Project Final Report: Course Content Search Engine

Chen Xuanwen
School of Data Science
The Chinese University of Hong Kong, Shenzhen
120090582

Peng Yiwei
School of Science and Engineering
The Chinese University of Hong Kong, Shenzhen
119010245

Zeng Zhuoru
School of Science and Engineering
The Chinese University of Hong Kong, Shenzhen
11901041

Zhuang Yan
School of Science and Engineering
The Chinese University of Hong Kong, Shenzhen
120090303

***Abstract*—A search engine specified for searching course content is designed and implemented in this project. Such an engine aims at helping students to find the most important and relevant course content when they are doing assignments or reviews. The engine takes in search keywords and provides a ranked list of related content results based on importance score, relevance score, and user feedback.**

***Index Terms*—Course Content Search Engine, Importance Score, Relevance Score, Inverse Page Frequency**

I. Introduction

With the advent of digital technology, search engines have become ubiquitous in our modern world. Search engines are commonplace tools used for various purposes ranging from product and service research to academic pursuits. Educational search engines are search engines that specialize in collating educational resources. These specialized search engines can help students and teachers find information more easily and efficiently.

Educational search engines offer several benefits to their users. One of the most significant benefits is that they can save time. A study by Barnes and Tullis (2015) found that students who used educational search engines spend less time searching for information than those who used general-purpose search engines. Educational search engines are designed to deliver highly relevant and credible educational content. Most educational search engines allow users to refine their search results by filtering results by topic, resource type, or other parameters, helping users find information more quickly.

Like generic search engines, educational ones are based on indexing and searching techniques. Four indexing and searching techniques exist for educational search engines. There is the technique of classic indexing and searching that indexes and searches only words that forget their meaning, this technique suffered from several concerns influencing the performance of search engines and their responses. There is another technique of indexing and semantic search, which indexes words and their meanings while improving the performance of information retrieval systems. He had appeared to overcome the limitations of the classic one. Another type of indexing by metadata exists, which does not fit into the contents of the learning objects but just index their structure, its indexation is done most of the time manually by human beings, it is based on standards for example the LOM; confirming the usefulness of learning objects searching. A combination of the different techniques becomes a necessity, hybrid, to take advantage of its strengths and to include them in a powerful model capable of responding to the indexing of learning objects.

While educational search engines offer several benefits, there are some drawbacks. One significant issue with educational search engines is that their algorithms can limit the variety of information available to users. Search engines prioritize popular or trending content, and less popular educational content may not appear in search results. Another issue is that search engines may inadvertently reinforce existing biases and stereotypes by prioritizing content from biased sources or by reflecting the biases of their creators.

One current issue related to educational search engines is their capacity for personalization. Personalization algorithms can help students receive more relevant educational content.

In this project, we proposed to build an educational search engine for searching textbooks contents and ppt slides. Several tasks have been finished in this project: we built a database of text content from ppt slides in EIE3280 courses, we proposed and implemented several search algorithms and compared their performance, and we built a website-based application for users to interact with our search engine and database.

II. System Design

The course content search engine system's overall structure consists of four parts: a database, a text-matching component, a content ranking component, and a user feedback component. Figure 1 demonstrates the structure of our system and the following subsections will illustrate the design of each component.

The input to the search engine is a search query sentence. In the text-matching component, decisive tags are determined and the related sections that contain decisive tags are found. In the content ranking component, the related sections are ranked by algorithms, and the search results of the search engine, a ranked list of sections, are produced.

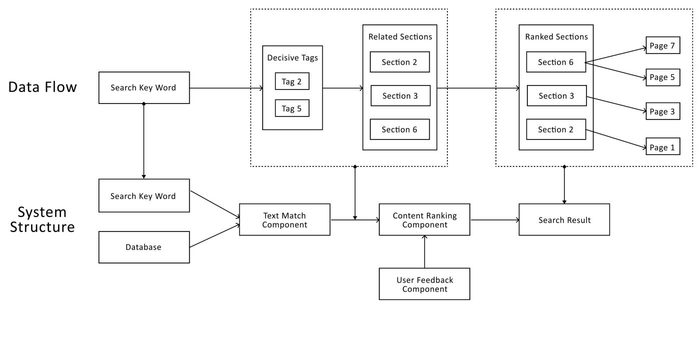

Fig. 1. Search Engine System Design

The code for database creation and algorithm implementation is available at https://github.com/Philip-ZENG/course-content-search-engine

A website application that allows users to interact with our search engine and database is implemented. The code for the website application is available at https://github.com/Philip-ZENG/course-content-search-engine-web

A. Data Structure and Database

1) Database: The database used in this project is created by our team members. The text content of the database mainly comes from the course slides of EIE3280. A Non-SQL database, MongoDB, is chosen to store structural text data.

2) Data Structure: Text data from course slides are split into pages and multiple pages form a section. Each page contains one title field and one content field. Each section is manually assigned tags that summarize its key contents. A graphical illustration of the database's schema is shown below in Figure 2.

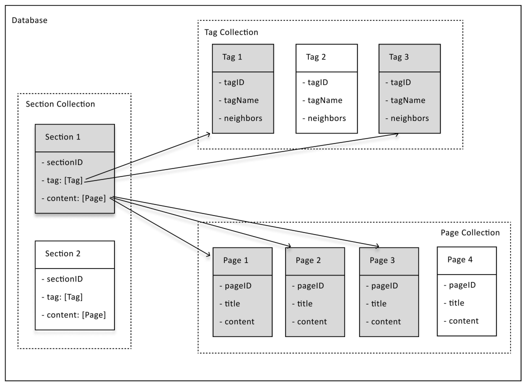

Fig. 2. Database Schema Design

3) Tags as a Graph: Tags are used to discover the connections and relationships between content (e.g. different chapters, theorem, or knowledge points). An undirected graph is built by taking each tag as a node and possible relationships between tags as the edges of the graph. Properties of graphs and algorithms that make use of the graph model are used to calculate a relevance score that produces the ranked search results. An example of the tag graph we built based on our data is shown in Figure 3.

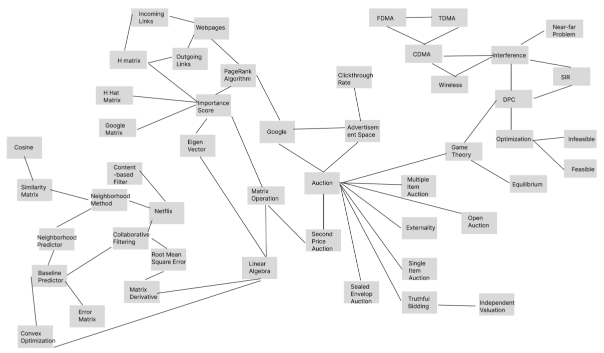

Fig. 3. A Portion of Tag Graph

B. Text Matching

The text matching component in the search engine does a simple job of matching the search keywords string entered by the user with text content in our course material database. The matched page, section, and tags will be used for further analysis and relevance score calculation in the consecutive components.

C. Content Ranking

The content ranking component is designed to rank the content extracted from the text-matching step. A ranking score will be calculated for each section and sections will be listed in descending order based on their score as the search results. Multiple content ranking algorithms are designed and implemented. These algorithms will be discussed in detail in the Content Ranking Algorithm section.

D. User Feedback

A user feedback mechanism is designed to help improve the quality of search results. After reading the search result content, users can give their feedback, a binary rating, on whether they think the content is helpful or not. A detailed discussion on how user binary feedback is used to adjust ranking scores will be presented in the Content Ranking Algorithm section.

E. Website Applicaition

A website application is built to provide a user interface for people to interact with our search engine and database [1]. The application has a front-end built with ReactJS/NextJS framework, a back-end built with ExpressJS framework, and a database built with MongoDB. Figure 4 shows the structure of the app.

[1]A video demo of this app can be found at https://github.com/Philip-ZENG/course-content-search-engine-web/tree/main/demo

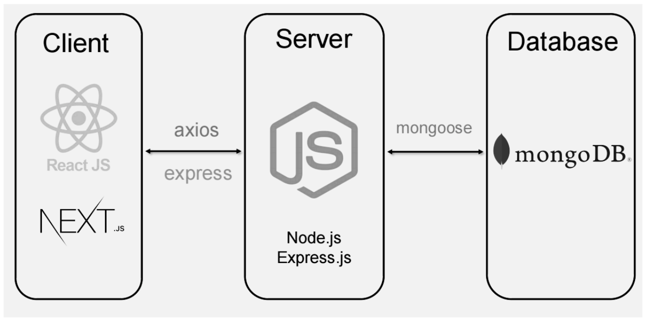

Fig. 4. Structure of the Website Application

III. Content Ranking Algorithm

Multiple content ranking algorithms are designed and implemented. The technical details of these approaches will be discussed in this section.

A. Distance Based Relevance Score Ranking

The first approach we design is based on an intuitive idea that the smaller the distance between two tags is (distance can be calculated through the tag graph we built), the more relevant these two tags are. This approach is taken as the benchmark of our study.

1) Find Decisive Tags: Corresponding tags of the pages extracted from the text-matching step will be counted. A different weight is assigned to the occurrence count when the text match occurs in the content field and title field of the page. The tags that had the top 3 highest occurrence frequencies are considered the deterministic tag for the search. The following Algorithm 1 describes the process in detail. The Text Matched Pages denote the database query results retrieved from the text-matching component. ν and μ are constant parameters.

Algorithm 1 Find Decisive Tags

```
create Map $FreqMap : Tag \rightarrow OccurrenceFrequency$
initialize all $OccurrenceFrequency$ to 0
**for** Page in Text Matched Pages **do**
    find corresponding Tags of the Page
    **for** Tag in Tags **do**
        **if** text match occurs in content field **then**
            $FreqMap[Tag] \leftarrow FreqMap[Tag] + \nu$
        **end if**
        **if** text match occurs in title field **then**
            $FreqMap[Tag] \leftarrow FreqMap[Tag] + \mu$
        **end if**
    **end for**
**end for**
find $DecisiveTag$ with the top 3 $OccurrenceFrequency$
**return** $DecisiveTag$, $FrequencyMap$
```

A graphical illustration of this process is shown in Figure 5. We first find pages that contain the search keyword, and then find the corresponding tags of these pages. We count tag occurrence frequency and determine the decisive tags.

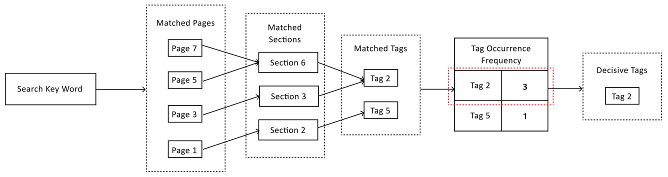

Fig. 5. Steps to Find Decisive Tags

2) Distance Based Relevance Score: A graph is built by taking each tag as a node and the connections between tags as edges. An intuitive idea is that the smaller the distance between two tags is, the more relevant these two tags are. Based on this idea, we first use Dijkstra Algorithm [1] to calculate the shortest distance between two tags x and y and denote it as $D(x, y)$. The Dijkstra Algorithm used in our project is described in Algorithm 2. The link cost of the tag graph is set to be a constant number λ in our model.

Algorithm 2 Dijkstra

```
create Map $DistanceMap : Tag \rightarrow ShotestDistance$
create Array $LeastCostPath$
$LeastCostPath.append(SourceTag)$
**for** Tag in All Tag Set **do**
    **if** Tag is SourceTag **then**
        $DistanceMap[Tag] \leftarrow 0$
    **else if** Tag is the neighbor of SourceTag **then**
        $DistanceMap[Tag] \leftarrow \lambda$
    **else**
        $DistanceMap[Tag] \leftarrow \infty$
    **end if**
**end for**
**while** not all tag is in $LeastCostPath$ **do**
    $MinDistance \leftarrow \infty$
    $MinDistanceTag \leftarrow Null$
    **for** Tag not in $LeastCostPath$ **do**
        **if** $DistanceMap[Tag] < MinDistance$ **then**
            $DistanceMap[Tag] \leftarrow MinDistance$
            $MinDistanceTag \leftarrow Tag$
        **end if**
    **end for**
    $LeastCostPath.append(MinDistanceTag)$
    **for** Tag that is neighbor of $MinDistanceTag$ **do**
        **if** Tag is not in $LeastCostPath$ **then**
            **if** $DistanceMap[Tag] > MinDistance + \lambda$ **then**
                $DistanceMap[Tag] \leftarrow MinDistance + \lambda$
            **end if**
        **end if**
    **end for**
**end while**
**return** $DistanceMap$
```

We use the following equation to calculate the relevance score RS between two tags. α and β are two constant parameter.

$$RS(x,y) = \frac{\alpha}{D(x,y)+\beta} \tag{1}$$

For a section that has N tags, we denote the set of these tags as T. We can calculate the relevance score of section s with regard to a decisive tag d using the following equation. The section relevance score is denoted by SRS.

$$SRS(s,d) = \frac{1}{N}\sum_{i\in T} RS(i,d) \tag{2}$$

Since we select multiple decisive tags for a specific search query in the previous step, we need to aggregate the section relevance score towards a single decisive tag.

We use a tag occurrence frequency weighted method to aggregate multiple section relevance scores. The Occurrence Frequency of tag j is denoted as $OF(j)$ and its value is already calculated in the Find Decisive Tags Algorithm. We denote the set of decisive tags as DT and the aggregated section relevance score of section s with regard to tag set DT as $ASRS(s, DT)$.

$$ASRS(s,DT) = \sum_{j\in DT} \frac{OF(j)}{\sum_{k\in DT} OF(k)} \times SRS(j) \tag{3}$$

3) User Feedback Adjusted Ranking Score: A user feedback mechanism is set up to help improve the search result quality. Every user can give binary feedback to a section returned as search results and the feedback for each section is recorded in the database.

For a section s with p positive feedback and q negative feedback, the feedback score of the section $FS(s)$ can be calculated by the following equation. σ and δ are constant parameters.

$$FS(s) = \begin{cases} \frac{\min\{p-q,\sigma\}}{\delta} & \text{if } p-q>0 \\ \frac{\max\{p-q,-\sigma\}}{\delta} & \text{if } p-q<0 \end{cases} \tag{4}$$

The final rank score of section s with decisive tags DT can be calculated by:

$$RankScore(s) = ASRS(s,DT) + FS(s) \tag{5}$$

For each section we retrieved from the text-matching step, we compute its rank score and rank these sections in descending score order. The ranked section list shows the relevance of each section with regard to the search keyword. The detailed process of calculating a ranked section list is shown in Algorithm 3.

B. Importance Weighted Relevance Score Ranking

This approach has a similar process as the Distance Based Relevance Score Ranking but takes the importance of tags into account during the ranking process. The importance of a tag is evaluated using the Google Page Rank Algorithm [2] with the tag graph we built. The page rank algorithm is described in Algorithm 4 below. ϵ is the error tolerance bound and π vector is the importance score vector (or called page rank vector).

Algorithm 3 Find Ranked Section List

```
create Map $ScoreMap : Section \rightarrow RankScore$
for Section in Text Matched Sections do
    $ASRS \leftarrow 0$
    for DecisiveTag in Decisive Tags do
        find corresponding Tags of Section
        $SRS \leftarrow 0$
        for Tag in Tags do
            $SRS \leftarrow SRS + RS(Tag, DecisiveTag)$
        end for
        $SRS \leftarrow \frac{1}{N} \times SRS$
        $ASRS \leftarrow ASRS + SRS$
    end for
    $RankScore \leftarrow ASRS + FS(Section)$
    $ScoreMap[Section] \leftarrow RankScore$
end for
create $SectionList$ with $Section$ in $ScoreMap$ as element
rank $SectionList$ element by corresponding $RankScore$ in descending order
return $SectionList$
```

Algorithm 4 Page Rank

```
construct Matrix $H$
$G \leftarrow \theta H + (1-\theta)\frac{1}{N}11^T$
create $\pi_t$ vector with N elements and each has value 0
create $\pi_{t+1}$ vector with N elements and each has value $\frac{1}{N}$
$diff \leftarrow \pi_{t+1} - \pi_t$
while norm of $diff < \epsilon$ do
    $\pi_t \leftarrow \pi_{t+1}$
    $\pi_{t+1} \leftarrow \pi_{t+1} \times G$
    $diff \leftarrow \pi_{t+1} - \pi_t$
end while
return $\pi_{t+1}$
```

The major difference between this approach and the Distance Based Relevance Score Ranking lies in the equation of calculating the section relevance score SRS. The importance score of a tag IS is used as the weighting factor.

$$SRS(s,d) = \frac{1}{N}\sum_{i\in T} RS(i,d) \times IS(i) \tag{6}$$

All the other process of this approach is the same as the Distance Based Relevance Score Ranking.

C. Inverse Page Frequency Relevance Score Ranking

In some cases, users may enter irrelevant search terms like “the”. Intuitively, these terms should contribute very little to our search results. If we calculate their contribution just like other essential words, our result may be quite inaccurate. Therefore, we will introduce the tf-idf technique [3] to our project. This means, if one term occurs in many sections/pages, the weight of the term should be very small. When we search for the result, this term’s contribution should be relatively small compared with other terms.

The equation to calculate the inverse page frequency is as follows, i denotes the search keyword, P denotes the number of total pages in the database, C_i denotes the number of pages in which the search keyword i is found:

$$ipf_i = log\left(\frac{P}{C_i}\right) \tag{7}$$

The inverse page frequency ipf is used to adjust the process of finding decisive tags as shown in Algorithm 5.

Algorithm 5 Inverse Page Frequency Adjusted Decisive Tags

```
create Map $FreqMap : Tag \rightarrow OccurrenceFrequency$
initialize all $OccurrenceFrequency$ to 0
for Word in Search Input Sentence do
  find InversePageFrequency $ipf$ of the Word
  for Page in Word Matched Pages do
    find corresponding Tags of the Page
    for Tag in Tags do
      if Word occurs in content field then
        $FreqMap[Tag] \leftarrow FreqMap[Tag] + \nu \times ipf$
      end if
      if Word occurs in title field then
        $FreqMap[Tag] \leftarrow FreqMap[Tag] + \mu \times ipf$
      end if
    end for
  end for
end for
find $DecisiveTag$ with the top 3 $OccurrenceFrequency$
```

IV. Experiment

A. Evaluation Method

We use a modified Mean Reciprocal Rank (MRR) as our evaluation criteria. MRR is a measure to evaluate systems that return a ranked list of answers to queries. For a single query, the reciprocal $rank$ is $\frac{1}{rank}$, where $rank$ is the position of the highest-ranked answer $(1, 2, 3...N$ for N answers returned in a query). If no correct answer was returned in the query, then the reciprocal rank is 0. The equation is:

$$\frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{rank_i} \tag{8}$$

For our project, according to the database, for different test cases, the number of the relative sections is different. So the number of correct answers for different index words is different. Based on the MRR method, we designed our own evaluation method. The equation is as follows:

$$\frac{1}{Q}\sum_{i=1}^{Q}\left(\frac{1}{n_i}\sum_{j=1}^{n_i}\frac{1}{rank_i}\right) \tag{9}$$

Q denotes the number of query test cases we have in our test set. n_i denotes the number of correct answers we have for the specific query i. $rank_i$ denotes the rank of a specific correct answer in our actual experiment output result.

Intuitively, the higher rank a correct answer is placed in the list of results, the more desirable the search result is, and the larger the MRR score would be.

B. Test Cases

Several test queries are designed by us and the "correct answers" (section IDs) to these queries are also assigned manually. In Table 1, some of the test cases are listed.

TABLE I
Example Test Cases

Test Query	Correct Answers (Section IDs)
vote	603, 604
Borda count	603
root mean square error	401, 503
page rank	302, 303, 304
movie rating	401, 402
neighborhood	402

C. Experiment Results

Four different algorithms are compared with each other with the same test cases. An example of the test result of a specific test query, say "Borda count", is shown in Table 2. The bold number is the correct answer.

TABLE II
Example Search Result on Borda Count

Algorithm	Search Results (Ranked Section IDs)
IS	602, 604, **603**
RS	602, **603**, 604
IS+RS	**603**, 602, 604
IR+RS+IPF	**603**, 602, 604

IS represents the importance score, in this approach, the importance score calculated from the Google Pagerank algorithm is used as the criteria for ranking relevant sections. RS represents the relevance score, and the distance-based relevance score is used as ranking criteria. IS+RS represents the importance-weighted relevance score approach, where the importance score is used as a weight to adjust the relevance score. IS+RS+IPF represents the importance-weighted relevance score with the inverse page frequency algorithm used in finding decisive tags.

The overall performance of these four approaches based on the Mean Reciprocal Rank evaluation method is shown in Table 3.

TABLE III
Performance Comparison

Algorithm	MRR Score
IS	0.4518
RS	0.4963
IS+RS	0.6119
IR+RS+IPF	0.5889

D. Discussions

From the experiment results, we can see that IS+RS approach has the best performance. The IS+RS approach is better than the performance of IS or RS alone is reasonable, since IS+RS approach takes both the importance score and relevance score into consideration. However, in our assumption, we expect the IS+RS+IPF approach would have better performance than the IS+RS approach.

The reason why the actual results differ from our expectations may be due to the following limitations of our experiment: the test cases are in relatively small number and the selection of the correct answer of the test cases are relatively subjective. Also, the tag graph and sections are manually built and assigned, their structure may have a significant influence on search results.

Future works can start with enlarging the database, and test cases. A more objective way of evaluating the search performance may also be an improvement to the project. In terms of the user experience, auto misspelling correction can be a good function to implement.

V. Conclusion

In this project, we built a database consisting of text data in course slides, designed several search algorithms and compared their performance, and construct a website application for search engine interaction.

References

[1] Dijkstra, E.W. A note on two problems in connexion with graphs. Numer. Math. 1, 269–271 (1959). https://doi.org/10.1007/BF01386390

[2] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank Citation Ranking: Bringing Order to the Web,” 1999.

[3] Rajaraman, A.; Ullman, J.D. (2011). ”Data Mining” (PDF). Mining of Massive Datasets. pp. 1–17. doi:10.1017/CBO9781139058452.002. ISBN 978-1-139-05845-2.

Contribution

The project workload is split into tasks, and each team member’s contribution to these tasks is listed in the table below:

Task	Contributor
System Design	Chen Xuanwen Peng Yiwei Zeng Zhuoru Zhuang Yan
Ranking Algorithm Design	Chen Xuanwen Peng Yiwei Zeng Zhuoru Zhuang Yan
Database Building	Chen Xuanwen Peng Yiwei Zeng Zhuoru Zhuang Yan
Coding	Chen Xuanwen Peng Yiwei Zeng Zhuoru Zhuang Yan

Contributor	Contribution (%)
Chen Xuanwen	22%
Peng Yiwei	22%
Zeng Zhuoru	34%
Zhuang Yan	22%