ENS 492 – Graduation Project

Effective Labeling Micro-topic Relationships with Active Learning

Supervisor: Prof. Yücel Saygın

Kerem Akıllıoğlu *24179*
Tugay Garib *23450*
Barış Baştürk *21020*
Furkan Reha Tutaş *21036*

Outline

1. Problem Statement (1)
Introduction

Brief Overview

- The **abundance of online technical material** on the internet accelerated the process of reaching out to specific information
- 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
- 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
- Our aim is to **automate the process** of identifying prerequisites as much as and as accurately as possible

I

1. Problem Statement (2)
Introduction

Graph database

- A storage for micro concepts;
- Incoming edge refers to prerequisite relationships;
- Directed and acyclic;
- A node refers to a micro topic;
- A subnode refers to sets of data for each node (videos, animations, etc.)

1. Problem Statement (3)
Objectives / Tasks

Objectives:

- Automatically assigning prerequisites
- Construction of a user friendly interactive system
- Creation of a personalized e-learning system

Tasks:

1. Building prerequisite relationships between the micro topics
   a. Sufficient information $\longrightarrow$ automatic allocation of prerequisites by our program
   b. Insufficient information $\longrightarrow$ ask to a human expert
2. Designing a user friendly interface with Flask
3. Integrating scraper into user interface
4. Integration of algorithms

1. Problem Statement (4)
Constraints

Realistic constraints:

- Abundance of micro topics
- Slow software

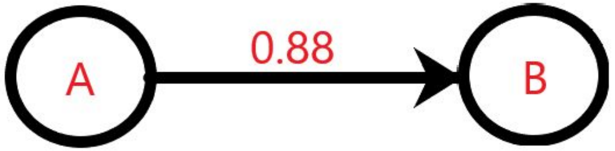
Solution:

- 60-100% less questions asked to a human expert with the assistance of developed algorithm.
- The graph database will make the future queries faster once the prerequisites are determined by the expert and can be used concurrently.

2. Design of the Graph Database (1)
Relationships of Micro topics

Representation of Relationships:

- The following figure demonstrates an example of two micro topics, namely A and B, and a prerequisite relationship score, *i.e. weight*, between them.



Input Format of Data:

- A text file where names of the micro topics are listed line by line
- An additional text file to just represent the weights of the micro topics.
    - In the format of a matrix: **rows and columns** refer to micro topics, **elements** refer to weights

2. Design of the Graph Database (2)
Setting Up The System

Building the Database:

- Subject micro topic names are listed line by line and provided in a **text file**
- While going through the file, a **node** in the graph database is created for every each micro topic
- Through the heuristics and algorithms, we decide **whether** we should **establish a connection between nodes by creating an edge**
- Regarding to the strength of prerequisite relationships, **weight** score is calculated and added to database

neo4j

Neo4j is selected as the graph database management system

I

3. Data Generation (1)
Brief Overview

What Data do we need?

- Data of **course subjects,** as micro topics, **and** their **prerequisite relationship** with each other
- **Datasets are needed to form prerequisite relationships.**

Why do we need Data?

- To make our program more **flexible**
- In order to implement the **machine learning models**
- Datasets are needed to form prerequisite relationships.

How did we generate?

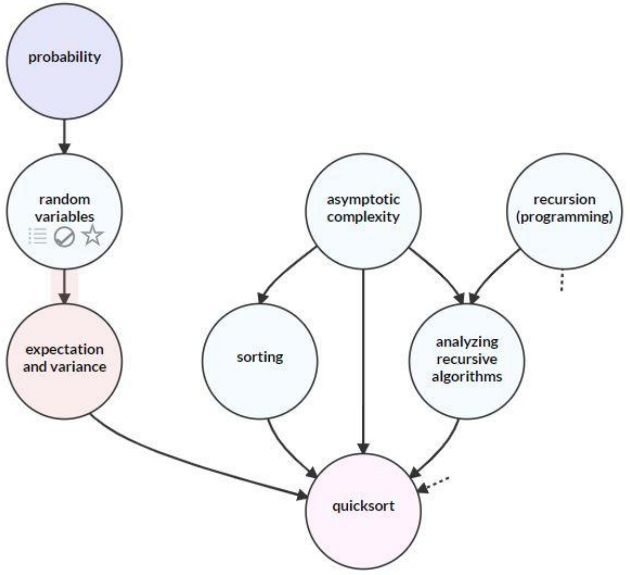
- Through a **web scraper** we **extracted** data
- Through the **weight generation tool**, we **computed weights** and combined everything altogether.

3. Data Generation (2)
Web Scraper

Metacademy is an e-learning website with sorted prerequisite relationships, was used for data extraction. Through a step-by-step approach:

1. Web scraper visits the *metacademy.org* website
2. Scraper goes through all general categories
3. When the desired topic *(e.g quicksort)* is found
4. All prerequisites are extracted for every each concept and written to a text file.

Note: It should be noted that our data still lacks weights between prerequisite relationships of concepts



Graph Representation of Topic Quicksort by Metacademy.org

3. Data Generation (3)
Weight Generation

Motivation and Methodology:

- Used a **weight generation** tool to add weights for the raw “microtopics data” we have
- Weight generation algorithm **iterates over microconcepts**, checks other concepts’ relevancy by conducting natural language processing (**NLP**) and computes the weights
- Although the tool is **accurate** for generating weights, it is **slow** due to NLP operations

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**.

3. Data Generation (4)
Further Improvements

Reducing the Number of Computations

- At the end, symmetrical n x n matrix is generated
- Instead of computing weights for the entire matrix, we computed weights for the upper triangle only
- We copied the computed values to the lower triangle
- We did compute weights for the diagonal line, *i.e concept's prerequisite relationship with itself is 0*

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

Impact

- We decreased the number of computations by more than than half and in parallel we noted that empirical **runtime** values are **reduced by half for weight generation**.

4. Heuristics (1)

Brute Force Algorithm and the First Heuristic

- The brute force algorithm asks the expert all possible prerequisite relationships.

- The first heuristic includes:
  - Choose the next possible prerequisite relationship to ask at random.
  - If A → B holds then don't ask whether B → A holds.
  - If A → B and B → C then don't ask whether A → C and C → A hold.

4. Heuristics (2)
The Second Heuristic

- Slightly changed version of the first heuristic.
- Choose a micro topic at random, and then **ask the expert** all possible prerequisite relationships from or to the chosen micro topic.
- **Motivation**: **increase** the proportion of **pairs that are implied** by the current extracted prerequisite relationships such as **(A → B) and (B → C) implies (A → C)**.
- For example, let's say A, B, C and D are micro topics:
  - Choose A at random; ask the expert (A,B), (A,C), (A,D) pairs
  - Choose D at random; ask the expert (D,B), (D,C)
  - ...

4. Heuristics (3)
The Third Heuristic

- **Sort micro topics from advanced to basic** by calculating ingoing edges in an approximated graph of micro topics with weights
- Instead of selecting the micro topic at random; **start from the most advanced one to ask the expert**
- Motivation: increase the proportion of pairs that are implied by the current extracted prerequisite relationships such as **(A → B)** and **(B → C) implies (A → C)**
- **Best performing heuristic**

5. Algorithms (1)

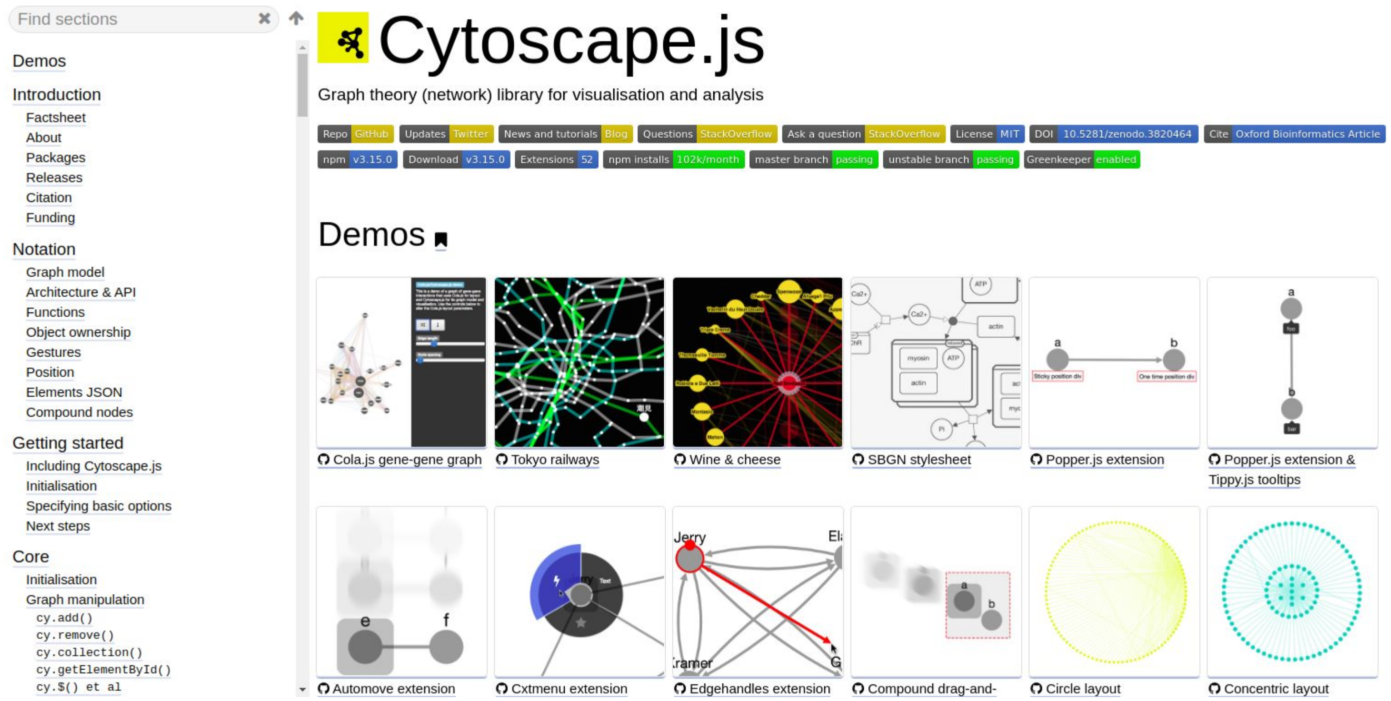
Approximating Prerequisite Relationships with Weights

- **Determine lower and upper weight thresholds**
- All pairs that have **higher weight than upper threshold** are **directly accepted as prerequisite**
- All pairs that have **lower weight than lower threshold** are **directly accepted non-prerequisite**
- Ask the expert all the rest
- Results:
    - **Varying lower threshold has more effect** on the number of questions and accuracy rate
    - **Binary Search to determine lower threshold**

5. Algorithms (2)
Binary Search

- **Upper threshold is fixed**
- **Lower threshold** is determined by binary search:
  - **Sort pairs** w.r.t weight scores.
  - **Choose middle element** and 2 neighbour pairs, do majority voting by asking the expert
  - **Recursively do the same** until there is no pair to determine or it reaches the limit number of steps determined by the expert

6. Graph Visualization (1)
Cytoscape.js



Find sections

Demos

Introduction
- Factsheet
- About
- Packages
- Releases
- Citation
- Funding

Notation
- Graph model
- Architecture & API
- Functions
- Object ownership
- Gestures
- Position
- Elements JSON
- Compound nodes

Getting started
- Including Cytoscape.js
- Initialisation
- Specifying basic options
- Next steps

Core
- Initialisation
- Graph manipulation
  - `cy.add()`
  - `cy.remove()`
  - `cy.collection()`
  - `cy.getElementById()`
  - `cy.$() et al`

Cytoscape.js

Graph theory (network) library for visualisation and analysis

Repo GitHub | Updates Twitter | News and tutorials Blog | Questions StackOverflow | Ask a question StackOverflow | License MIT | DOI 10.5281/zenodo.3820464 | Cite Oxford Bioinformatics Article

npm v3.15.0 | Download v3.15.0 | Extensions 52 | npm installs 102k/month | master branch passing | unstable branch passing | Greenkeeper enabled

Demos

Cola.js gene-gene graph | Tokyo railways | Wine & cheese | SBGN stylesheet | Popper.js extension | Popper.js extension & Tippy.js tooltips

Automove extension | Cxtmenu extension | Edgehandles extension | Compound drag-and- | Circle layout | Concentric layout

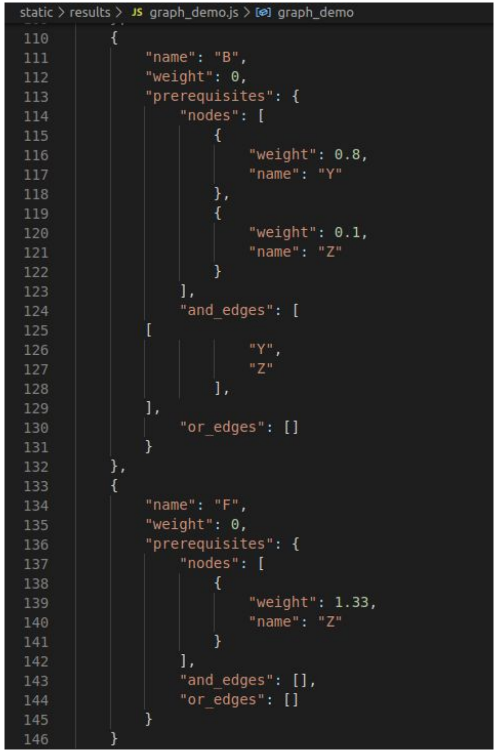
6. Graph Visualization (2)
Cytoscape.js

- **Cytoscape** is a complete software package written in Java for network/graph visualization to be used in a desktop environment.

- **Cytoscape.js** is a **JavaScript library**
  - has a great **documentation** https://js.cytoscape.org/
  - **easily extendable** by various plugins
  - **graphs can be customized** in various ways in terms of visual representation
  - **allows to modify graph** (e.g. add/delete nodes and edges), **changes reflected immediately**

6. Graph Visualization (3)

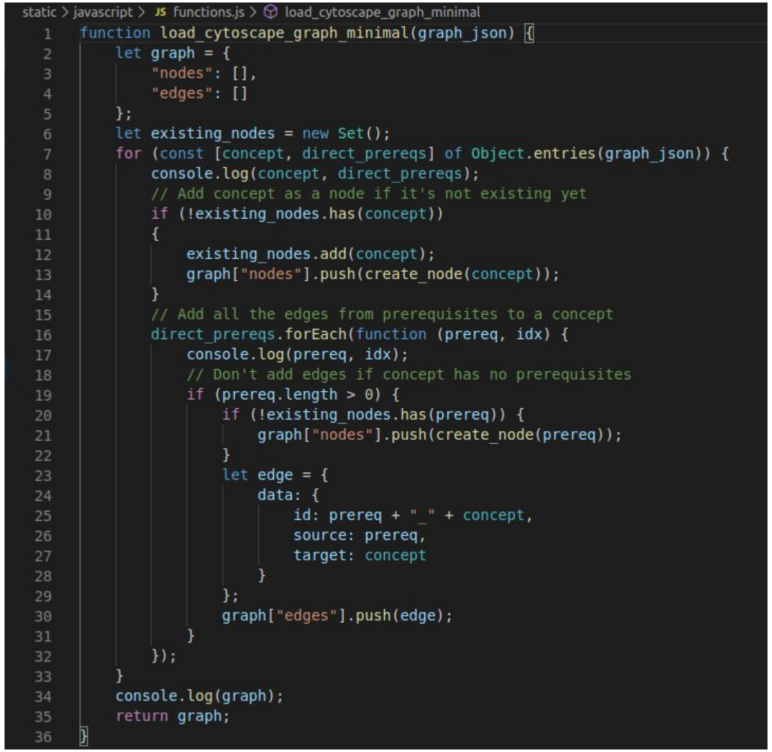
Cytoscape.js - loading graph from custom JSON format



static > results > JS graph_demo.js > graph_demo

```
    {
        "name": "B",
        "weight": 0,
        "prerequisites": {
            "nodes": [
                {
                    "weight": 0.8,
                    "name": "Y"
                },
                {
                    "weight": 0.1,
                    "name": "Z"
                }
            ],
            "and_edges": [
        [
                    "Y",
                    "Z"
                ],
        ],
            "or_edges": []
        }
    },
    {
        "name": "F",
        "weight": 0,
        "prerequisites": {
            "nodes": [
                {
                    "weight": 1.33,
                    "name": "Z"
                }
            ],
            "and_edges": [],
            "or_edges": []
        }
    }
```

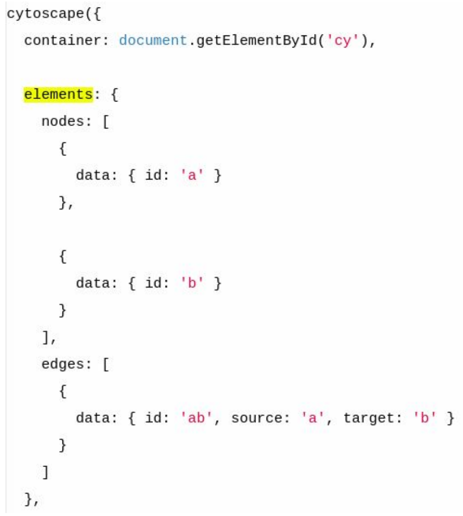
Custom JSON (Array) Format



static > javascript > JS functions.js > load_cytoscape_graph_minimal

```
function load_cytoscape_graph_minimal(graph_json) {
    let graph = {
        "nodes": [],
        "edges": []
    };
    let existing_nodes = new Set();
    for (const [concept, direct_prereqs] of Object.entries(graph_json)) {
        console.log(concept, direct_prereqs);
        // Add concept as a node if it's not existing yet
        if (!existing_nodes.has(concept))
        {
            existing_nodes.add(concept);
            graph["nodes"].push(create_node(concept));
        }
        // Add all the edges from prerequisites to a concept
        direct_prereqs.forEach(function (prereq, idx) {
            console.log(prereq, idx);
            // Don't add edges if concept has no prerequisites
            if (prereq.length > 0) {
                if (!existing_nodes.has(prereq)) {
                    graph["nodes"].push(create_node(prereq));
                }
                let edge = {
                    data: {
                        id: prereq + "_" + concept,
                        source: prereq,
                        target: concept
                    }
                };
                graph["edges"].push(edge);
            }
        });
    }
    console.log(graph);
    return graph;
}
```

Transform Custom JSON into native Cytoscape.js



```
cytoscape({
  container: document.getElementById('cy'),

  elements: {
    nodes: [
      {
        data: { id: 'a' }
      },

      {
        data: { id: 'b' }
      }
    ],
    edges: [
      {
        data: { id: 'ab', source: 'a', target: 'b' }
      }
    ]
  },
```

Native Cytoscape.js JSON Elements

6. Graph Visualization (4)
Cytoscape.js - Extending Graph Controls

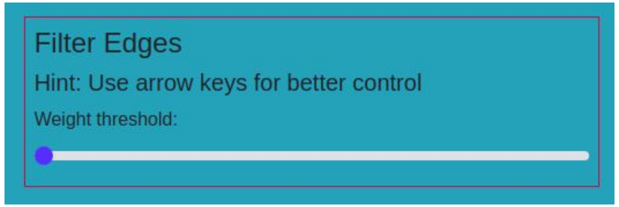
- **Cytoscape.js** implements basic functionality as a core functionality to add or delete nodes and edges to an existing graph
- We extended this core functionality so that the **interaction with the graph would be more intuitive, faster and easier for the expert.**

Expert can

- **Add nodes** and specify the weight of the node to be added (default 1)
- **Delete a selected node** (highlighted with yellow after selection for visual feedback)
- **Add a new edge** by first selecting the source and then pressing ‘e’ key to specify target, weight of the new edge is by default 1
- **Delete selected edge** (highlighted with red after selection)

6. Graph Visualization (5)
Cytoscape.js - Extending Graph Controls (2)

- **Cytoscape.js** allow to filter edges based on attribute, we extended this feature to remove (and restore) edges based on a weight threshold adjustable by a slider
- **Step-size for slider is crucial** in order for expert to be able to reflect any possible change to the graph, therefore the slider is designed to be dynamic and is usable
- On initialization, *i.e. after the graph loaded*, the weights are processed and minimal distance between the weights is set as the step-size.
- Similarly, **min and max attribute for slider is set accordingly**


Filter Edges

Hint: Use arrow keys for better control

Weight threshold:

Example

- **Weights-1 :** 1 2 3 4 5 6 7
- **Weights-2 :** 10 20 30 40 50
- **Weights-3 :** 0.1 0.2 0.5 0.8 0.9

A different threshold needed for each case

7. Graphical User Interface (1)
Flask - Motivation

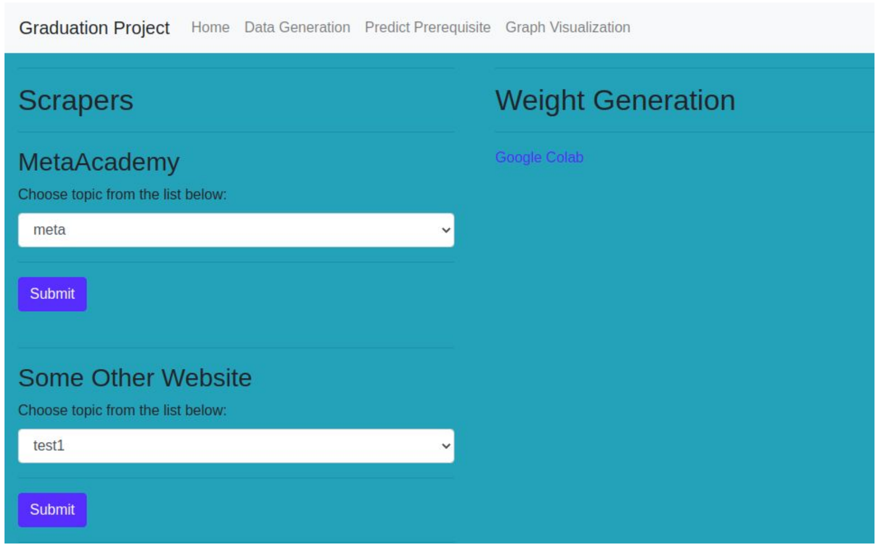
- We needed a **GUI** for experts to interact with the various parts that we developed
- Flask is a lightweight, micro web framework that has everything we needed
- Since the algorithms are implemented in Python, integration is significantly easier compare to other frameworks
- Development is much faster compare to other python web frameworks (e.g. Django)
  - Flask only implements the core functionalities of a web framework and thus the syntax is much simpler and faster in both development and production

7. Graphical User Interface (2)
Flask - Overview

There are 3 main pages:

- **Data Generation**
  - Meta Academy scraper
  - Weight Generation (colab)
- **Predict Prerequisite (using algorithms)**
  - Algorithms (with parameter options as dropdown menus)
- **Graph Visualization**
  - Visualization of the final graphs for expert to modify and inspect further
  - Filter edges with respect to their weights with a slider
  - Export or import the modified graph
  - Save prerequisite pairs to text file (human readable format)

7. Graphical User Interface (3)
Flask - Data Generation



Scrapers

MetaAcademy

Choose topic from the list below:

meta

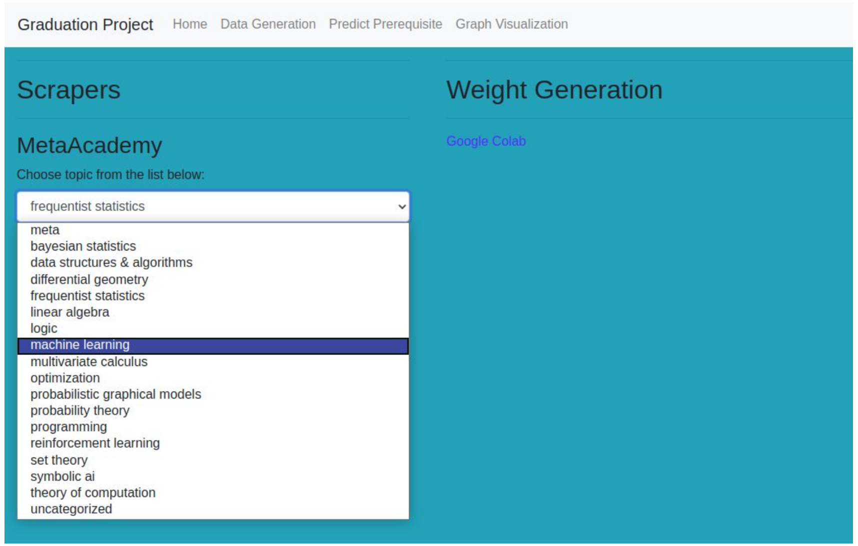
Some Other Website

Choose topic from the list below:

test1

Weight Generation

Google Colab



Scrapers

MetaAcademy

Choose topic from the list below:

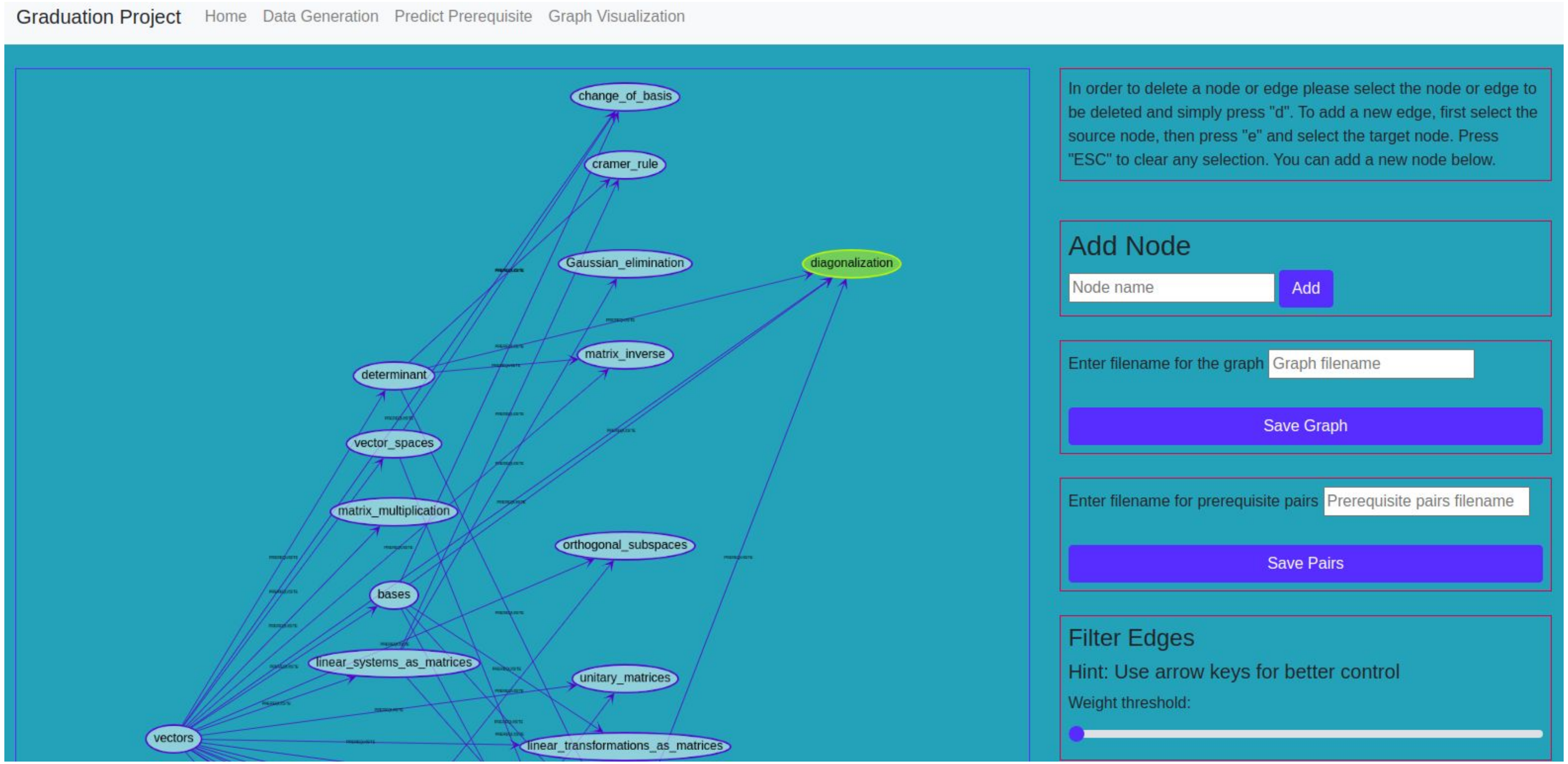
frequentist statistics

- meta
- bayesian statistics
- data structures & algorithms
- differential geometry
- frequentist statistics
- linear algebra
- logic
- machine learning
- multivariate calculus
- optimization
- probabilistic graphical models
- probability theory
- programming
- reinforcement learning
- set theory
- symbolic ai
- theory of computation
- uncategorized

Weight Generation

Google Colab

7. Graphical User Interface (4)
Flask - 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.

Add Node

Node name

Enter filename for the graph: Graph filename

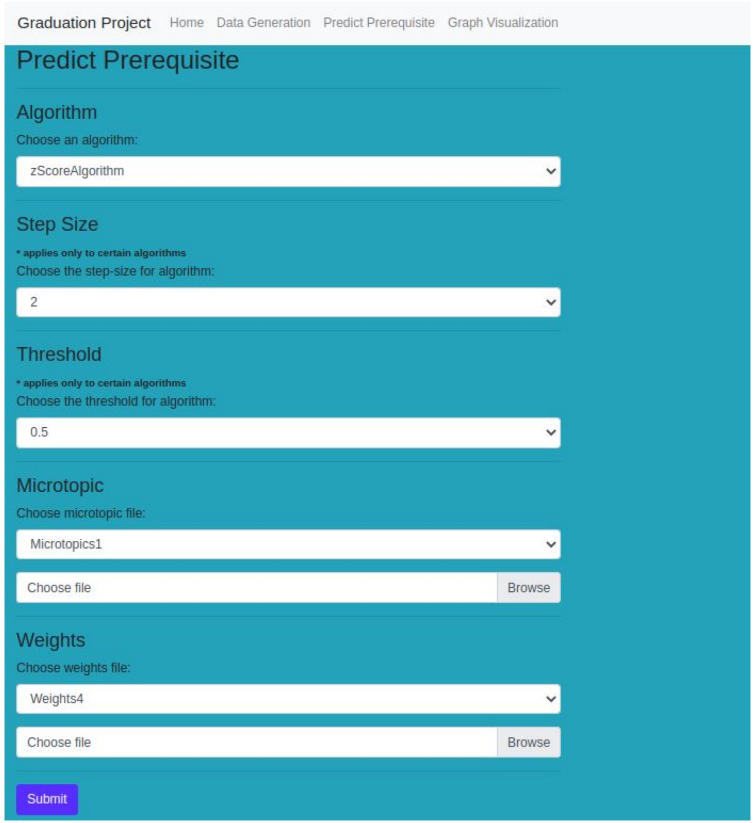
Enter filename for prerequisite pairs: Prerequisite pairs filename

Filter Edges

Hint: Use arrow keys for better control

Weight threshold:

7. Graphical User Interface (5)
Flask - Predict Prerequisite


Graduation Project · Home · Data Generation · Predict Prerequisite · Graph Visualization

Predict Prerequisite

Algorithm
Choose an algorithm: zScoreAlgorithm

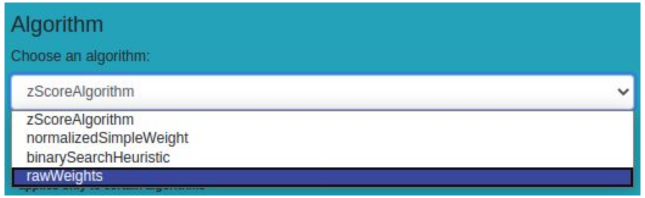
Step Size
* applies only to certain algorithms
Choose the step-size for algorithm: 2

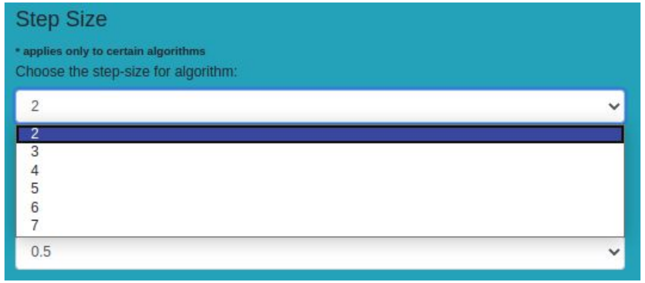
Threshold
* applies only to certain algorithms
Choose the threshold for algorithm: 0.5

Microtopic
Choose microtopic file: Microtopics1
Choose file · Browse

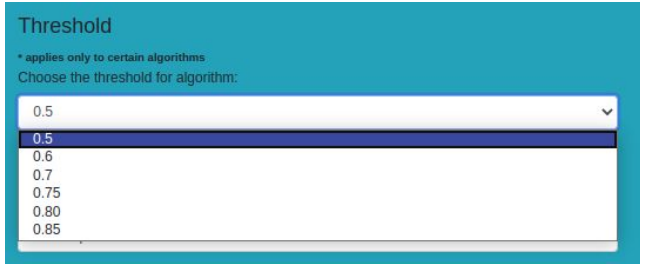
Weights
Choose weights file: Weights4
Choose file · Browse

Submit


Algorithm
Choose an algorithm: zScoreAlgorithm
- zScoreAlgorithm
- normalizedSimpleWeight
- binarySearchHeuristic
- rawWeights


Step Size
* applies only to certain algorithms
Choose the step-size for algorithm: 2
- 2
- 3
- 4
- 5
- 6
- 7

0.5


Threshold
* applies only to certain algorithms
Choose the threshold for algorithm: 0.5
- 0.5
- 0.6
- 0.7
- 0.75
- 0.80
- 0.85

7. Graphical User Interface (6)
Flask - Video Demos

- Web Scraper
- Predict Prerequisite & Graph Visualization

8. Conclusion (1)

Brute Force

- 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.

First and Second Heuristic

- On average over 100 iterations, the expert will be asked around 55 times
- 38% better than the pure brute force algorithm.

Third Heuristic

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

8. Conclusion (2)

Approximating Prerequisite Relationship with Weights

- 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.

8. Conclusion (3)

Final Look

- **The objective is achieved since we developed a graphical user interface with the needed functionalities for experts to use.**
- **The most important limitation is calculating weights of micro topics.**
- The calculation of weights with the active learning algorithms takes days, there is no sufficient data at the moment, therefore data to test algorithms are not sufficient.
- **The calculation of weights is done with a google colab script, but there is no proper way to run this script from the Flask backend.**