically shows developer's how to complete these tasks for the Twitter Knowledge Graph for Researchers project. This manual assumes the reader has some programming experience, and their own personal Windows machine where they have root permissions. We installed and operated our database on a dual-core 16GB RAM Windows machine. The speed of the Grakn database depends on the size of the dataset. For small datasets, you will not need to sacrifice much storage space. Grakn recommends a minimum on 1 core and 2 GB RAM, but recommends 4 cores and 8 GB RAM for an optimal.[20]

This manual assumes the developer has experience with either Python or Java, and is at least somewhat familiar with the Windows cmd prompt. The Graql language will likely be new to the developer, but we can walk through the basics to help the developer get accustomed to it. By using this language, the developer will be able to gather insights from their data and create flexible schemas for any future graph application.

7.2 Installing Grakn

First, verify that you have Java 8 installed on your machine by opening the cmd prompt on Windows and typing "java -version". If you see something like below, you should be all set. If you need to download java,

Figure 11: Seeing Java Version

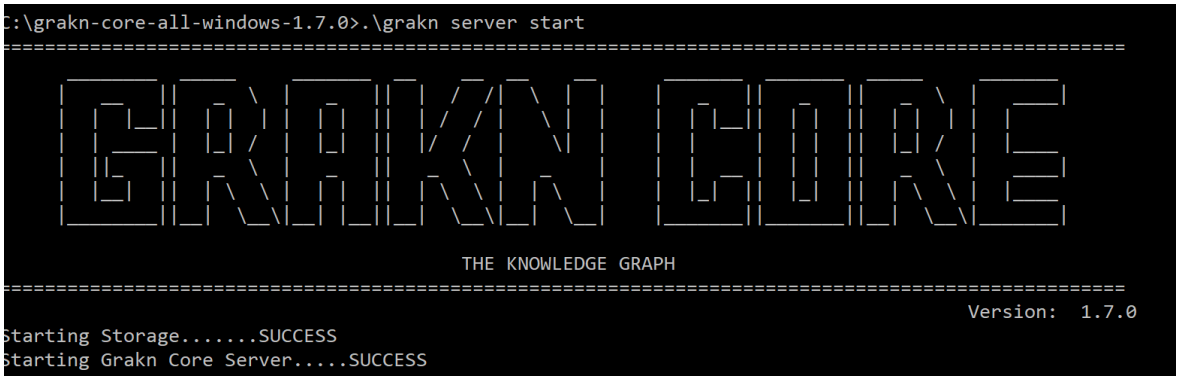
```
C:\grakn-core-all-windows-1.7.0>java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) Client VM (build 25.251-b08, mixed mode, sharing)
```

navigate to `http://openjdk.java.net/install/` and download Java JDK 8, or the latest version linked in the installation steps for Grakn found at `https://dev.grakn.ai/docs/running-grakn/install-and-run#system-requirements`.

You can download the latest version of Grakn Core for Windows at `https://grakn.ai/download?os=linux#core`. While you are on this page, you might as well Grakn Workbase too. Once it's downloaded, unzip it into your C:/ folder, so you should end up with a folder path "C:/grakn-core-all-windows-1.7.0/", or whatever grakn version you happen to download.

Now open up you cmd prompt, and navigate to the new folder you downloaded. Once there, type "\grakn server start". Once you hit enter you should hopefully receive a dialog similar to the one in Figure 12.

Figure 12: Server Start Up Message



7.3 The Schema File

In order to use the Grakn Core database, you need to first define how your data will be structured. The schema file used for this project is schema.gql, and will be included in the submission. The next few screenshots will describe that schema. Our project is based on datasets and algorithms, or files and tasks. The tasks are the operations that need to be run on one file to produce another. See Figure 13 to follow along. The logical way to set up this scenario in Graqn is to establish file as our entity, and task as our relation. Every schema file needs to start with "define". "file" is a subclass of entity, while "task" is a subclass of relation. Each of these concepts have attributes that match up with the columns in our ontology files, edges.csv and nodes.csv. The "key" specifier is used for attributes that are meant to be unique.

Figure 13: Schema.gql part 1: setting up files and tasks

```
define
#Our dataset nodes
file sub entity,
  key fileId,
  has name,
  plays inputfile,
  plays outputfile;

#The tasks that connect the nodes
task sub relation, abstract,
  key taskId,
  has name,
  has inputId,
  has outputId,
  has functionsAndLibraries,
  has manual,
  has commandLineParameters,
  has reportUrl,
  has reportName,
  has domainCollection,
  relates inputfile,
  relates outputfile,
  plays ancestor,
  plays decendent;

#points from input to output
produces sub task,
  relates inputfile,
  relates outputfile;

need-two-inputs sub task,
  relates outputfile,
  relates inputfile;
```

We make task an abstract relation so that the following two relations, produces, and need-two-inputs, can inherit all of the attributes from the task concept. Abstract classes cannot have any entities "constructed" from them. As a result, our graph will have edges that point from input files to output files called "produces", and edges that point from output files to input files called "need-two-inputs". The latter relation is how we will differentiate the "AND" issue described earlier. "relates" is a keyword that is used to define roles. Entities and Relations can each play roles. For instance, in our graph, an entity can either be an inputfile in a task, or an outputfile in a task, so it can play either of those two roles. We will address the ancestor and decendent roles later. Yes, we understand that decendent is actually spelled descendant, but that's the advantage of variable names.

The second part of the file, shown in Figure 14, describes entites, relations, rules, and attributes. If we want eventually connect nodes and edges based on whether they are on the same path, we need to establish a relation that uses transitive property to connect them. We will call this relation "workflow", and it will relate ancestors and decendents, similar to how ancestors and descendants could be separated by several relationships on a family tree. We now need to introduce a new concept in Grakn, called a Rule. Rules are logic when/then statements that are used to set/change concepts on-the-fly, meaning that these rules don't come into play until you enter a query into the console, which we will come to later.

After the transitive property workflow rules, we establish a sub entity to file, called "leaf" that will represent all files that are leaf nodes in the graph. If a file never plays the inputfile role, then it is a leaf. After this, we need to define the attribute types for all of the attributes described in the file. Conveniently, they are all strings.

Now that the schema file is done, save it inside of your grakn folder, and return to the cmd prompt. While you are within that folder on the command line, enter the following command: ./grakn console –keyspace knowledge_graph –file C:/grakn-console-all-Windows-1.7.0/schema.gql

If no errors pop up, your schema is now loaded into the database. We use the keyspace "knowledge_graph" for this project, which represents the database and schema combo you use. Now our graph just needs some data.

Figure 14: Schema.gql part 2: Establishing Rules and Attributes

```
workflow sub relation, relates ancestor, relates decendent;
  workflow1 sub rule,
      when {
          (inputfile: $x, outputfile: $y) isa produces;
      },
      then {
          (ancestor: $x, decendent: $y) isa workflow;
      };

  workflow2 sub rule,
      when {
          (inputfile: $x, outputfile: $z) isa produces;
          (ancestor: $z, decendent: $y) isa workflow;
      },
      then {
          (ancestor: $x, decendent: $y) isa workflow;
      };

leaf sub file;

  leafrule sub rule,
      when {
        $x isa file;
        not {(inputfile: $x, outputfile:$y) isa produces;};
      },
      then {
        $x isa leaf;
      };

fileId sub attribute, datatype string;
name sub attribute, datatype string;
taskId sub attribute, datatype string;
inputId sub attribute, datatype string;
outputId sub attribute, datatype string;
functionsAndLibraries sub attribute, datatype string;
manual sub attribute, datatype string;
commandLineParameters sub attribute, datatype string;
reportUrl sub attribute, datatype string;
reportName sub attribute, datatype string;
domainCollection sub attribute, datatype string;
```

7.4 Data Migration

7.4.1 Introduction

Data migration is definitely the most time consuming aspect of working with Grakn, but if you can get the script to run with no errors, it can actually be a very quick process. Our two data files, edges.csv and nodes.csv need to be converted to Grakn data. There are a variety of ways to do this. You can use one of 3 clients supported by Grakn: Java, Python, or Node.js, or manually enter the data one by one. For this project, our team explored the Python and Java options, but ultimately ended up entering a small segment of our data manually because the tools did not work as described. The following section will walk through dependencies, installation steps, and our code to migrate the data .

7.4.2 Python

The steps we used to work with the grakn client can be found here: `https://dev.grakn.ai/docs/client-api/python`. Just make sure that you already have Python 3+ installed on your machine. Once you have the grakn-client installed for python, you can start working through the migrate code shown in the following figures. This code follows a format explained in the following article: `https://blog.grakn.ai/loading-data-and-querying-knowledge-from-a-grakn-knowledge-graph-using-the-python-client-b764a476cda8`

Figure 15: Python part 1: Inputs

```
def parse_data_to_dictionaries(input):
    items = []
    with open(input["data_path"] + ".csv") as data: # 1
        for row in csv.DictReader(data, skipinitialspace = True):
            item = { key: value for key, value in row.items() }
            items.append(item) # 2
    return items

inputs = [
    {
        "data_path": "C:/grakn-core-all-windows-1.7.0/data/nodes",
        "template": file_template
    },
    {
        "data_path": "C:/grakn-core-all-windows-1.7.0/data/edges",
        "template": task_template
    }
]
print("running")
build_knowledge_graph(inputs=inputs)
```

It's important to know where you are storing the data. Each csv file represents a different concepts we are trying to load. edges.csv contains all of the task data while nodes.csv contains all of the file data. Make sure to also include the parse data to dictionaries function that will be used later. Place these code snippets at the bottom of the file. You'll notice the main for this code is running the build knowledge graph function after printing "running". Let's look at that function now.

First we need to make sure to import the grakn.client package as well as the csv package for this code to work. This function takes the inputs we just saw, and after establishing a grakn connection to our keyspace, calls the load data into grakn function.

The Load Data function, Figure 17, uses the input templates and executes the queries returned by each of the template functions.

Figure 16: Python part 2: Building the Graph Main function

```
from grakn.client import GraknClient
import csv

def build_knowledge_graph(inputs):
    with GraknClient(uri="localhost:48555") as client:
        with client.session(keyspace = "knowledge_graph") as session:
            for input in inputs:
                print("Loading from [" + input["data_path"] + "] into Grakn ...")
                load_data_into_grakn(input, session)
```

Figure 17: Python part 3: Load Data

```
def load_data_into_grakn(input, session):
    items = parse_data_to_dictionaries(input)
    for item in items:
        with session.transaction().write() as transaction:
            graql_insert_query = input["template"](item)
            print("Executing Graql Query: " + graql_insert_query)
            transaction.query(graql_insert_query)
            transaction.commit()

    print("\nInserted " + str(len(items)) + " items from [ " + input["data_path"] + "] into Grakn.\n")
```

The template functions are the real meat and potatoes of the code. They take in each line of csv and generate a graql query for it so that it can be added to the database. Each query starts with an insert statement and ends with a semicolon. For the file query, we simply have to set it's fileId and name. The task template is much more complicated however. Since it is possible for our inputId field in the csv to say "33 AND 8" for example, we need to account for when we have to make two relations instead of just one, between an output and input node. Task concepts also have a lot more attributes, as seen by the length of the code snippet in Figure 18.

7.4.3 Java

The process of creating and running the migration script in Java was infinitely more difficult than in Python, but for some developers it may be the preferred client. Grakn provides some documentation on how to get started with the Java Client: `https://dev.grakn.ai/docs/client-api/java`. However, this leaves out a lot of helpful information. In order to run the Grakn client, you will need to have not only the Java JDK installed, but Maven as well. Maven is used to compile and package the Java program. We will not go into detail about the code for this, but the file will be included with our other files as migrateCSV.java, mostly because the general code structure is the same but also because it just has a lot more lines.

The two most helpful links we found for setting up the Java code were:

- `https://www.vogella.com/tutorials/ApacheMaven/article.html`
- `https://blog.grakn.ai/using-grakn-ai-to-stream-twitter-data-35682032864f`

Make sure that you add all of the necessary dependencies described to the pom.xml file that you generate. When you come to the point where you have to generate the maven project, make sure you add the -DGeneratePom=true flag to the generation statement. For instance, you should have something like this: mvn archetype:generate -DgroupId=ai.grakn -DartifactId=twitterexample -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false -DGeneratePom=true

The pom.xml file that we used has been included with the other documents, but it is important that it is placed in the right spot, so just copy the code over to your autogenerated one. It is possible that Maven

Figure 18: Python part 4: Templates

```python
def file_template(file):
    # insert file
    graql_insert_query = 'insert $file isa file, has fileId "' + str(file["fileId"]) + '"'
    # file is a file
    graql_insert_query += ', has name "' + file["name"] + '"'

    graql_insert_query += ";"
    return graql_insert_query

def task_template(task):
    inputIds = str(task["inputId"]).split(" AND ")
    # match input file
    graql_insert_query = 'match $input isa file, has fileId "' + inputIds[0] + '";'
    if len(inputIds) == 2:
        # match second input file
        graql_insert_query += ' $input2 isa file, has fileId "' + inputIds[1] + '";'
    # match output file
    graql_insert_query += ' $output isa file, has fileId ' + str(file["outputId"]) + ';'
    if len(inputIds) == 2:
        # insert contract
        graql_insert_query += " insert $newtask (outputfile: $output, inputfile: $input) isa need-two-inputs;"
        graql_insert_query += ', has taskId "' + str(task["taskID"]) + '"'
        graql_insert_query += ', has name "' + task["name"] + '"'
        graql_insert_query += ', has functionsAndLibraries "' + file["functionsAndLibraries"] + '"'
        graql_insert_query += ', has manual "' + file["manual"] + '"'
        graql_insert_query += ', has commandLineParameters "' + file["commandLineParameters"] + '"'
        graql_insert_query += ', has reportUrl "' + file["reportUrl"] + '"'
        graql_insert_query += ', has reportName "' + file["reportName"] + '"'
        graql_insert_query += ', has domainCollection "' + file["domainCollection"] + '"'
        graql_insert_query += ";"
        graql_insert_query += " insert $newtask2 (outputfile: $output, inputfile: $input2) isa need-two-inputs;"
        graql_insert_query += " insert $newtask3 (inputfile: $input, outputfile: $output) isa produces;"
        graql_insert_query += " insert $newtask4 (inputfile: $input2, outputfile: $output) isa produces;"
    else:
        # insert contract
        graql_insert_query += " insert $newtask (outputfile: $output, inputfile: $input) isa requires;"
        graql_insert_query += ', has taskId "' + str(task["taskID"]) + '"'
        graql_insert_query += ', has name "' + task["name"] + '"'
        graql_insert_query += ', has functionsAndLibraries "' + file["functionsAndLibraries"] + '"'
        graql_insert_query += ', has manual "' + file["manual"] + '"'
        graql_insert_query += ', has commandLineParameters "' + file["commandLineParameters"] + '"'
        graql_insert_query += ', has reportUrl "' + file["reportUrl"] + '"'
        graql_insert_query += ', has reportName "' + file["reportName"] + '"'
        graql_insert_query += ', has domainCollection "' + file["domainCollection"] + '"'
        graql_insert_query += ";"
        graql_insert_query += " insert $newtask2 (inputfile: $input, outputfile: $output) isa produces;"
    return graql_insert_query
```

will get mad at you for having your JAVA_HOME variable pointing to a jre instead of a jdk. To fix that, simply update the variable in your Advanced System Settings – Environment Variables settings in Windows to wherever you have the jdk installed. If you already had Java, it is likely it is just a separate folder in your Java directory. With all this in place, and after following the steps described in the links above, you should be ready to run your code as a packaged jar file.

7.5 Methodology

This section constructs the workflow-based system description and solution for our capstone project.The following subsections capture the goals and user requirements for use of our system, describing the necessary workflows for each aspect of the solution design and implementation specifics.

7.5.1 User Personas and Goals

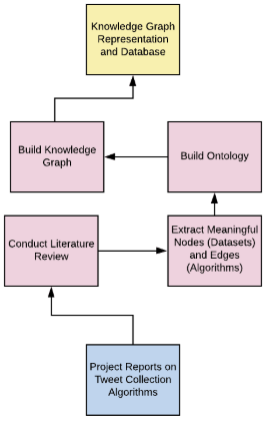
Our system, named the "Twitter-Based Knowledge Graph for Researchers," is built to support SMEs in non-computer science (CS) fields. SMEs are increasingly performing research involving advanced analysis, facilitated by a series of workflow management systems requiring significant background technical knowledge. The aim of our system is to "bridge the gap" between SMEs and workflow management systems, allowing users to query the information they need and generate the corresponding workflow. The goal of an SME user is to determine the workflow necessary to compute the desired computation (an algorithm), involving the necessary input files, output files, libraries, functions, and environments. The specific goals, each denoted by the query entered into the system, comprise a wide range of capabilities, from lemmatizing/cleaning tweets to clustering tweets with k-means. All the current possibilities for the system are stored in the ontology, which will be uploaded into the Grakn database to build a knowledge graph.

7.5.2 Tasks and Subtasks

Below are two instances of tasks necessary to complete the SME's goal of determining the necessary workflow for the given desired result.

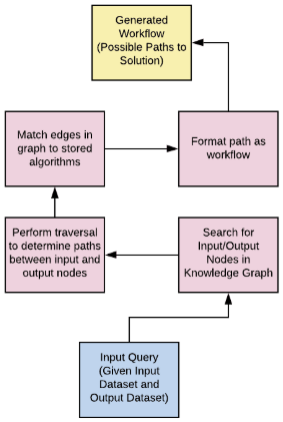
For the first task, the view is developer-centric, involving a literature review to construct the ontology and knowledge graph. A team of database experts (for our case, this was our project team) first collects reports on tweet collection algorithms and conducts a literature review. From this literature review, the team extracted meaningful nodes (or datasets) and edges (or algorithms) to build an ontology. This ontology was then used to build the knowledge graph representation and database, which can be used as the query service. The flowchart is shown in Figure 19.

Figure 19: Developer-Centric Workflow



The second task is user-centric, where the SME enters the goal into the query service to eventually output a workflow representation. This involves searching for input/output nodes in the knowledge graph, performing a graph traversal to determine possible paths between these nodes, matching graph edges to algorithms in the database, formatting the path as a workflow, and returning the generated workflow to the user.The flowchart is shown in Figure 20.

Figure 20: User-Centric Workflow



7.5.3 Implementation-Based Service

Each task has an associated implementation-based service. The user (SME) will enter a query for information, determined by a given input file and desired output file (or outcome). The system will then return the workflow necessary to achieve the goal. Below is the breakdown of our implementation-specific information for building the knowledge graph representation and database:

- Conduct Literature Review
  - Input File(s): 2016 and 2017 CS 5604
  - Information Retrieval Reports
  - Task Producing Input File: N/A
  - Output File(s): CS4624_2020_Ontology (Google Sheets)
  - Libraries/Functions/Environments: VTechWorks, Google Sheets
- Extract Meaningful Nodes (Datasets) and Edges (Algorithms)
  - Input File(s): CS4624_2020_Ontology (Google Sheets)
  - Task Producing Input File: Conduct Literature Review
  - Output File(s): nodes.csv, edges.csv
  - Libraries/Functions/Environments: Google Sheets, Microsoft Excel
- Build Ontology
  - Input File(s): nodes.csv, edges.csv
  - Task Producing Input File: Extract Meaningful Nodes (Datasets) and Edges (Algorithms)
  - Output File(s): Ontology in Grakn
  - Libraries/Functions/Environments: Grakn.AI
- Build Knowledge Graph
  - Input File(s): Ontology in Grakn
  - Task Producing Input File: Build Ontology
  - Output File(s): Knowledge Graph Representation/Database in Grakn
  - Libraries/Functions/Environments: Grakn.AI

Below is the breakdown of our implementation-specific information for querying the knowledge graph to generate a workflow to the solution.

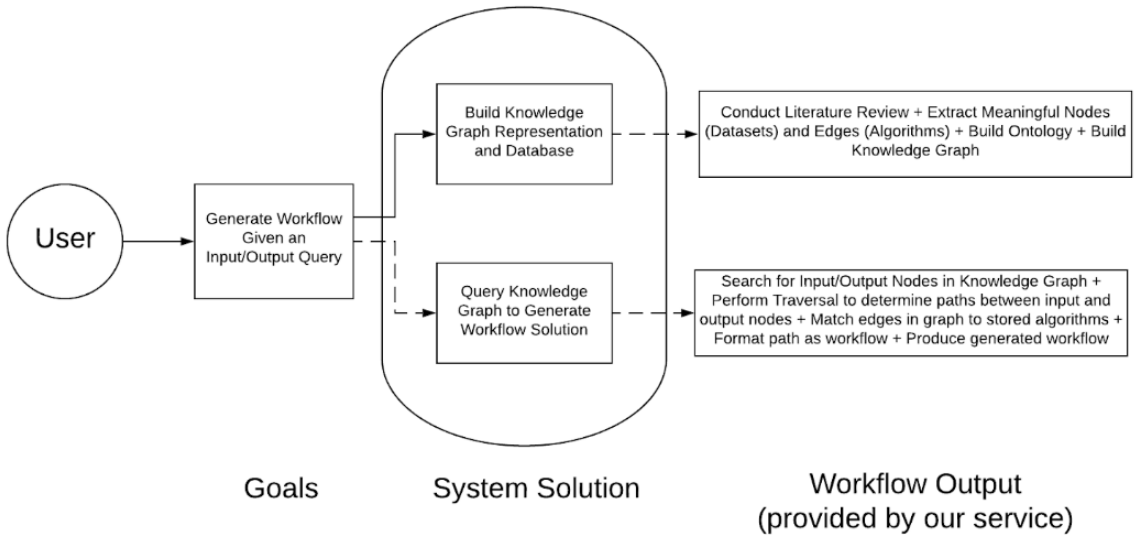
- Search for Input/Output Nodes in Knowledge Graph
  - Input File(s): Query (Input Dataset, Output Dataset)
  - Task Producing Input File: N/A
  - Output File(s): Corresponding Input/Output Nodes in Knowledge Graph/Database
  - Libraries/Functions/Environments: Grakn.AI
- Perform Traversal to determine paths between input and output nodes
  - Input File(s): Corresponding Input/Output Nodes in Knowledge Graph/Database
  - Task Producing Input File: Search for Input/Output Nodes in Knowledge Graph
  - Output File(s): Path(s) between queried input and output nodes
  - Libraries/Functions/Environments: Grakn.AI

- Match edges in graph to stored algorithms
  - Input File(s): Path(s) between queried input and output nodes
  - Task Producing Input File: Perform Traversal to determine paths between input and output nodes
  - Output File(s): Edges matched to algorithms in path
  - Libraries/Functions/Environments: Grakn.AI
- Format path as workflow
  - Input File(s): Edges matched to algorithms in path
  - Task Producing Input File: Match edges in graph to stored algorithms
  - Output File(s): Workflow of algorithms (tasks)
  - Libraries/Functions/Environments: Grakn.AI

7.5.4 Workflows

The aggregation of the methodology information and task/subtasks sequences discussed above is represented in Figure 21 as a list of workflows.

Figure 21: Workflow Representation of System Solution



7.6 File Inventory

Included in the master zip folder of files for this project are the following :

- schema.gql
- migrate.py
- migrateCSV.java
- pom.xml
- edges.csv
- nodes.csv

8 User's Manual

This section assumes that the user has either followed the steps in the Developer's Manual in this report to set up a Grakn Database instance, or has set up a similar instance on a different machine. As long as the user has access to the Grakn console, via running .\grakn console –keyspace knowledge_graph in the terminal, and they see the console prompt start up, they are good to go.

8.0.1 Sample Queries

The following figures show some of the queries that can be run on the data. The Graql language is very flexible, and this only represents a small subset of the types of queries that are available to the user. We recommend reading the documentation on Graql to get a grasp of the language available at `https://dev.grakn.ai/docs/query/overview`. However, it is possible to get a decent grasp on it just from looking at the following queries.

Figure 22: Some Simple Queries

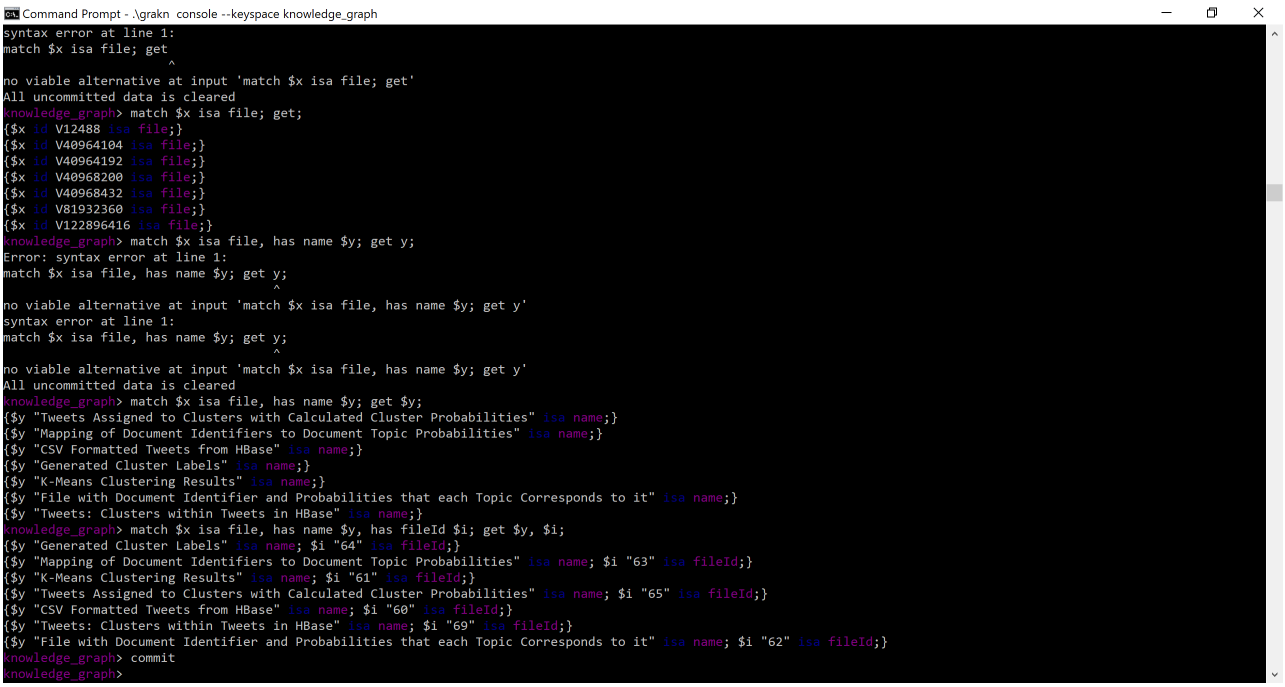


Figure 22 shows some basic "get" queries. The get query is primary query for retrieving information. It prints that information to the console. "get" can also be paired with "count" to return a count of the number of things returned in the form "get; count;". The first query in the figure, "match $ x isa file; get;" simply returns all of the files currently in the database. Recall that "file" is our primary entity object. Just seeing the id's of the objects doesn't tell us a whole lot though. After that query, you will see a query that didn't run correctly due to a syntax error. ATTENTION: If at any point you mistype a query, all uncommitted data will be cleared. If you are inserting data into the console, it is important to type "commit" to save that new data into the database so you don't accidentally lose it.

To get the names along with the files, enter "match $ x is a file, has name $ y; get $ y;"
To get the fileIds and names, enter "match $ x isa file, has name $ y, has fileId $ i, get $ y, $ i;"

Figure 23 shows two more queries of interest. Let's say we want to find the leafs in the graph. Luckily, in the schema.gql we remembered to add that entity type, so we can query for leaves the same way we query for files. Now that we have found a leaf, we want to find a workflow that gets us there. The second query in the figure returns all the nodes with a workflow relation to the leaf we found, where the leaf is the decendent.

Figure 23: Leaf and Workflow Queries

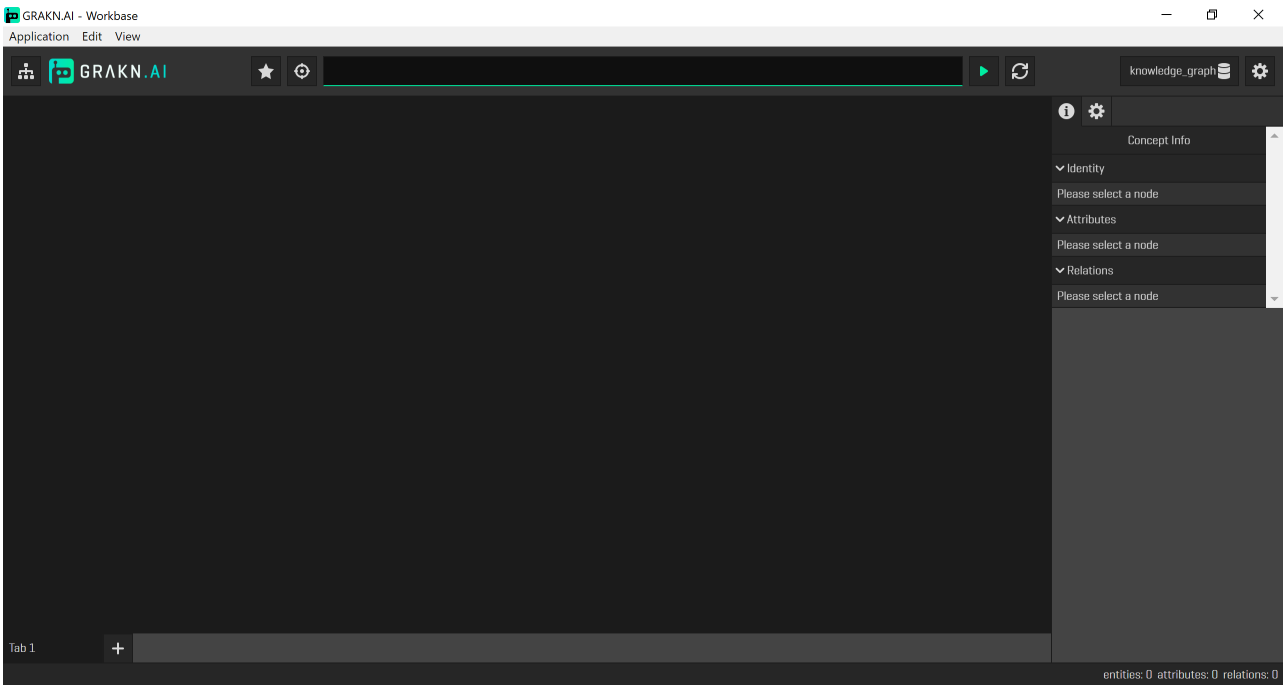
```
knowledge_graph> match $x isa leaf, has attribute $a; get;
{$a "69" isa fileId; $x id V122896416 isa file;}
{$a "Tweets: Clusters within Tweets in HBase" isa name; $x id V122896416 isa file;}
knowledge_graph> match $reach (ancestor: $x, decendent: $y) isa workflow; $y isa file, has name "Tweets: Clusters within Tweets in HBase"; get $reach, $x;
{$reach id V82022440 (ancestor: id V12488, decendent: id V122896416) isa workflow; $x id V12488 isa file;}
{$reach id V122970192 (ancestor: id V40968432, decendent: id V122896416) isa workflow; $x id V40968432 isa file;}
{$reach id V82018344 (ancestor: id V81932360, decendent: id V122896416) isa workflow; $x id V81932360 isa file;}
{$reach id V81998064 (ancestor: id V40968200, decendent: id V122896416) isa workflow; $x id V40968200 isa file;}
{$reach id V122941528 (ancestor: id V122896416, decendent: id V122896416) isa workflow; $x id V122896416 isa file;}
```

Now this only returns the nodes along that path, further development of the schema is needed to return the edges and nodes, as well as any alternate paths.

8.0.2 Workbase

When we set up this project originally, Grakn Core was in Version 1.7.0, and Grakn Workbase was in 1.2.7. As we were developing, we ran into a glaring issue that we could not visualize the data in Workbase. A day before report submission, we were informed by the people at Grakn that Workbase 1.2.7 is not compatible with Grakn 1.7.0, and that only earlier versions will work. If the User has a desire to use Workbase, ensure that the versions are compatible. If everything is installed correctly, and the correct keyspace is selected in the top-right, data should be visible in this pane. Our presentation will try to highlight what the data actually looks like in Workbase. Since there is little to no configuration required for the tool, and it uses the same query language as the console, the user should find it fairly intuitive.

Figure 24: Grakn Workbase, No Data



9 Lessons Learned

9.1 Timeline and Schedule

Our initial timeline plans involved setting a series of milestones over the course of the semester. These milestones are listed below and represented visually in Figure 25.

- Milestone 0 (1/27-1/31)
  - Project Approval from client (Prashant) and instructor (Dr. Edward A. Fox)
- Milestone 1 (2/03-2/07)
  - Study literature related to knowledge graphs. Look into how Facebook, Google, Uber, etc. use theirs
- Milestone 2 (2/13)
  - Presentation 1
- Milestone 3 (2/10-2/14)
  - Review past projects from CS5604 Information Retrieval
  - Build Twitter mining ontology
  - Start writing final report
  - Meet with client to discuss direction and progress
- Milestone 4 (2/17-2/21)
  - Establish query format for Network Science knowledge graph
  - Determine whether Network-based Clinical Trials graph is feasible
  - Look into Neo4J as interface option
  - Update report with new details and meet with client
- Milestone 5 (3/1-3/31)
  - Work on front-end of the knowledge graph
  - Understand how to extract workflows from the graph
  - 3/17 or 3/19 - Presentation 2. Assignment due 3/27.
- Milestone 6 (4/1-4/26)
  - Continue meeting with client to discuss updates and fixes
  - Repeat process for Twitter data graph
  - Record progress and finalize report
  - 4/17 Presentation 3 assignment due
  - 4/23, 4/28, 4/30 - Final presentation to class

There were two members for this project team, Emma Meno and Kyle Vincent. The division of project roles are illustrated in Table 5.

Figure 25: Initial Timeline

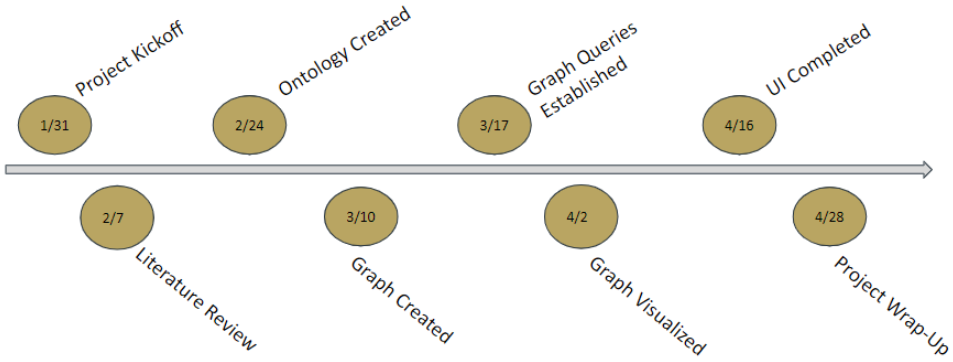


Table 5: Group Roles

Emma Meno	Kyle Vincent
Report Lead	Presentation Lead
Meeting Chair	Meeting Notetaker
Requirements and Design Lead	Prototyping, Implementation, and Testing Lead
Presentation Back-Up	Report Back-Up
Notetaker Back-Up	Meeting Chair Back-Up
Prototyping, Implementation, and Testing Backup	Requirements and Design Backup

9.2 Challenges Faced

This section addresses some of the challenges our project group faced throughout the semester.

9.2.1 Waiting for Data

Many software projects end up being delayed or slowed down by the lengthy process of acquiring data needed for the project. That was the case with this project. Originally, as seen in the timeline in Figure 25, our client was hoping to work with the Network Science project data. This data, however, was reliant on communication from a sister university. The data for the Network Science-Based Knowledge Graph did not arrive while still leaving feasible time to complete the project.

9.2.2 Gathering Sufficient Material from Project Reports

Initially, we were concerned whether the material from the project reports would be sufficient to build our ontology and knowledge graph. These reports were lengthy, but we were unsure if the specific algorithm and dataset information would be enough to build a full knowledge graph to the scale our client had hoped.

9.2.3 Selecting an Appropriate Project Toolkit

Our team had difficulties choosing a toolkit for our project. We kept encountering obstacles with different environments. Originally, we were planning to use Neo4J as an interface option for implementing our knowledge graph, Python as the backend to our graph, and Lucene for searching and indexing functionality. We were also planning to use LucidChart for some conceptual diagrams when outlining some of the algorithms and nodes. After thinking more about how we wanted to store and interface our knowledge graph data, we wanted to transition to a more established application stack. One of our team members researched and found the GRANDStack. At the time, we knew that we wanted to use Neo4j for our Graph Database but had a better idea of the front end and query language, which was GraphQL. React Javascript would be used

for the front-end UI. Apollo is a development tool that connects GraphQL workflows to the Neo4j Database. Neo4j does have a UI console to view your data, which we could have used as last resort, but we wanted to establish a more fleshed out application. However, after working more with the data, we found that the nodes fed by two or more edges of input (where the datasets output from multiple algorithms) were not easily modelled in the system. We still had to find a different project toolkit.

9.2.4 Transition to Virtual Learning

After the coronavirus outbreak, Virginia Tech transitioned to online learning. With social distancing guidelines, the project team members could no longer hold team in-person meetings, which were conducted weekly to bi-weekly during the first part of the semester.

9.3 Solutions

This section discusses the solutions to the challenges addressed in the previous section (in the same respective order).

9.3.1 Focus on Different Data

After the Network Science project data was still left unavailable to the client, we needed to pivot focus during the project. Instead, our project adapted to study the previous CS5604 Information Retrieval reports. Ultimately, this shift did not end up setting the team back too long. But this setback did serve as a reminder to never start a project unless you know the data is obtainable.

9.3.2 Include Reports from Multiple Years

We had a three-pronged approached to the challenge of whether the VT Libraries reports would be sufficient:

1. Pull workflows for knowledge graph from VT Libraries Database for CS 5604 Twitter projects
2. Investigate publications from the Digital Libraries Lab
3. Meet in-person with Subject Matter Experts to gather information on tool usage

Ultimately, we decided to include the CS5604 reports from multiple semesters: Fall 2016, Spring 2016, and Fall 2017.

9.3.3 Shift to Grakn

Our client recommended Grakn.AI as an alternative toolkit for us, which supported hypergraph functionality to model our "AND" problem. This transition was a bit difficult, since it involved an entirely different API, but the client worked diligently with us to help where he could.

9.3.4 Use of Zoom and Online Platforms

After the virtual learning transition, our team migrated meetings to be held over Zoom. We met multiple times during our in-class team discussions. We also scheduled meetings with our clients to discuss updates. Any other communications were done over Gmail and Google Docs. This transition was a bit difficult, especially since we were all at different locations with different schedules after cancellations and class changes, but we were able to navigate this shift fairly well.

9.4 Overall Lessons

This section discusses some of the lessons are project team learned throughout conducting the capstone work, not necessarily tied to any one challenge. These lessons also serve as recommendations for future teams' work.

9.4.1 Asking Questions

The start to any project involves the collection of requirements. We had many meetings with our client to understand the purpose and idea behind the project. While we asked many questions, we fell behind due to a lack of a complete understanding of the data granularity required and all the parameters that may be needed. As a result, the project team had to comb through reports multiple times in order to grab additional pieces of information that were deemed important. In future projects, the need to fully flesh out requirements in as few meetings as possible will be imperative.

9.4.2 Communicating with the Professor

Dr. Edward A. Fox, the instructor for this CS4624 capstone course, was very helpful any time that we contacted him about this project. He gave us helpful feedback for our interim report, suggesting edits incorporated throughout this report. Dr. Fox was also helpful throughout the team discussions held in class, dropping in to offer advice and suggestions. The team could have continued to reach out to him more for his advice throughout the project.

9.4.3 Using the Right Tool

We don't know if we can confidently say that at any point in this project we used the correct tool for job. Neo4j provided familiarity and ease of use. It was easy to manage and migrate data into. However, the stack that we would have had to build on top of that would have been difficult to configure within the scope of this project. In addition, RDF triples are limited in their flexibility and are unable to represent the full scope of the knowledge graph. Grakn presented it's own set of issues. The most glaring of these was the lack of documentation available for the tool. In developing our Grakn solution, we hit four different unsurpassable roadblocks that derailed our project that could have potentially been fixed if the creators of Grakn were more helpful in addressing these issues. Our hope is that Grakn continues to get better. However, the lesson learned here is to never commit to a tool unless it has very detailed documentation.

9.5 Future Work

The idea of a knowledge graph for subject matter experts is one that deserves attention. In the scope of this project, we developed an ontology to represent workflow data in a graph database, and explored a variety of application technologies. That being said, there is plenty more room for work to be done. Firstly, the solution needs to be developed on a virtual machine, so that a wide range of users can access it. Once the migration scripts are functional, the entirety of the data needs to be added to the database. The Graql schema needs to be updated to include relations and rules that will allow returning an entire path, along with all of it's information to the user via the console or Workbase, as well as additional rules to solve the "AND" issue described in this report.

Further exploration into Workbase is required, but that will come with later versions. To eliminate the learning curve that comes with learning the Grakn Query Language, development is needed to implement an API, or a front end interface that utilizes one of the clients. We believe that the Node.js interface could prove useful in this area, at least until Workbase is in better shape. New projects surrounding Twitter data will continue to pop up, and researchers will continue to ask more detailed questions. As a result, he data for the workflows will constantly be expanding. A stretch goal would be to automate the process of parsing through reports to generate the workflows used, as that was the most time consuming element to this project. In the future, it would be nice to see this knowledge graph be an incrementally improved project by a variety of teams from both the CS 4624 and CS 5604 capstones.

10 Acknowledgements

The project team would like to acknowledge our client Prashant Chandrasekar and our professor Dr. Edward A. Fox for their assistance in developing the requirements and vision of the project. We would especially like to thank Prashant for his time spent on the design and implementation of the project.

References

[1] https://d2908q01vomqb2.cloudfront.net/77de68daecd823babbb58edb1c8e14d7106e83bb/2019/01/10/Neptune-Metaphactory-1.png

[2] E. Williamson, S. Chakravarty, "CS5604 Fall 2016 Classification Team Final Report," VTechWorks, Dec. 8, 2016. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/73713

[3] M. Wagner, F. Abidi, S. Fan, "CS5604: Information and Storage Retrieval Fall 2016 - CMT (Collection Management Tweets)," VTechWorks, Dec. 8, 2016. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/73739

[4] S. Vishwasrao, S. Thorve, L. Tang, "CS5604: Clustering and Social Networks for IDEAL," VTechWorks, May 3, 2016. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/70947

[5] T. Dao, C. Wakeley, L. Weigang, "Collection Management Webpages - Fall 2016 CS5604," VTechWorks, Mar. 23, 2017. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/76675

[6] M. Bock, M. Cantrell, H. Shanin, "Classification Project in CS5604, Spring 2016," VTechWorks, May 4, 2016. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/70929

[7] S. Mehta, R. Vinayagam, "Topic Analysis project in CS5604, Spring 2016: Extracting Topics from Tweets and Webpages for IDEAL," VTechWorks, May 4, 2016. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/70933

[8] Y. Ma, D. Nan, "Collection Management for IDEAL," VTechWorks, May 4, 2016. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/70930

[9] T. Li, P. Nakate, Z. Song, "Collaborative Filtering for IDEAL," VTechWorks, May 4, 2016. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/70948

[10] A. Bartolome, MD Islam, S. Vundekode, "Clustering and Topic Analysis in CS5604 Information Retrieval Fall 2016," VTechWorks, May 8, 2016. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/73712

[11] M. Eagan, X. Liang, L. Michael, S. Patil, "Collection Management Webpages" VTechWorks, Dec. 25, 2017. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/81428

[12] A. Baghudana, A. Ahuja, P. Bellam, R. Chintha, P. Sambaturu, A. Malpani, S. Shetty, M. Yang, "CS5604 Fall 2017 Clustering and Topic Analysis" VTechWorks, Jan. 13, 2018. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/81761

[13] F. Khaghani, J. Zeng, M. Bhuiyan, A. Tabassum, P. Bandyopadhyay, "Collection Management Tweets Project Fall 2017" VTechWorks, Jan. 17, 2018. Accessed on: April 18, 2020. [Online]. Available https://vtechworks.lib.vt.edu/handle/10919/81996

[14] S. Klarman, "Modelling Data with Hypergraphs," Grakn.AI, Apr. 19, 2017. Accessed on: April 23, 2020. [Online]. Available https://blog.grakn.ai/modelling-data-with-hypergraphs-edff1e12edf0

[15] https://github.com/adriens/schemacrawler-plugin-neo4j

[16] "Getting Started With GRANDstack·GRANDstack." GRANDstack, grandstack.io/docs/getting-started-neo4j-graphql.html.

[17] "Grakn.AI – Text Mined Knowledge Graphs", https://www.stockwerk.co.at/event/grakn-ai-text-mined-knowledge-graphs/

[18] "Grakn Schema Overview", https://dev.grakn.ai/docs/schema/overview

[19] "Grakn Compute Query" https://dev.grakn.ai/docs/query/compute-query

[20] "Grakn System Requirements", https://discuss.grakn.ai/t/minimal-system-requirements/324