ENS 492 – Graduation Project

(Implementation)

Final Report

Project Title: Effective Labeling Micro-topic Relationships with Active Learning

Group Members: Barış Baştürk, Furkan Reha Tutaş, Kerem Akıllıoğlu, Tugay Garib

Supervisor(s): Prof. Yücel Saygın

Date: 17/05/2020

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

Online learning has grown exponentially during the first decade of the twenty-first century due to the rapid advancements in internet technologies. Many large colleges and universities began putting their course materials on the internet and making it publicly available to everyone interested in learning a particular subject. There are also many growing online encyclopedias, especially Wikipedia which is the largest and most popular general reference work on the World Wide Web with more than 300.000 active users and thousands of contributors from all around the world. The abundance of online technical material on the internet accelerated the process of reaching out to specific information. However, this simplicity has also brought certain disadvantages along with it. The major challenge for the learner is to be able to choose the right resources among a huge amount of online materials. The particular concept she/he is trying to learn may require some background knowledge, in that case, the learner should be able to identify the other concepts that are necessary to understand that particular concept and start studying them first. In this project, we are trying to eliminate this obstacle from users' life by forming reliable prerequisite relationships. In the work of Liang et al. [1], a prerequisite is something that must exist or happen before something else can exist or happen. In this project, a prerequisite defines a relationship among two concepts A and B. We define it as a score between [-1 and 1] where the sign indicates the direction of this relationship and the score gives us the strength of the prerequisite. Our work aims to build a complete weighted graph structure out of all the prerequisite scores between

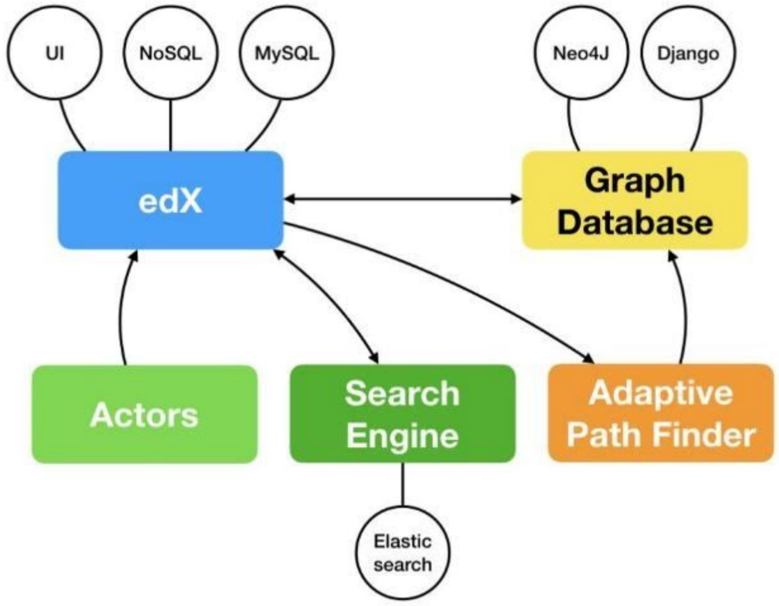
concept pairs. Thanks to this structure, a user will be able to understand the order of topics that she needs to study.

The second aim of this project is to create a user interface for an expert. The aim is to make the determination process of prerequisite relations for a given topic as smooth as possible. Given the goal, our graph database needs manual inputs initially. It is required for human experts to handle this process manually. Since the topics have a wide range (e.g from computer science to social sciences), the expertise of experts may vary from subject to subject. An important focus is to develop a user-friendly interface that can be used by anyone who has the basic skills required for operating a computer.

The major goal of this project is to develop different software algorithms that can automate the prerequisite determination process and reduce the number of manual inputs. The learning platform that we work on is based on micro-topics and it will have thousands of micro-subjects. Therefore, it would be very cumbersome to process inputs manually. In order to automate this process, we develop software algorithms that decrease the number of questions asked to the expert. This reduces the occurrence of possible human errors in the process and saves the expert's time. The algorithms decrease the number of questions asked from 500s to around 50s.

As a conclusion, we developed a software which includes different types of algorithms for detecting prerequisite relationships among microtopics, a tool to draw graphs of the given micro topics, script to get micro topics from Meta Academy and a user friendly graphical user interface.

2. PROBLEM STATEMENT



A graph database is aimed to be developed to store micro concepts. Graph databases are superior because a user can easily change the stored data or extend it by adding other attributes. In addition, graph databases are naturally indexed by relationship which makes it faster to run relationship based searches and access information compared to relational databases.

The graph database is directed and acyclic. All the incoming edges indicate prerequisite relationships. Each node represents a micro topic. Subnodes represent the sets of data stored for each micro topic such as videos, animations, slide presentations, reading materials, assessments etc.

There will be human experts who will use the software and an active learning algorithm that calculates weights to the prerequisite relationships between micro topics.

The algorithms developed in this project reduce the number of questions asked experts to around 0-50 using the weight provided by the active learning algorithm. This will reduce the total number of inputs needed from the human expert which will save time.

2.1 Objectives / Tasks

In the bigger picture, the main task of this project is to create a personalized e-learning system. For our project team, the aim is to build a user friendly interactive system that retrieves and displays information from a database in a rapid manner. From our system’s structural design point of view, a course can be described as a combination of several interconnected micro topics. We aim to maximize personalization by bringing relevant micro topics for every student. We start from determining the outline of the micro topic structures. Then, we focus on building a program to understand the connection between these micro topics. We build a graph database which stores all the related information about micro topics to increase the performance of our system. After storing the micro topic information, we try to understand the prerequisite relationships between the micro topics. Our model automatically assigns prerequisites to a given micro topic if there is enough information. If there is not enough information about prerequisite relationships, we try to understand the relationships between the courses using inference or by simply asking a human expert. The algorithm behind this program extracts prerequisite micro topics from the constructed graph. In order to understand these prerequisite relationships better, we add weights to prerequisite relations of micro topics in the graph database. Moreover, we add a feature which allows the expert to input percentages instead of 0 and 1 for prerequisite relations. After developing our program, we test the database for simple inputs.

Since this is an interactive system, we design a user friendly interface to increase usability. Users would expect to have an easy path to use a system where they can get quick responses. Our interface makes it easy to insert and remove elements from the database. Also, we try to build a consistent and robust system where users will not get surprised in terms of their expectations.

2.2 Realistic Constraints

The final software can be distributed by using a virtualization software like Docker to run on any operating system and can be scalable to support any number of users. The graph database will make the future queries faster once the prerequisites are determined by the expert and can be used concurrently. The caching mechanisms can be implemented in the server to increase the speed and decrease the load on the graph database.

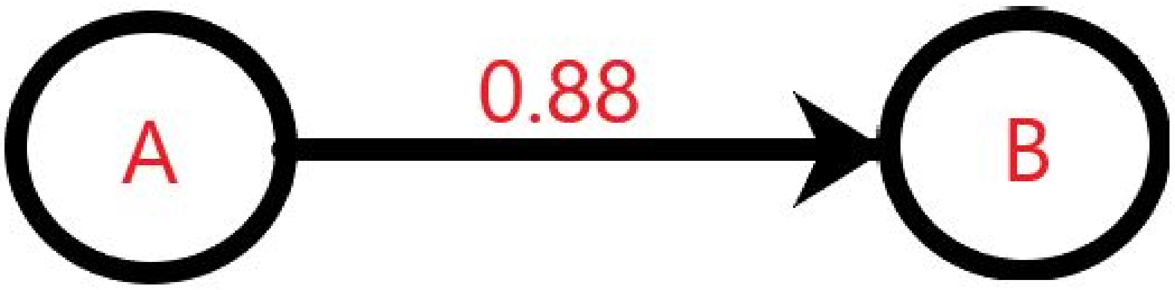
The experts will determine the prerequisites thus any ambiguity will be eliminated in the final graph. After constructing the graph, another mechanism to extract prerequisites from this graph can be used to make the queries much faster.

Performance is an important issue in this project because this software will be used by people who need to manually input prerequisite relationships. Micro topics are created by breaking big topics into many micro topics. Hence, there is an abundance of micro topics. As a result, the human experts job already becomes a time consuming one. The software must be quick to respond for it to be adopted by many people. Slow software might cause boredom and a waste of time. To surpass this constraint, the developed algorithms decrease the number of questions asked human experts significantly around 60-100% depending on the algorithm and the expected accuracy.

3. METHODOLOGY

3.1 Design of the Graph Database

The first technical development was to design a graphical database to represent both micro topics and prerequisite relationships among them. The following figure demonstrates an example of two micro topics, namely A and B, and a prerequisite relationship score between them.

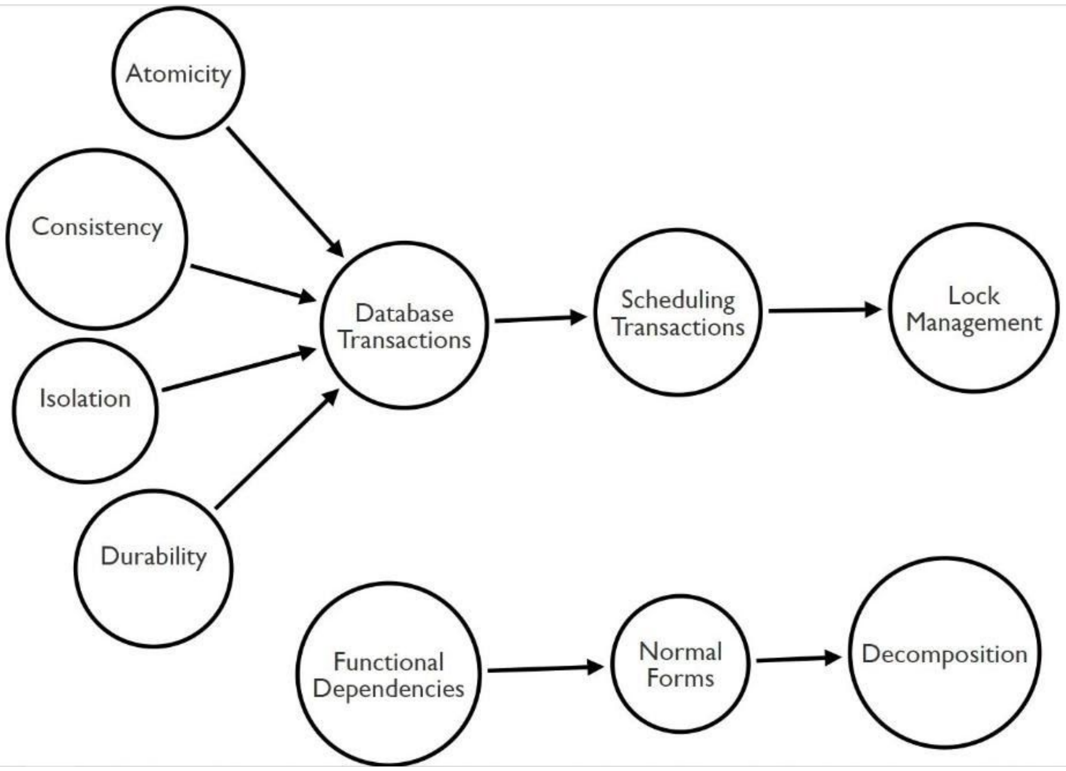


[Figure 1: Example Demonstration of Two Micro Topics with Prerequisite Relationship]

Two nodes represent micro topics A and B. The directed edge between A and B represents the prerequisite relationship with a score of 0.88 over a possible maximum score of 1. Additionally, the edge between A and B implicitly indicates another edge with an opposite directed edge has the score -0.88. However, it is sufficient to use either one of the scores, so it was decided to use only positive prerequisite relationship scores.

3.2 Brute Force Algorithm

The second technical development was to develop brute force algorithms with some heuristic approaches to find out all prerequisite relationships among given micro topics. The brute force algorithm asks all the possible prerequisite relationships to an expert to determine whether they have such relationships or not. With the aim of testing the algorithms that we developed, the following sample dataset with ideal prerequisite relationships is used.



[Figure 2: Sample Dataset with Database Systems Microtopics]

In the pure brute force algorithm, the expert will be asked P(N, 2) times in total, where N is the number of micro topics. For the sample dataset, it is equal to 90 questions.

3.2.1 First Heuristic

For given two micro topics A and B, if there is a prerequisite relationship in the direction of A→ B, it can be said that A is a prerequisite of B. Then, B cannot be a prerequisite of A because it contradicts the fact that A is a prerequisite of B. Therefore when a combination of possible prerequisite relationships is asked to the expert if expert marks the corresponding combination as a prerequisite, it is trivial to ask the opposite direction (e.g., A→ B marked as prerequisite then, B→ A is already known). Additionally, the next possible prerequisite to ask the expert is randomly selected.

Pseudo Code:

Define some sets: possible_pairs (all possible pairs of given micro topics), prerequiste_relationships (the empty set)

Define a notation: p1 → p2: p1 is a prerequisite of p2

```
1. Pick a random pair from possible_pairs and remove it from possible_pairs
2. Let’s call the picked pair as (p1, p2)
3. Check whether p1 → p2 or p2 → p1 is in or *implied by prerequiste_relationships
4. If it is the case go back to step 1.
5. Else, ask whether p1 → p2 holds
6. If it holds, add p1 → p2 to prerequiste_relationships, go to step 9
```

7. If it doesn't hold, ask whether p2→ p1 holds
8. If it holds, add p1 → p2 to prerequiste_relationships, if not go directly to step 9
9. Check whether possible_pairs is empty
10. If it is not empty go back to step 1
11. If it is empty, terminate and report prerequiste_relationships

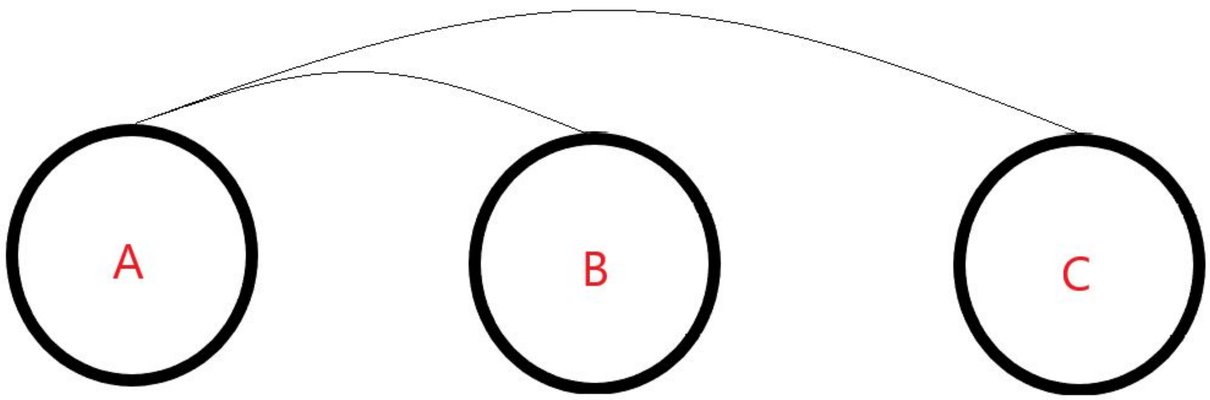
*Implied means that, the picked pair is not directly in the prerequiste_relationships list, but can be referred by using prerequiste_relationships list i.e., A→B and B-->C implies that A→ C

Our empirical results showed that it is good to have randomness (generalization) for experimental analysis of the algorithm.

According to our empirical results, the expert will be asked around 55 times over 100 iterations on average. It is 38% better than the pure brute force algorithm.

3.2.2 Second Heuristic

In the first heuristic approach, the next possible prerequisite relationship is chosen at random in each iteration. Another approach can be a slightly changed version of the first heuristic approach such that choosing a micro topic at random, and then ask the expert all possible prerequisite relationships from or to the chosen micro topic. Apply the same step until no micro topic is unvisited. The motivation to develop the second heuristic is to increase the proportion of pairs that are implied by the current prerequiste_relationships (Mentioned in 3.2.1). The following is an exemplary demonstration of a part of the algorithm.



[Figure 3: Example Demonstration of Brute Force Algorithm with Heuristic 2]

Assume that the algorithm randomly picks micro topic A and there are two other micro topics B and C that are not visited. Then the algorithm will ask the expert whether there is a prerequisite relationship between {A, B}, and {A, C}. After asking all possibilities which involve A, it will select another random micro topic, let's say B and so on.

Pseudo Code:

Define some sets: micro_topics (list of all micro topics), prerequiste_relationships (the empty set)

Define a notation: p1 → p2: p1 is a prerequisite of p2

```
1. Pick a random micro topic p from micro_topics and remove it from micro_topics
2. Let's call all pairs including p as p_pairs
3. Run First Heuristic Algorithm with p_pairs and append the result to
   prerequiste_relationships
4. Check whether micro_topics is empty
```

5. If it is not empty go back to step 1
6. If it is empty, terminate and report prerequiste_relationships

Experimentally, on average over 100 iterations, the expert will be asked around 55 times, which is 38% better than the pure brute force algorithm but the same as the first approach.

3.2.3 Third Heuristic

Assume that micro topics are sorted from advanced to basic topics. This can be done by calculating ingoing edges in an approximated graph of micro topics with weights. Instead of selecting the micro topic at random, sort micro topics from more advanced to basic topics, and start from the most advanced one to ask the expert. For example, in the sample dataset, the most advanced topic is Lock Management (See Figure 2) since it has the maximum number of prerequisites (same as the maximum number of ingoing edges to a node in a graph in which all prerequisite relationships explicitly are shown). The motivation to develop this algorithm is similar to the motivation of developing the second heuristic which is to increase the proportion of pairs that are implied by the current prerequiste_relationships (Mentioned in 3.2.1).

Pseudo Code:

Define some sets: sorted_micro_topics (list of all micro topics that sorted from basic to advanced), prerequiste_relationships (the empty set)

Define a notation: p1 → p2: p1 is a prerequisite of p2

1. Pick a random micro topic p from sorted_micro_topics and remove it from sorted_micro_topics
2. Let’s call all pairs including p as p_pairs
3. Run First Heuristic Algorithm with p_pairs and append the result to prerequiste_relationships
4. Check whether micro_topics is empty
5. If it is not empty go back to step 1
6. If it is empty, terminate and report prerequiste_relationships

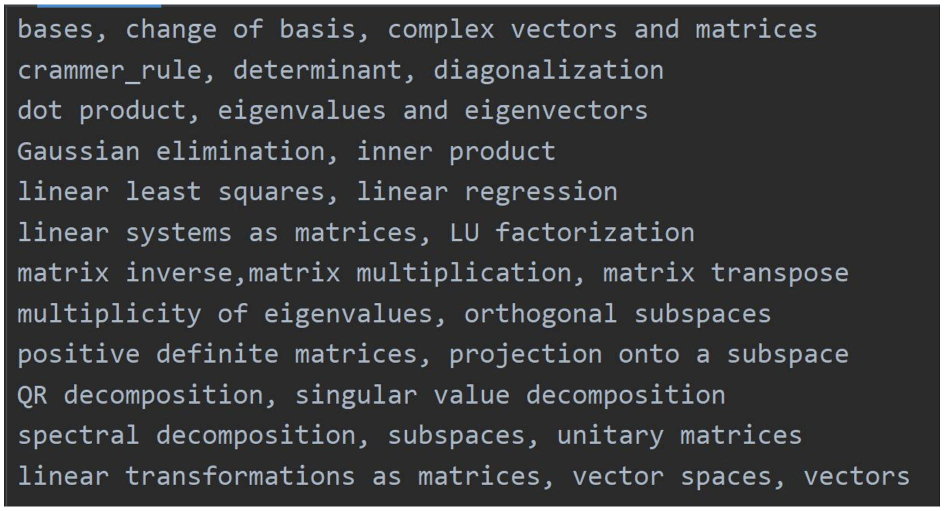
Experimentally, on average over 100 iterations, the expert will be asked exactly 48 times which is 46% better than the pure brute force algorithm and also 12% better than the first two heuristic approaches.

In conclusion, to find out all prerequisite relationships, the brute force algorithm with 3rd heuristic must ask an expert 48 number of questions, which is quite large for such a small number of micro topics.

3.3 Approximating Prerequisite Relationships with Weights

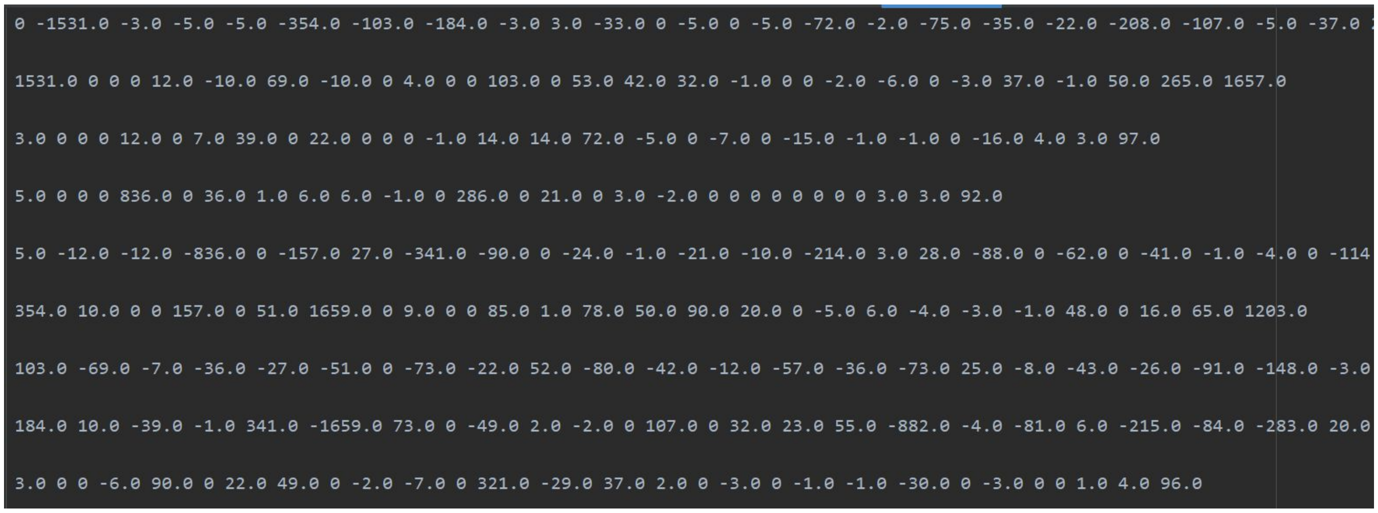
As indicated before, brute force algorithms with some heuristics ask too many questions to the expert. Therefore, there is a need for an algorithm that can reduce the number of questions asked to the expert dramatically.

Assume that there is a weight measure between all possible prerequisite relationships such that higher weight indicates a higher probability of being a prerequisite relationship and lower weight indicates the opposite. In order to test the approximating algorithm, the following 29 micro topics with given weights are used.



```
bases, change of basis, complex vectors and matrices
crammer_rule, determinant, diagonalization
dot product, eigenvalues and eigenvectors
Gaussian elimination, inner product
linear least squares, linear regression
linear systems as matrices, LU factorization
matrix inverse,matrix multiplication, matrix transpose
multiplicity of eigenvalues, orthogonal subspaces
positive definite matrices, projection onto a subspace
QR decomposition, singular value decomposition
spectral decomposition, subspaces, unitary matrices
linear transformations as matrices, vector spaces, vectors
```

[Figure 4: Sample Micro Topics]



```
0 -1531.0 -3.0 -5.0 -5.0 -354.0 -103.0 -184.0 -3.0 3.0 -33.0 0 -5.0 0 -5.0 -72.0 -2.0 -75.0 -35.0 -22.0 -208.0 -107.0 -5.0 -37.0

1531.0 0 0 0 12.0 -10.0 69.0 -10.0 0 4.0 0 0 103.0 0 53.0 42.0 32.0 -1.0 0 0 -2.0 -6.0 0 -3.0 37.0 -1.0 50.0 265.0 1657.0

3.0 0 0 0 12.0 0 7.0 39.0 0 22.0 0 0 0 -1.0 14.0 14.0 72.0 -5.0 0 -7.0 0 -15.0 -1.0 -1.0 0 -16.0 4.0 3.0 97.0

5.0 0 0 0 836.0 0 36.0 1.0 6.0 6.0 -1.0 0 286.0 0 21.0 0 3.0 -2.0 0 0 0 0 0 0 0 0 3.0 3.0 92.0

5.0 -12.0 -12.0 -836.0 0 -157.0 27.0 -341.0 -90.0 0 -24.0 -1.0 -21.0 -10.0 -214.0 3.0 28.0 -88.0 0 -62.0 0 -41.0 -1.0 -4.0 0 -114

354.0 10.0 0 0 157.0 0 51.0 1659.0 0 9.0 0 0 85.0 1.0 78.0 50.0 90.0 20.0 0 -5.0 6.0 -4.0 -3.0 -1.0 48.0 0 16.0 65.0 1203.0

103.0 -69.0 -7.0 -36.0 -27.0 -51.0 0 -73.0 -22.0 52.0 -80.0 -42.0 -12.0 -57.0 -36.0 -73.0 25.0 -8.0 -43.0 -26.0 -91.0 -148.0 -3.0

184.0 10.0 -39.0 -1.0 341.0 -1659.0 73.0 0 -49.0 2.0 -2.0 0 107.0 0 32.0 23.0 55.0 -882.0 -4.0 -81.0 6.0 -215.0 -84.0 -283.0 20.0

3.0 0 0 -6.0 90.0 0 22.0 49.0 0 -2.0 -7.0 0 321.0 -29.0 37.0 2.0 0 -3.0 0 -1.0 -1.0 -30.0 0 -3.0 0 0 1.0 4.0 96.0
```

[Figure 5: A Portion of Weights for All Possible Prerequisite Relationships]

The given weights are not normalized; therefore, they can take any real value.

For given weights and micro topics, the task is to develop an algorithm to approximate prerequisite relationships and reduce the number of questions asked to the expert. Firstly, negative prerequisite relationships can be ignored since they are just the opposite of positive relationships. Secondly, an upper and lower threshold can be determined such that all the weights bigger than the upper threshold are marked as a prerequisite relationship while all weights lower than the lower threshold are not marked. The weights between upper and lower thresholds are asked to the expert to be marked or not.

Pseudo Code:

Define some sets: sorted_micro_topics (list of all micro topics that sorted from basic to advanced), prerequiste_relationships (the empty set), positive_weights (list of all pairs such that have a positive weight between them)

Define a notation: p1 → p2: p1 is a prerequisite of p2

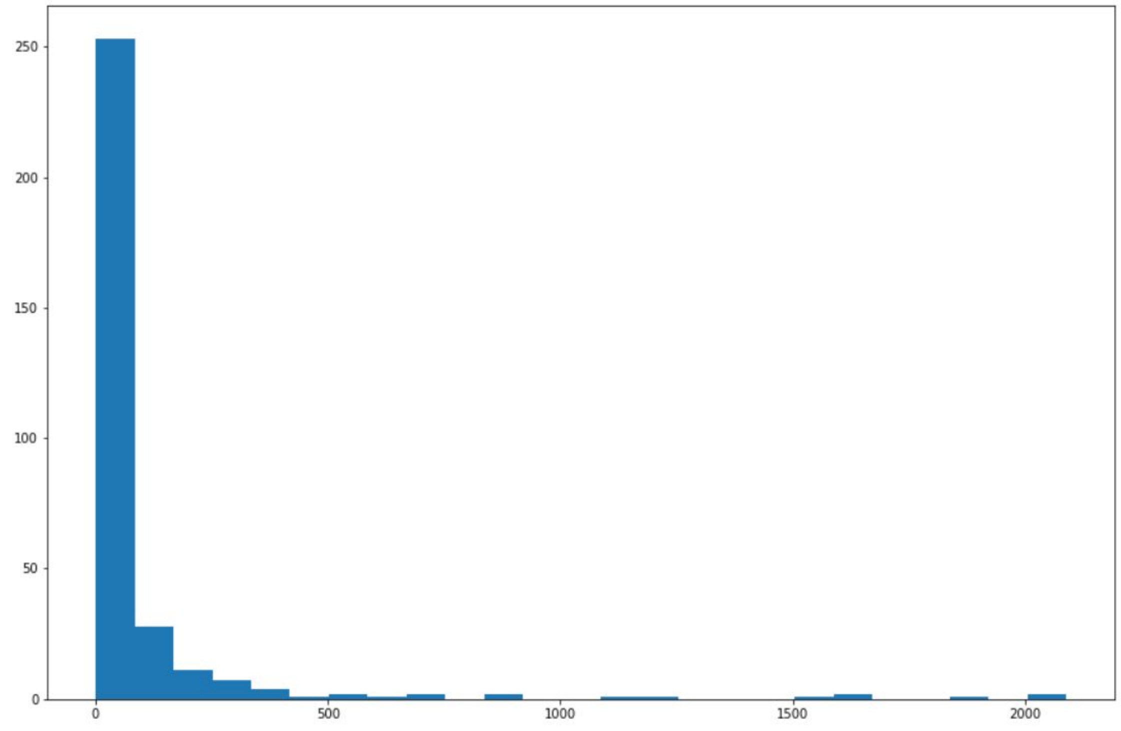
1. Get from user: lower_threshold and upper_threshold
2. Pick a random pair from positive_weights and remove it from positive_weights
3. Let’s call the picked pair as (p1, p2) and associated weight pair_weight
4. If pair_weight is smaller than lower_threshold go to step 9
5. Else if pair_weight is higher than upper_threshold, add p1→ p2 to prerequiste_relationships and go to step 9
6. Else, ask whether p1→ p2 holds
7. If holds, add p1→ p2 to prerequiste_relationships
8. If does not hold go to step 9
9. Check whether positive_weights is empty
10. If not empty go to step 1
11. If it is empty, terminate and report prerequiste_relationships

According to our experiments, the algorithm for thresholds [60, 100] provides 73% accuracy rate without asking any questions to the expert. The optimum threshold is around [10, 150] with 85% accuracy and 20 number of questions asked to the expert. In terms of best accuracy score with some optimization on the number of questions asked, [0, 240] provides 96% accuracy with 153 questions. If the prerequisite relationships are determined by the brute force algorithm with the third heuristic, the number of questions asked to the expert is 486.

As a result, the approximation algorithm with lower and upper thresholds provides a good ratio of accuracy rate while decreasing the number of questions radically, even the more flexible threshold [0, 240] provides 68% improvement in terms of the number of questions asked.

3.3.1 Histogram of Weights

The following histogram indicates the distribution of all positive weights in the dataset.

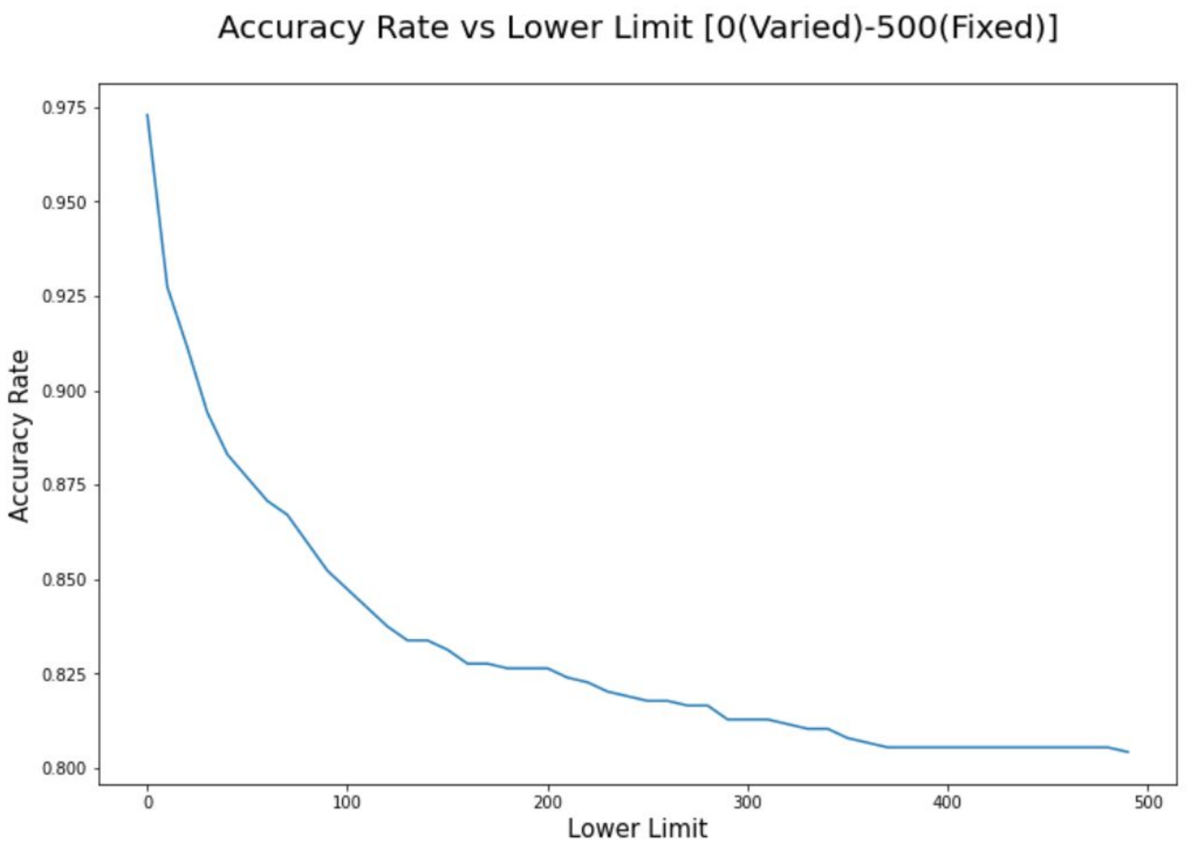


[Figure 6: Histogram of Positive Weights]

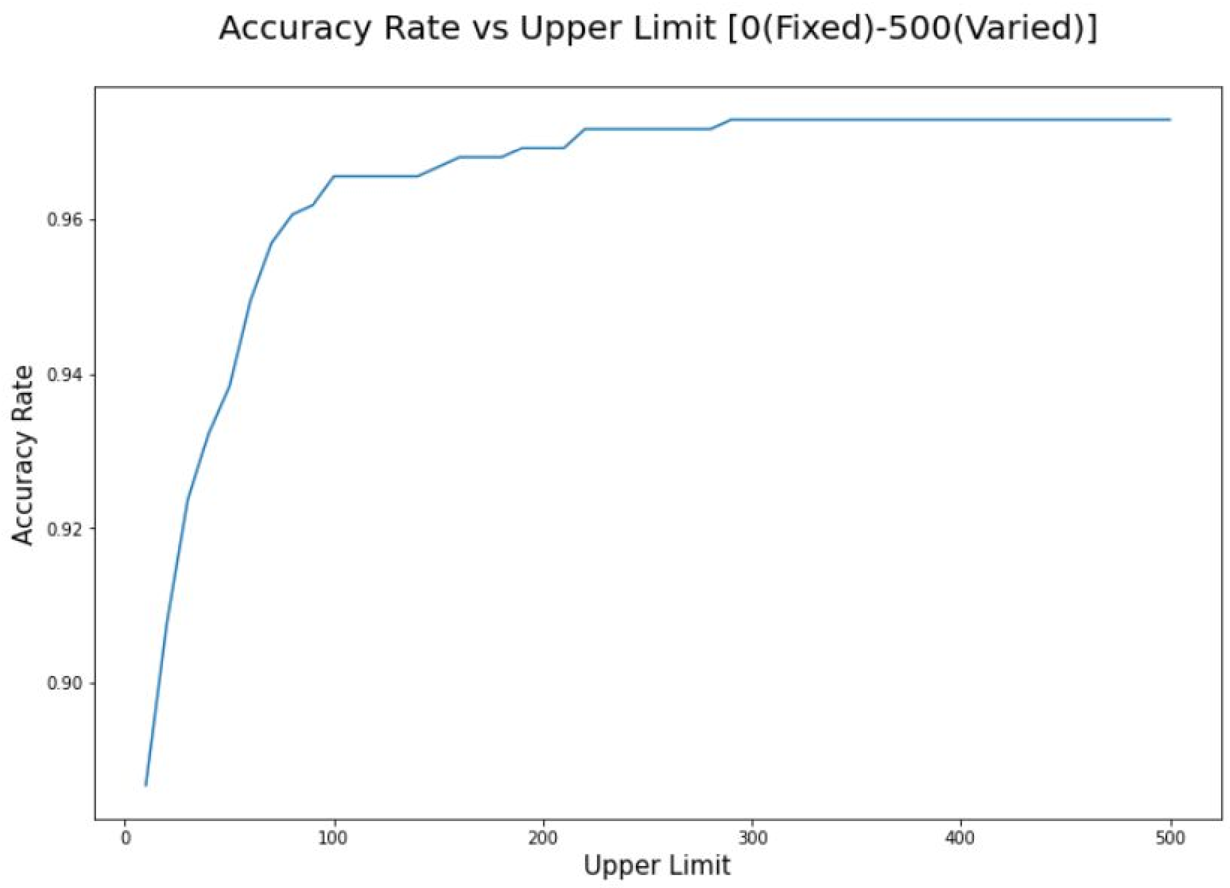
As the histogram indicates, the weights are right-skewed and the outliers are at the right handside of the distribution. It may point out that the choice of lower threshold is more important than the choice of upper threshold. In order to strengthen this claim further, some other graphs are also analyzed. Additionally, most of the weights occur in the region [0-500] as the histogram indicates. Therefore, this range is analyzed in the next graphs.

3.3.2 Accuracy Rate vs Lower and Upper Thresholds

The following graphs indicate accuracy rate by varying lower or upper threshold while keeping the other constant.



[Figure 7: Accuracy Rate vs Lower Threshold]

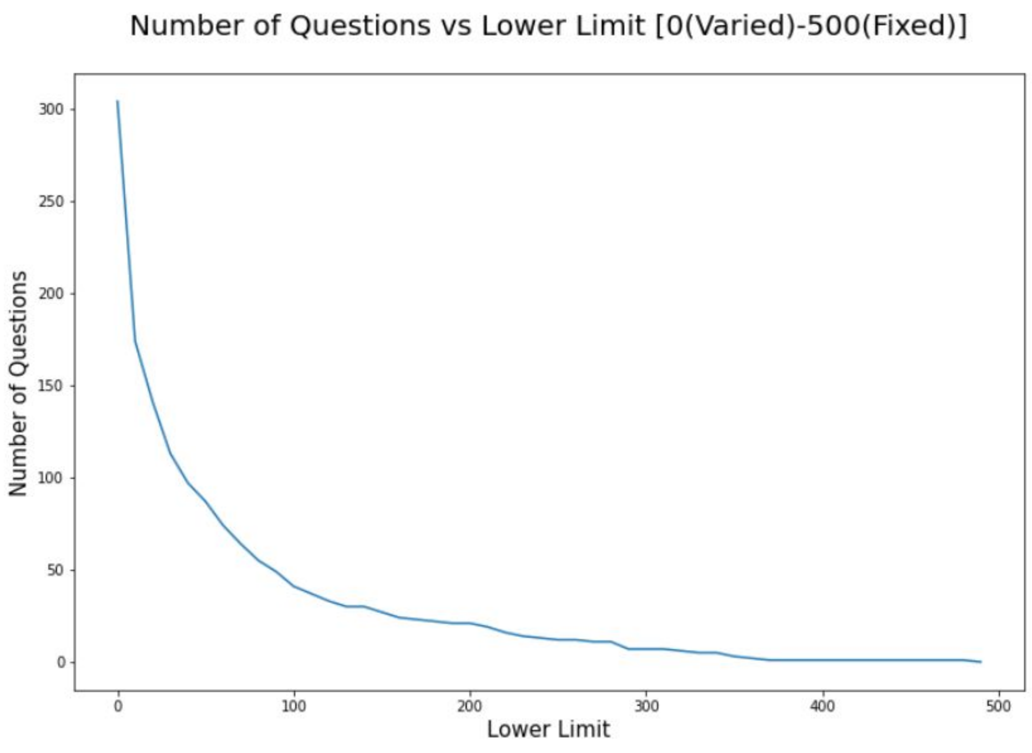


[Figure 8: Accuracy Rate vs Upper Threshold]

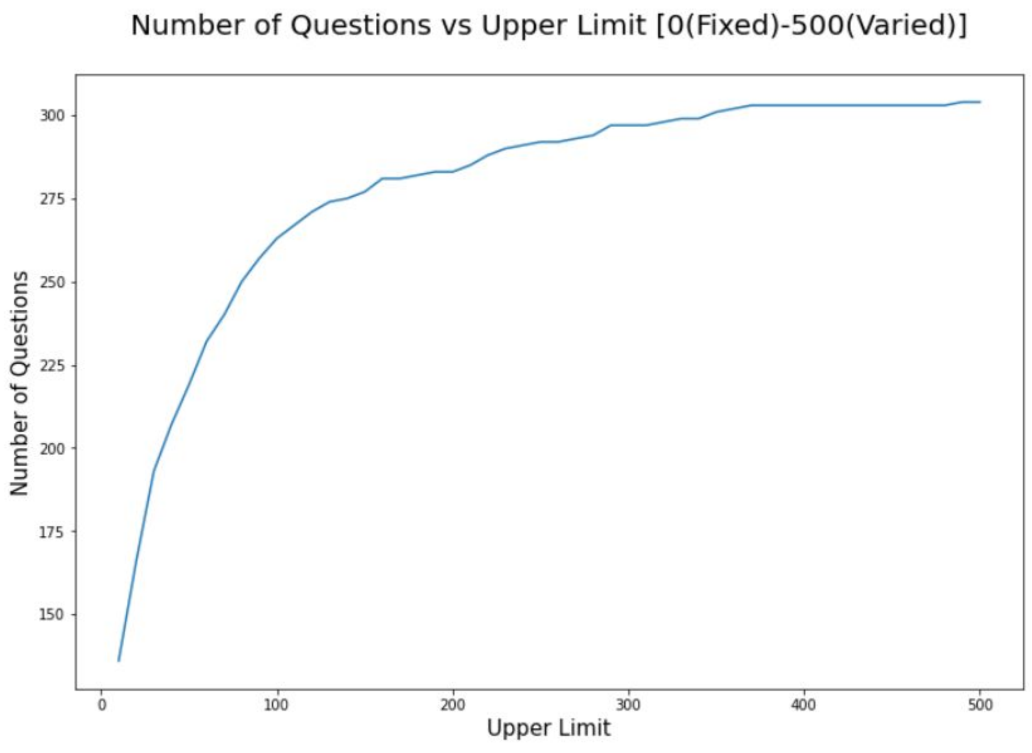
As Figure 7 and Figure 8 indicate, varying lower threshold has more effect on the accuracy rate (See above, varying upper threshold decreases accuracy up to 6% while varying lower threshold decreases accuracy up to 17.5%).

3.3.3 Number of Questions vs Lower and Upper Thresholds

The following graphs indicate the number of questions by varying lower or upper threshold while keeping the other constant.



[Figure 9: Number of Questions vs Lower Threshold]



[Figure 10: Number of Questions vs Upper Threshold]

As Figure 9 and Figure 10 indicate, varying lower threshold has more effect on the number of questions (See above, varying upper threshold decreases number of questions up to 150 while varying lower threshold decreases number of questions up to 300).

3.3.4 Binary Search Algorithm

As indicated in the parts 2.3.1, 2.3.2 and 2.3.3, a varying lower threshold is more effective on both the number of questions and accuracy. Therefore, in the adopted binary search algorithm, the upper threshold is fixed to some value (a threshold between 300-500 is the most efficient) and the algorithm applies binary search idea to lower bound.

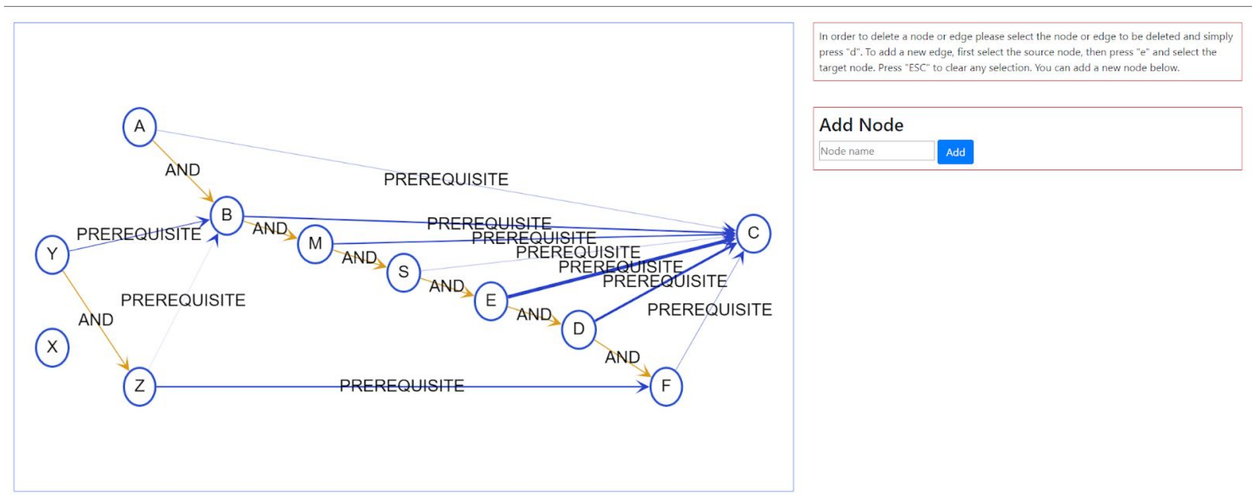
Pseudo Code:

1. Initially, all possible prerequisite relationships with positive weights are sorted.
2. The algorithm starts from the middle element of available weights and asks the current possible prerequisite relationship with two other possible prerequisite relationships around the chosen one to the user and majority voting applies.
3. If 2 or 3 of them are marked as prerequisite, then all the possible prerequisite relationships that have higher weights compared to middle weight are marked as 1
4. Otherwise, all the possible prerequisite relationships that have lower weights compared to middleweight are marked as 0.

5. This algorithm continues iteratively within the unmarked possible prerequisite relationships (the half in each iteration/unmarked ones) until there is no unmarked one or the user interrupts by limiting the number of steps.

This algorithm asks 6 questions with 83% accuracy which is approximately 3% better on average from the original algorithm described in part 2.3. It is not a wanted improvement, but with larger datasets, it can improve performance further.

3.4 Graph Visualization



In order to delete a node or edge please select the node or edge to be deleted and simply press "d". To add a new edge, first select the source node, then press "e" and select the target node. Press "ESC" to clear any selection. You can add a new node below.

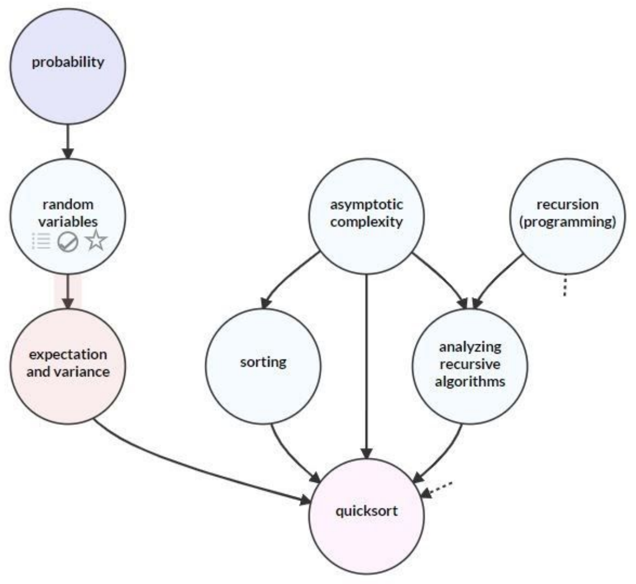
[Figure 11: Graph Visualization]

The end users should be able to see the final graph created by the algorithm. This is useful for various reasons for both end users and the experts that supervise the process. The experts can analyze the final outcome of the algorithm as a visual representation and can detect flaws if any exist. The visualization tool allows them to add or remove nodes and edges so that they can correct some easy to spot mistakes in the graph caused by the algorithm. Furthermore, they can remove some unnecessary edges to decrease unnecessary complexities to clear any confusion the end users might face. Similarly, the visualization tool is useful for the users because it enables them to view the prerequisites from a bigger perspective so that they can plan ahead their course of action when learning a subject/micro topic. The visualization of the graphs are done by a javascript library called cytoscape.js and various javascript functions are written to import and export graphs to the cytoscape.js. Since the standard format of cytoscape.js was not really useful for our purposes, the functions written accept an easier to understand and write format and convert these from and back to the cytoscape.js native format.

3.5 Data Generation

3.5.1 Web Scraper

For our project, one of our primary aims is to implement machine learning algorithms successfully. Therefore, we need data to train our machine learning models. However, it was not possible for us to find a dataset that represents prerequisite relationships. In order to satisfy our need for data, we implemented a web scraper and extracted data from https://metacademy.org/ website. Metacademy is an e-learning website where prerequisite relationships are clearly shown. Moreover, the way that they store data fits our project perfectly. An example can be seen below for the topic “quicksort”:



[Figure 12: Metacademy Graph Representation]

With the help of web scraper, our code visits the https://metacademy.org/ website and goes through all general categories. When the desired topic (e.g quicksort) is found, all prerequisites are extracted per concept and written to a text file.

It should be noted that our data still lacks weights between prerequisite relationships of concepts. Since we have micro topics and the corresponding prerequisite relationships that are somehow close to the ideal graph, we can develop/use an algorithm to calculate weights between micro topics and test the accuracy of our algorithm.

3.5.2 Weight Generation

In the development process, the most important limitation was calculating weights of micro topics. In the beginning, calculation of weights by active learning took days and it was not possible for us to generate sufficient data.

Our algorithm uses a deep learning library, pytorch, to conduct natural language processing and understand the relationships between micro topics. Firstly, the algorithm finds the corresponding concept’s YouTube videos and extracts the subtitles. Then it does the same operation for the concept that a prerequisite relationship is desired to be found. In the subtitle datasets, a comparative scoring mechanism works and eventually the algorithm generates weights between the micro topics. At the end, the algorithm computes an n x n matrix, where n is the total number of micro topics and filled values are the weight scores. Then we tried to optimize the algorithm and environment.

Our algorithm works on Google Colab because Colab provides 25GB RAM and we have more computational power. In addition, it is possible to load entire data into memory even for big data-sets . Using Google Colab speeds up the process around 2.5x time.

Then we noticed that we can optimize our algorithm by reducing computations. As explained we generate an n x n matrix but this matrix is symmetrical. Therefore we decided to run computations for just one half of the matrix and then paste the computed values to the other half, instead of computing the values for the entire matrix. In addition we do not run computations for the diagonal line because diagonal line refers to the concept’s prerequisite relationship with itself which is 0.

$$\begin{pmatrix} 0 & a & b \\ 0 & 0 & c \\ 0 & 0 & 0 \end{pmatrix}$$

[Figure 13: Sample Matrix that is Computed After Reduced Number of Operations]

$$\begin{pmatrix} 0 & a & b \\ a & 0 & c \\ b & c & 0 \end{pmatrix}$$

[Figure 14: Final Matrix that is Generated After Replacement with the Symmetric Values]

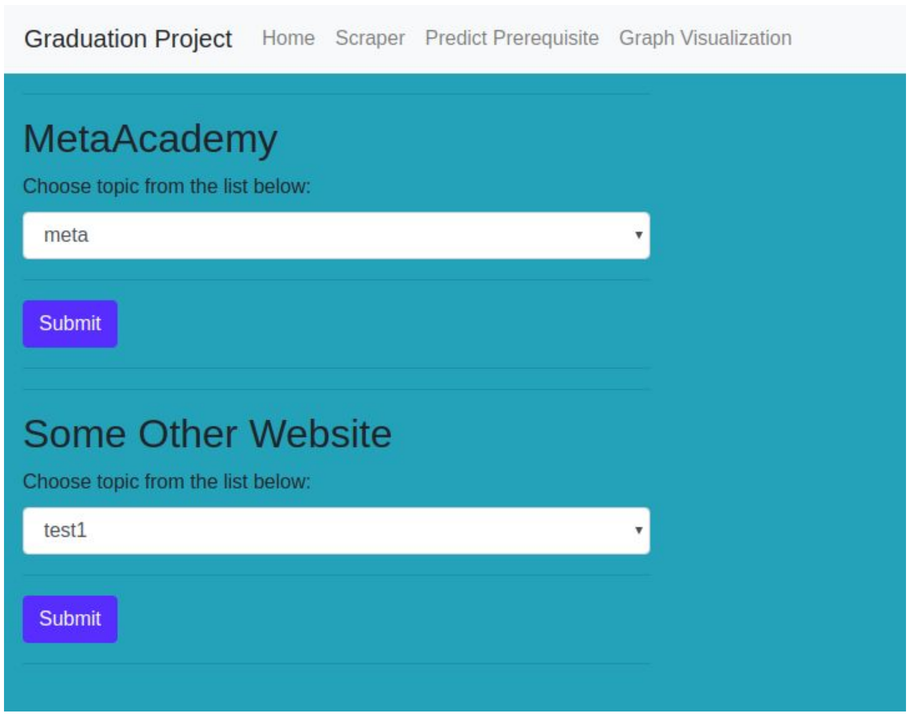
Our complexity was initially n^2, now it became $\frac{(n^2-n)}{2}$. Even though its complexity can be considered as n^2 still, the practical results were much more effective. Since the datasets are not that large to reflect this complexity, we noted that empirical runtime values are reduced by half. At this stage, it is not possible for us to make further improvements in the code because the operations itself are fairly complex and they take a lot of time due to natural language processing operations.

3.6 Graphical User Interface

3.6.1 Flask

We discussed as to which framework we should use for the development of the graphical user interface and after a bit of research and recommendations, we decided to use Flask. There are various reasons that affected our choice. First of all, it is lightweight. It is definitely an advantage because we didn't want to build a complex website with a lot of functionalities but rather we wanted to build a simple graphical user interface where users can interact with the algorithms we developed easily without having to deal with anything low-level and also able to visualize the end result in a web browser for further correction. Another advantage is that we already developed most of our algorithms and other related parts of the tool in python programming language so the integration was quite easy compared to other frameworks.

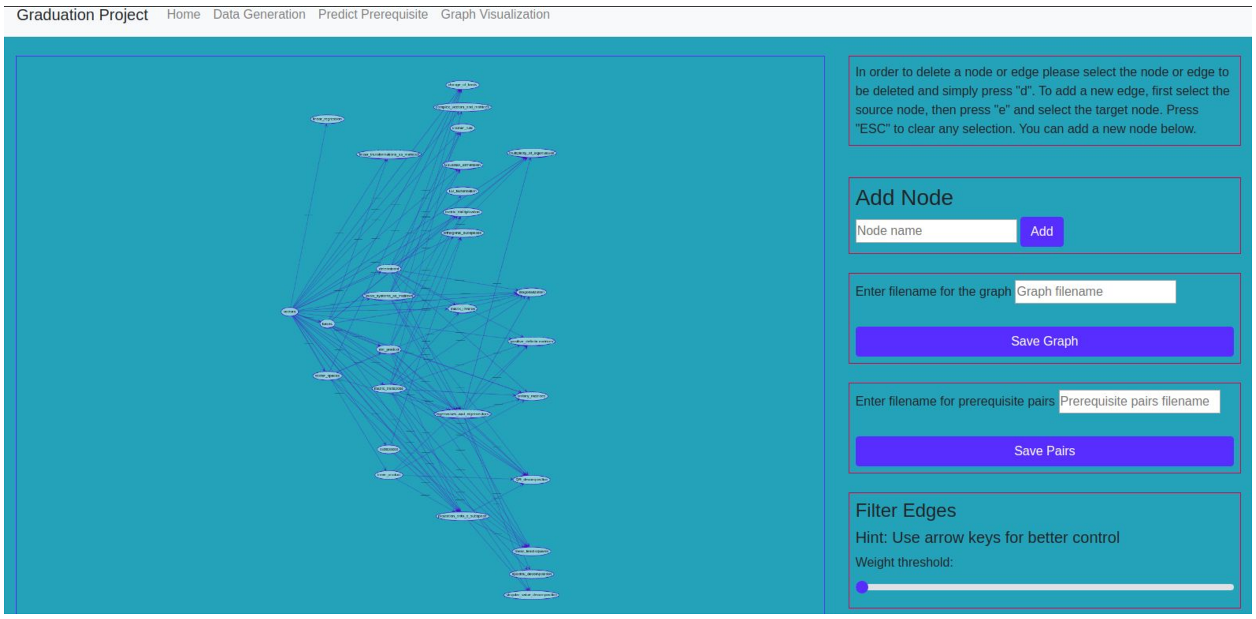
3.6.2 Scraper Integration



[Figure 15: Home Screen of the Graphical User Interface]

We were already given a simple scraper to scrape the data from a specific website (Meta Academy) and we updated the code so that we can use it directly from the GUI. Users can select any topic from the dropdown menu and will get the result as plaintext/html.

3.6.3 Visualization Integration



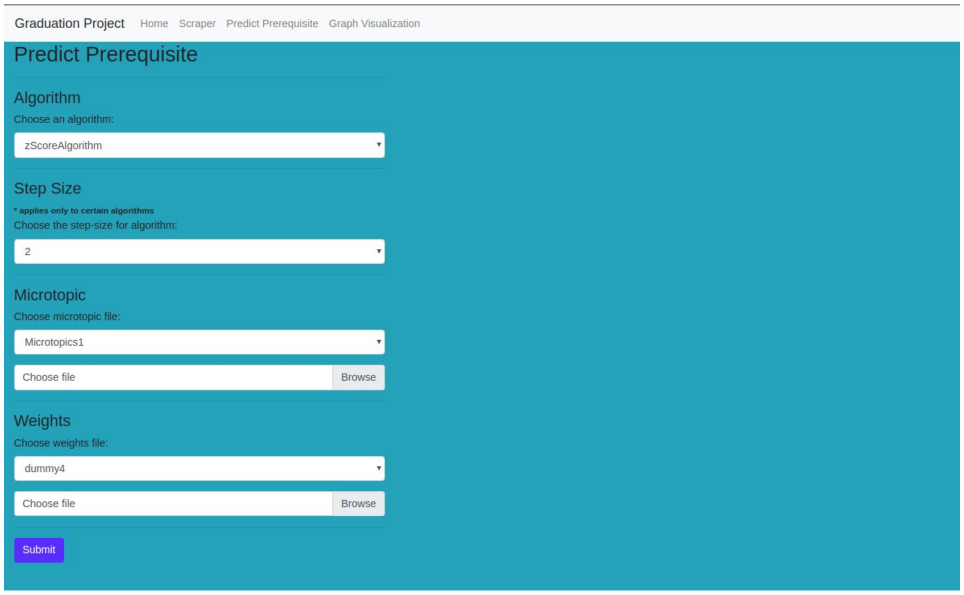
In order to delete a node or edge please select the node or edge to be deleted and simply press "d". To add a new edge, first select the source node, then press "e" and select the target node. Press "ESC" to clear any selection. You can add a new node below.

Add Node
Node name
Add
Enter filename for the graph
Graph filename
Save Graph
Enter filename for prerequisite pairs
Prerequisite pairs filename
Save Pairs
Filter Edges
Hint: Use arrow keys for better control
Weight threshold:

[Figure 16: Graph Representation Section of the Graphical User Interface]

The cytoscape.js library used to visualize the graphs and we integrated the code we developed into our Flask application. The graphs created by the algorithms are listed and users can select the resulting graph at any time to view and change it. This is especially useful when there are errors that are easy to spot to the human eye and experts can use the visualized graphs and make the necessary adjustments as they liked. The tool allows for any arbitrary change to the graph, in other words, you can add and delete prerequisites and new micro topics. Since the output of the algorithms are adjusted to be used with cytoscape.js library, we only store the output of the graphs in a text file (using a specific format) without having to deal with a graph database. After the algorithm creates the initial graph, experts can begin to modify it as they see necessary. After necessary modification is done by the expert, the resulting graph can be exported in two ways. First way to export graphs is to export the prerequisite pairs along with their weights to a file which is especially useful if the final result is to be revised at a later time because the output is simple and highly human readable. The second way to export graphs is useful when the expert (or other experts) wants to continue or revise changes made to the graph at a later time by using the same visualization tool. Experts can go to the visualization section and upload their exported graph to view it at any time. Another useful feature we added is to filter edges with respect to their weights. This is very useful since most of the prerequisites are found between 0.7 - 0.8 (normalized) and raw weights (algorithm is also implemented) can be visualized to quickly get a rough idea of the ideal graph. The filter is done by various javascript functions built on top of the core cytoscape.js library functions and the result is reflected in real time after the computations since any computation is done on the client-side (browser).

3.6.4 Integration of Algorithms



Predict Prerequisite

Algorithm
Choose an algorithm:
zScoreAlgorithm

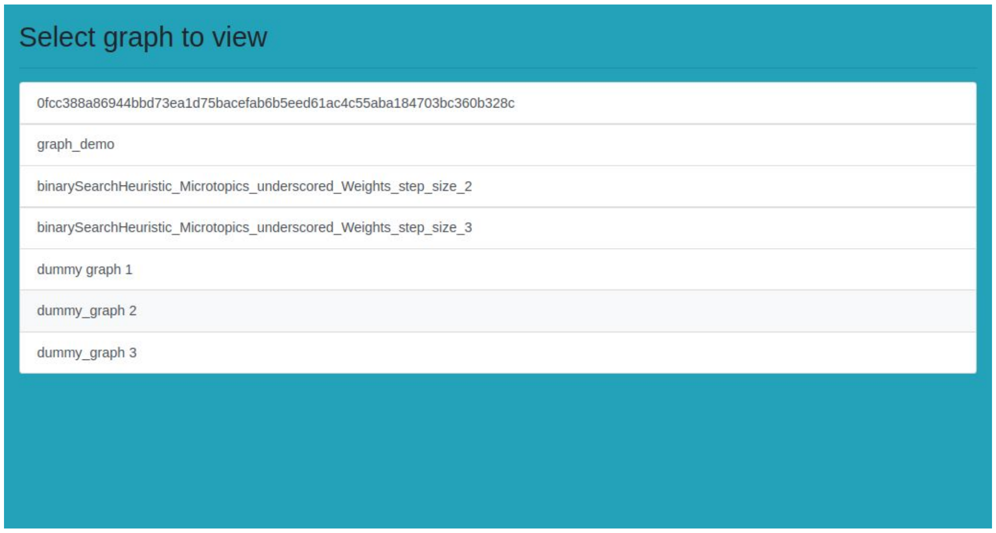
Step Size
* applies only to certain algorithms
Choose the step-size for algorithm:
2

Microtopic
Choose microtopic file:
Microtopics1
Choose file Browse

Weights
Choose weights file:
dummy4
Choose file Browse

[Figure 17: Algorithm Selection Section of the Graphical User Interface]

We come up with different algorithms to predict the prerequisite relation given weights. The GUI allows the user to easily choose or upload the related files (micro topic list and its weight file) and adjust some parameters such as step size after selecting the desired algorithm. The result is saved as a file and listed in the visualization part named with the parameters chosen. Sample list is below.



0fcc388a86944bbd73ea1d75bacefab6b5eed61ac4c55aba184703bc360b328c
graph_demo
binarySearchHeuristic_Microtopics_underscored_Weights_step_size_2
binarySearchHeuristic_Microtopics_underscored_Weights_step_size_3
dummy graph 1
dummy_graph 2
dummy_graph 3

[Figure 18: Graph Selection Section of the Graphical User Interface]

One problem we encountered during the integration of the algorithms to the Flask backend was that some of the algorithms were asking questions which affected the preceding questions that are asked and as a result we couldn't asked them all at once but rather we needed to implement a mechanism to pause the execution for user's answer and later resume it. The first solution we tried was to use a shared object which controlled the access to prevent any race condition. This goal was to spawn a new thread whenever the aforementioned kind of algorithm was selected by the user and share this object between the Flask thread and the spawned thread. The problem we encountered with this approach was that Flask wasn't thread safe and moreover sometimes it didn't run on a thread but rather spawned new processes. As a consequence, we changed our approach and used a text file to implement this shared object mechanism. The race condition problem, no matter how improbable, may arise in this case since there is no access control for the file thus once in production it can simply be solved with the help of a database to regulate access.

4. RESULTS & DISCUSSION

The initial aim of the project was to develop an independent tool for experts to use for creating a prerequisite graph with minimum effort so that the e-learning platform can later make use of this information. Overall, the objective is achieved since we developed a graphical user interface with the needed functionalities for experts to use. However, there are still incomplete small parts that either need to be developed from scratch or improved further. For instance, the calculation of weights is done with a google colab script but since there is no proper way to run this script from the Flask backend (or any other framework for that matter), the experts still need to use an outside resource to achieve the initial objective. Although it seems not possible to connect google colab at the moment, we are sure that google will enable developers to connect and run scripts in the future. Until the e-learning platform is fully developed along with the tool we developed, we have no means to know whether the project will contribute to the current state-of-art e-learning platforms.

Since all of the developed algorithms are heuristics, and in each step the performance is improved, it is best to give results and discussions for each algorithm, at the end of the corresponding algorithm to emphasize on the development process. Therefore, you can find results and discussions for each algorithm in section 3.

5. IMPACT

Investigating prerequisite relationships is a research area that is tackled from different aspects and approaches. However, it is hard to evaluate new approaches due to lack of existing datasets with ground truth prerequisite relationships. Since most papers use low-quality datasets and their approach depends on questionable datasets, the end result is not reliable. As this tool gets developed further, it can be used to generate reliable data and overcome this problem in the long run.

6.ETHICAL ISSUES

This project does not include any physical production process (i.e. assembly line production etc) so there is no possibility of harming the environment. All the code/software/algorithms written are the project group’s work, well known classic algorithms and are developed using original versions of software so there are no issues with software copyright.

As a result there are no ethical issues in this project.

7.PROJECT MANAGEMENT

Although the initial plan for the implementation of the project didn’t change tremendously, we had to make a few adjustments to the plan. For instance, we were planning to use machine learning algorithms to further improve our accuracy but due to technical problems we encountered during the implementation such as the time it takes for current algorithms to generate data delayed this improvement. We should’ve started to work on the data generation earlier so that now we would’ve had enough data to apply machine learning algorithms. Although it caused a delay, if we manage to generate enough data, we are still planning to use machine learning. Apart from this delay, any changes to the initial plan is not worth mentioning and only constitutes minor changes.

8.CONCLUSION AND FUTURE WORK

As a conclusion, we developed a software which includes different types of algorithms for detecting prerequisite relationships among microtopics, a tool to draw graphs of the given micro topics for visualization, script to get micro topics from Meta Academy and a user friendly graphical user interface.

In the development process, the most important limitation is about calculating weights of micro topics. Since the calculation of weights by the active learning algorithms takes days, there is no sufficient data at the moment, therefore datas to test algorithms are not sufficient. Since in the future there will be enough data (in terms of weights), machine learning algorithms can be developed and also the developed algorithms can be tested in more detail. Additionally, the developed software has no direct connection with the active learning algorithm that calculates the weight score among micro topics, the connection can also be established.

Since this is only a part of the e-learning project, in the future all components will be needed to be integrated and tested if they are compatible, robust, etc.

REFERENCES

[1] - Liang, C., Ye, J., Wang, S., Pursel, B., & Giles, C. L. (2018). Investigating active learning for concept prerequisite learning. In 32nd AAAI Conference on Artificial Intelligence, AAAI 2018 (pp. 7913-7919). (32nd AAAI Conference on Artificial Intelligence, AAAI 2018). AAAI press.