École polytechnique fédérale de Lausanne

Master Thesis

A Concrete Search Engine: Recommending Pedagogical Resources through Semantic Annotations

Frédéric Jan Ouwehand

Artificial Intelligence Laboratory - EPFL
Under the direction of Prof. Boi Faltings
Expert Claudiu Musat

June 2016

Abstract

In this thesis we suggest a search engine based on a novel approach to index pedagogical content using semantic annotations together with a graph oriented analysis. The approach takes into account the explicit structure of the resources and deduces the importance of each aspect discussed after an analysis of their relations. The structure is also reflected in the search engine results. Chapters and sections of books or modules and parts of online courses are considered as documents and indexed at this finest granularity level. Furthermore the system deduces and evaluates the key topics discussed in a book or a course based only on its table of content or syllabus without necessarily accessing its content. It can also detect the lack of correlation between the topics extracted and reject the task.

Wikipedia is used as a knowledge base both to understand the documents on a semantic level and to decide what the central topics are and the ones that should be used as index terms. The search engine allows the user to express its queries as a raw text or as a set of concepts from Wikipedia.

The user study shows that the algorithm can be applied to a concrete case with more than thirty thousand resources and that its results are comparable to the Bing.com engine within that dataset.

A demo of the search engine is available at https://wikichimp.com

Contents

1 Introduction

Actual search engines are based on the full-text search model which can index very large collections of documents. This simple model transforms the words in the document by a sequence of natural language processing steps and produces tokens which are indexed using inverted files as efficient data structures. The computational cost of this process depends on the complexity of the successive processing steps which may involve stemming or taking into account the syntax and semantics.

Full-text search is by definition bound to the text which contrasts with other approaches that consider the metadata associated with the document. In this thesis we focus on generating a set of tags for each document and use these as indexes. The tags should be equivalent to the entities from an extensive ontology in order not to limit the user which may search for any known topic and concept. The difference with full-text search can be seen as the addition of a selection step. A topic might appear in the text and not be an index term if the engine decides so. This selection process expresses the fact that not all topics that are contained in a document are relevant or central to the discussion in the document. We will later use the term *index* to refer to a Wikipedia entity (topic, concept) that was assigned to a document along with a centrality score.

This approach models the concepts and their relations with a directed graph. Vertices correspond to concepts from an ontology with a unique *is-related-to* type of relation between entities equivalent to the edges of the graph. The graph model is adapted to the analysis of interrelationship between the candidate concepts associated with the documents.

Not all documents are constituted by complete sentences. An important part of the information about the document is contained in its title and other headings. For instance table of contents are associated with books and syllabuses with courses. These are designed to be concise summaries of their contents. It also embeds a sense of hierarchy. Chapters are more general than their sections. And sections are related to their chapter. These hierarchical relations are embedded in the graph since the ontology is likely to contain them. For instance there could be a chapter about *Sorting Algorithms* that has a section about *Heapsort*. It is likely that the Wikipedia page of *Heapsort* has a link to the *Sorting Algorithms* page.

Wikipedia is an open and frequently updated online encyclopedia which contains a large collection of articles interconnected by links that could be interpreted as a graph of concepts with *is-related-to* relations. Our interrelationship analysis is based on these graphs. Our approach is evaluated on a concrete dataset constituted by table of contents from books and syllabuses from online courses. A dataset of pedagogical resources is a natural choice to evaluate such an approach since authors commonly organize their resources in many learning steps with titles, keywords or explicit curriculum.

In Part I, we first describe the dataset, a general view of the search engine architecture and its query interface. Part II explains how the documents are tagged using topics from the text and how the graph analysis is performed. Part III contains the evaluation of the entire search engine through a user study. Finally, we conclude with a discussion about future works in Chapter 15.

2 Related Work

Several solutions have been suggested for the annotation of documents with elements from an entity catalog, many of which using Wikipedia and the annotation process is referred as Wikification.

Mihalcea & Csomai [1] have suggested a way to enrich a text with links to Wikipedia articles which mimics the style of the Wikipedia page links. They estimate the likelihood of each term to be a keyword referring to a Wikipedia article. This measure is called the *keyphraseness* $p(keyword|W)$ of a term W and is estimated by counting the number of times that the term is used as link in the Wikipedia corpus divided by the number of times it appears in it. They decide to annotate a term if its keyphraseness exceeds a fixed limit.

Medelyan, Witten & Milne [2] noted that the keyphraseness should be a property related to the direct context of the term rather than a precomputed likelihood from the Wikipedia corpus. In this sense a *keyword* is a central term in the current document where the keyphraseness should reflect its importance. Their approach first extracts terms from the text linking them to candidate senses from the Wikipedia entity catalog and then proceed to a disambiguation step. Finally a filtering step categorizes each candidate as relevant or irrelevant for representing the document.

The extraction step uses a condition on the keyphraseness measure. They extract the surface forms associated with each topic and concept from Wikipedia. For instance *mouse* and *mice* both refer to the *mouse (animal)* concept. The terms that have enough keyphraseness in the document are then associated with all their candidate senses. For instance the *mouse* keyword is associated with the Wikipedia topics *mouse (animal)* and *mouse (computer device)*.

The disambiguation proceeds iteratively on the set of terms associated with multiple candidate senses. They evaluate each candidate by a similarity score with the set of terms that are unambiguous (i.e. associated with a single candidate sense). The Wikipedia Link Measure WLM by Milne & Witten [3] based on the link structure between two Wikipedia entities is defined as the similarity score. This measure is defined for any pair of candidates and counts the number of links to the set of topics that are related to both candidates. For instance *mouse (computer device)* and *algorithms* are both linked to *computer science*.

The WLM score is averaged over all the unambiguous keywords in the document. This is a first example of collective annotation and analysis performed on the Wikipedia graph of links. The average WLM score is then associated with the term *commonness* which is defined as the likelihood that a candidate sense is associated with a given term. Finally this score is used as one of the features of a classification based on machine learning techniques to decide whether the keyword should be retained as a link. This filtering step uses the position of the term in the document as a feature and is an example of how the structure of the document can be taken into account. Later Milne and Witten [4] also used the WLM relatedness measure as a feature to learn to attribute the correct sense to each word n-gram identified as a potential keyword.

Kulkarni *et al.* [5] expressed the task as an optimization problem in which

the solution maximizes the local compatibility between the term and its candidate senses. The overall agreement between all the candidates is based on the WLM measure. In contrast with previous approaches, Kulkarni suggests a way to consider the entire set of interconnections between each candidate topic. The final decision takes into account the entire set of links from this graph of candidates. However it does not evaluate how central each term is in the document.

Coursey and Mihalcea [6] explicitly expressed the problem with a graph model. They use an adaptation of the PageRank algorithm [7] by Haveliwala [8] which is biased towards the topics which appear in the text. This approach ranks the topics by how central they are in the document and uses the WIKIFY system [1] to extract topics from the document. Hence its performance depends on the performance of the wikifier system which may produce an important part of irrelevant topics. Their approach does neither evaluate the coherence of the tags produced nor reject the task when the wikifier performs badly.

In a previous study [9], we evaluated the biased PageRank approach from Coursey and Mihalcea [6] using the SPOTLIGHT wikifier [10] on a small set of table of contents. We found that it is able to successfully rank the important topics in the first positions and relegate the topics considered as pollution in to the last ranks. The study suggested that a connection analysis could detect poorly annotated documents and reject them but did not provide any such algorithm. Also the study suggested that the graph analysis could help the wikifier with its knowledge of interconnections like Kulkarni *et al.* did. Finally, the study is based on potentially large graphs and does not specify a way to use smaller graphs for concrete cases.

None of the above studies weight the information by priority. Magdy and Darwish [11] suggested that some parts of the documents are more important to index. In practice, we want to prioritize titles, headings and table of contents over descriptions and the content itself.

Part I
Environment

We evaluate the system in the concrete scenario of a search engine for pedagogical resources. We first describe in Chapter 3 the dataset and the difficulties associated with it. Chapter 4 presents the global architecture of the search engine and how it differs from the usual information retrieval models. Chapter 5 details the user interface for typing in the query using topics from Wikipedia.

3 Dataset

The dataset contains a large majority of table of contents (TOC) from books that are accessible online as shown in Table 1. It is important to note that these TOCs are freely available in contrast with their content which are accessible only via a paid plan or via public libraries. The experiment was performed with students which had access to the books online when connected from the university network. The books are not particularly from a specific domain or field.

The dataset contains a considerable amount of public online courses on a large collection of fields but corresponds roughly to the high school, college and university material. The set is completed with a thousand of scientific articles from an online encyclopedia on the fields of Astrophysics, Celestial mechanics, Computational neuroscience, Computational intelligence , Dynamical systems and Physics. See [1] for a complete list of the domains.

Source	Category	Amount
Coursera.org	Online courses	1,811
ocw.mit.edu		1,350
KhanAcademy.org		475
Safaribooksonline.com	Online books	29,797
Scholarpedia.org	Scientific articles	938

Table 1: Documents distribution

This dataset was chosen to contain as many different fields and expertise levels as possible in order to match a large spectrum of user queries.

3.1 Challenges

The main challenge is to evaluate the algorithms on a diverse set of documents and assess their ability to work in various scenarios. In this sense, the books TOCs contrast with online courses syllabuses since they are often longer and have a more complex hierarchy. It is not unusual to have parts, chapters, sections, subsections and sub subsections in books where courses tend to have a single level week-by-week or modules oriented hierarchy.

The wikifier success depends on the number of ambiguous terms present in the documents. Some fields have less ambiguous terms like Medicine or Biology. For instance the molecules or diseases often have unambiguous names. The dataset does not target a particular field which makes the wikifier performance vary. Therefore the evaluation of the entire system is less dependent of its external modules performances.

Each author has its own style of writing and structuring a resource. Some authors like to explicitly append a structural annotation such as "Chapter x: .."

[1] http://www.scholarpedia.org/article/Scholarpedia:Topics

before each chapter title. Other authors insert administrative elements in their course syllabus such as "Exercise" or "Final Exam". The TOCs and syllabuses come from different authors which helps to minimize the bias and make more accurate performance measurements.

A consequent part of the resources do not have a meaningful table of content or syllabus but are still considered since relevant information can be extracted from their title or descriptions. The dataset also includes some metadata such as keywords or related categories which appear in the resource page.

3.2 Extracting information from websites

This module downloads the remote and unstructured data into a local and structured database. Documents on the web change frequently and are not always available, if not sometimes slow to access due to latencies. Hence it is crucial to have the resources in a local database which serves them efficiently with fewer failures. The algorithms consider a resource to be a set of textual content that can take the form of a structural hierarchy similar to a tree.

Each downloaded resource should be stored in a normalized format in order for the other modules to reason about them more easily. Technically, the books TOCs and the courses syllabuses are downloaded and stored in a JSON file of *key* - *value* pairs. The values may themselves be *key* - *value* pairs in order to store in a tree like manner a table of content of arbitrary depth.

The extraction system consists of three separate modules. A first module navigates on the web and stores the links of courses and books TOCs. A second module downloads each page by following the collected links. Finally the third module matches the downloaded pages with known layouts and extracts the information in the structured JSON format.

These three steps successively tackle three problems. Downloading and rendering web pages are the main time cost of the extraction system. Reducing the number of pages to visit is hence a priority. In this step the system reduces the set of pages to visit close to the optimal set constituted of the pages which contain the table of contents and the course syllabuses. The module usually proceeds by fetching a set of listing pages which reference all the resources. An analysis of these listings produces the set of links to relevant resources.

Actual websites are made of several scripts and page fragments that are loaded dynamically. Rendering a page can take several seconds before the information is displayed. The second step is to download the page scripts and execute them until the relevant information is displayed. This step also tackles problems associated with loading such as server errors or redirects. This task is managed by a scheduler which takes care of planning when and whether a failed page is rescheduled. Web pages often contains errors in their scripts which do not affect the browsers in most of the cases but may lead to rare crashes or infinite loops in the browser. For this reason the module is handled at the process level by a control program which saves the state of the module after each step and relaunches it in case of crashes.

The website presents the information in multiple ways but each layout is not necessarily associated with a specific kind of information. The third step first recognizes the page layout and then extracts its information. The module quickly evaluates how complete is the data through several tests.

4 Text Retrieval System Architecture

The search engine modules are arranged using the standard scheme defined by Baeza-Yates and Ribeiro-Neto in the book Modern information retrieval [13]. Some adaptations have been made to integrate the topics from the Wikipedia entity catalog as index terms. The adapted scheme is displayed in Figure 1.

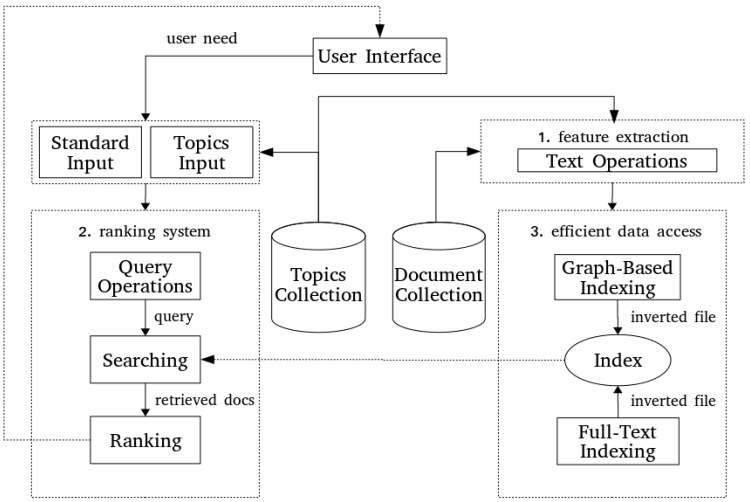


Figure 1: Text Retrieval System Architecture

The documents are first processed to extract features which are then analyzed to produce indexes that are stored in an inverted file. Our approach may not have results for every topic present in the entity catalog. The user might want to express their queries as sentences like in the actual popular search engines. The scheme contains an optional full-text search engine to complement our approach in such cases. Both our approach and the full-text one produce indexes that are stored in an inverted file. These first steps are shown on the right side of Figure 1.

The features are different for each indexing method. The graph-based algorithm reasons about topics extracted by a wikifier which are considered to be the document features. In contrast the full-text search engine uses stemming to produce tokens which are the features. The wikifier and the graph-based algorithm rely on the entities and interlinks from Wikipedia. Figure 1 expresses this dependence with an arrow from the Topics Collection to the feature extraction step.

This thesis uses an open implementations of both the full-text search engine and the wikifier. They are respectively ELASTICSEARCH [2] and SPOTLIGHT [3].

Standards full-text search engines use the feature extraction step on both the documents and the user queries. In our approach the user query is treated differently. Topics are suggested as the user types its query using the automatic completion mechanism described in Chapter 5. This mechanism is convenient for the user and allows the query to be expressed as Wikipedia topics. It is not mandatory and the users might combine textual sentences and Wikipedia topics in their query. However our graph-based engine can only understand Wikipedia topics. This affects how the queries are handled. For instance, a user who inputs the topic *Algorithms* and the sentence "*computational costs calculation examples*" will trigger from both our graph-based and the full-text search engines. However our approach will only search for documents about *algorithms* although the full-text search engine considers the entire sentence "*algorithms computational costs calculation examples*".

Figure 1 shows these two types of input on the top left side. The completion system requires to know what the topics are contained in the entity catalog. An arrow from the topics collection to the topics input expresses this relation.

[2] https://www.elastic.co/

[3] https://github.com/dbpedia-spotlight/dbpedia-spotlight

5 Topics Suggestion

Our graph-based topic extraction approach produces indexes that are made from Wikipedia topics. The search engine hence requires the user queries to be formulated as topics rather than sentences. For instance the phrase "*math proofs using algorithms*" should be translated into the topics {"*Mathematical_proof*", "*Algorithm*"} which correspond to Wikipedia identifiers for the articles about *mathematical proof* and *algorithms* respectively. These identifiers are not convenient for the users. A completion mechanism helps the user to type in faster its query by suggesting topics which match the current textual input. Then transparently converts topics into Wikipedia identifiers. Section 5.1 describes this completion mechanism and Section 5.2 details its performance.

5.1 Completion mechanism

The completion mechanism associates multiple surface forms with each topic. The surface forms are extracted from three types of Wikipedia data. Each article contains a title which is used as the primary surface form. The title type is not sufficient in general since many more labels are associated with each topic such as acronyms or plural forms. These variations are encoded by Wikipedia redirects that are pages containing only the title variation, linked to the main article page. In contrast, one main surface form can refer to many topics of different domains. Wikipedia contains disambiguation pages with links to the candidate senses. Titles, redirects and disambiguations were extracted from Wikipedia by the DBPedia project[4] which aims at providing a structured version of Wikipedia for programs to reason about. Table 2 displays some surface forms associated with the animated sitcom *The Simpsons* with their type. This topic has in total 85 redirects and 2 disambiguation pages.

The popular surface form "*simpson*" may refer to the animated sitcom but also to other topics and is hence a disambiguation. Table 3 presents some different concepts associated with the "*simpson*" term.

The suggestions are displayed instantly when the query is entered by the user. The system searches surface forms which have the user input as prefix in one of their words. The titles, redirects and disambiguations are checked for the input and the most relevant topics are displayed to the user. The number of topics depends on the size of the input since small inputs prefix more surface forms. Table 4 displays the number of distinct articles which match a given prefix of "*simpson*". There is a minimum of 133 different articles matching this prefix.

The system first displays the candidates with a length equal to the input prefix. The rest of the candidates are ordered by decreasing popularity. Only a fraction of the Wikipedia topics have a matching index in at least one of the documents. Such topics are said to be available and are always displayed before the unavailable ones. Table 5 shows the top concepts matching the input "*simp-*

[4] http://dbpedia.org/

Surface form	Type
The Simpsons	Title
Simpsons	Redirect
The Simpson's	Redirect
TheSimpsons	Redirect
The simppsons	Redirect
The simsons	Redirect
Thr Simpsons	Redirect
Neologisms in The Simpsons	Redirect
Recurring jokes on the simpsons	Redirect
Simpsons TV ads	Redirect
Why You Little	Redirect
Simpson	Disambiguation
The Simpsons (disambiguation)	Disambiguation

Table 2: Surface forms associated with the animated sitcom *The Simpsons*

Concept	Domain
The Simpsons	animated sitcom
Simpson's paradox	probability and statistics
The Simpson index	diversity index

Table 3: Disambiguations associated with the input "*simpson*"

Prefix	Number of articles
s	223,246
si	24,094
sim	3,021
simp	548
simps	134
simpso	134
simpson	133

Table 4: Number of articles matching the prefix

Concept	Popularity	Available
The Simpsons	5,230	yes
Simpson family	290	yes
Simpson's rule	37	yes
Simpson Medal	179	no
Simpson College (Iowa)	159	no
Scott Mountains (Antarctica)	19	no

Table 5: Suggested concepts for input "*simpson*"

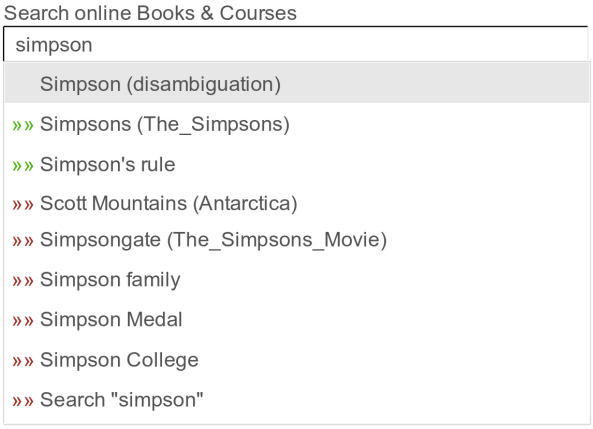


Figure 2: User interface showing suggestions for input "*simpson*"

son" and their availability in our dataset. In practice we define the *popularity* of a topic as the number of Wikipedia pages that link it.

The user interface is shown in Figure 2. Available topics are marked with a green arrow and unavailable ones with a red arrow. The last entry "*Search "simpson"*" allows the user to search for the input as a raw text. This option is displayed last when there are suggested topics but is the only option when the suggestion mechanism does not find any matching topics. In this case the system will not use the graph-based search engine. The first entry of Figure 2 is a disambiguation which means that many concepts correspond to this input. The user can see the candidate concepts as shown in Figure 3 .

5.2 Performance

The number of candidate topics associated with a prefix can grow rapidly as seen previously and the system must perform instantly when the user types a query. Surface forms from titles, redirects and disambiguations are stored in a relational database along with the Wikipedia topic identifier. The number of

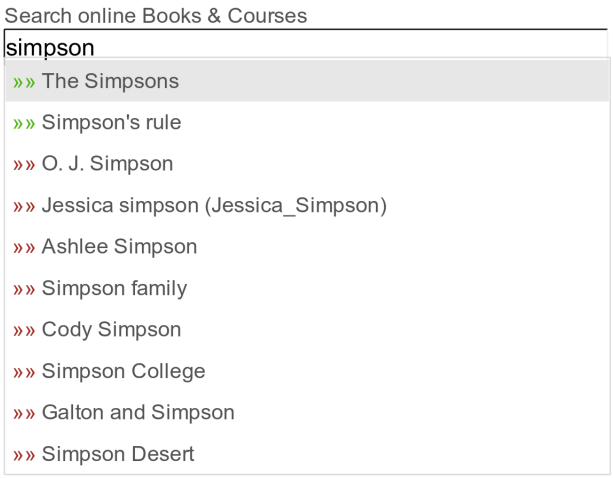


Figure 3: User interface showing disambiguation options for entry "*simpson (disambiguation)*"

Type	Number of entries
Disambiguations	709,951
Redirects	3,879,979
Titles	1,561,134

Table 6: Number of surface forms of each type

surface forms is fixed and of reasonable size as shown in Table 6. A hash index on the surface form provides constant time access to the candidate topics. Table 7 shows the different strategies to reduce the computational cost. Suggestions for letters only return exact matches sorted by popularity. Input of two and more letters return exact matches and topics that are prefixed by the input. The length of each candidate is not stored and is recomputed at each query. Such computations are only made when the input has four or more letters since there are fewer candidates with such prefix.

Input size	Strategy	Sorting criteria
One	Exact match	Popularity
Two or three	Exact or prefix match	
Four or more		Length and Popularity

Table 7: Dynamic queries used by the suggestion mechanism

Part II
System Description

We have previously seen what is the environment and the search engine scenario that we use to evaluate our approach. We now focus on the central component which assigns tags to any document by analyzing its structured and unstructured textual content.

The tags constitute the metadata of each document and may be used by a search engine as features to compute the similarity between any query and document. The component considers the topics that appear in the document but might produce tags that do not appear in the resource. These tags are finally ranked by how central they are in the document.

Chapter 6 introduces the principal steps of this tagging system and their respective roles. Chapters 7, 8 and 9 explain each step in more details. Chapter 11 shows how the engine uses the produced tags to retrieve relevant documents and allow pagination in the results.

6 Problem Approach

The process of reading a text and extracting topics from Wikipedia is performed by a wikifier. There are three major problems that can occur.

1. The wikifier fails to detect surface forms in the text which refer to some important topics.
2. The wikifier detects an ambiguous surface form but fails to attach the correct sense.
3. An important topic appears in the text with a surface form that is unknown to the system.

These scenarios affect the global performance of the system and are solved iteratively as shown in Figure 4.

Figure 4: From some textual data to a topical view

The first step is to extract the candidate senses that are attached to the terms in the text. It can optionally produce scores associated with each candidate such as the *commonness* (likelihood of a term to designate the candidate sense), its *popularity* and *keyphraseness*. The second step *Relevance evaluation* reasons about the set of candidates and their scores as a whole and produces a relevance score for each candidate. The third step *Candidate ranking* takes a broader view and does not only consider the known candidates but also new ones that were previously unknown. New topics can emerge and a final ranking is produced.

Each step analyzes the document at a different level. The extraction step considers the candidates and their local compatibility in the text. The *relevance evaluation* step considers the candidates and their intern compatibility. The final view takes the largest view by also considering new candidates.

Our approach is based on SPOTLIGHT and its statistical model for the extraction step. The relevance evaluation and the candidate ranking use in combination a graph analysis and a modified version of the PageRank algorithm.

7 Candidate Extraction

The candidate extraction is performed by SPOTLIGHT [10]. Each candidate is evaluated on the local compatibility with its textual context by a score similar to the TF-IDF one and on the *commonness* (likelihood that the surface form is associated with the candidate sense). Their combination gives a final score which is normalized to 1.

A threshold on this final score controls how aggressive the extraction process is. A low threshold might produce a large proportion of bad candidates where as a too selective threshold might give very few candidates.

The dataset is composed of table of contents and syllabuses that have the particularity to contain very few textual context around each keyword. Headings such as chapters and sections are usually composed by a few number of terms and this situation hurts the performance of Spotlight. For this reason, the hierarchical structure of each resource is removed before the Spotlight extraction and is restored just after. The spotlight system hence annotates the entire resource instead of separate pieces of it.

Figure 5 was generated on fifty thousand candidates produced by Spotlight and shows that most of them have a score close to the extrema.

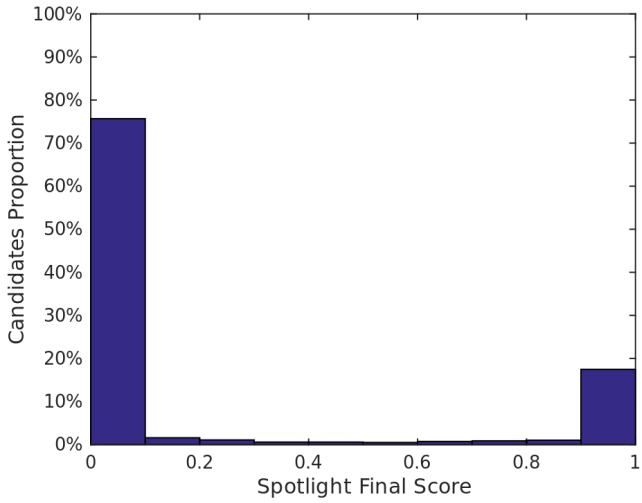


Figure 5: Distribution of the Spotlight final scores

Because of this special score distribution, we will later consider that a spotlight candidate is either likely or unlikely depending on which side of the 0.5 limit it is.

8 Graph-Based Relevance Evaluation

This module evaluates how relevant the extracted candidates are by analyzing their coherence between each other. For instance the set of topics {*algorithms*, *machine learning*, *artificial intelligence*} is more coherent than the set {*algorithms*, *machine learning*, *chemistry*} because chemistry is less related to machine learning and algorithms than artificial intelligence is. A graph-based approach is appropriate because it does reason directly on the interconnections between each candidate.

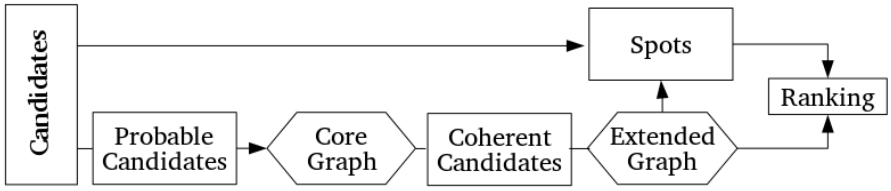


Figure 6: Candidate Selection Scheme

Figure 6 presents the architecture of our graph-based approach for the problem of relevance evaluation. The candidates previously extracted by SPOTLIGHT are filtered based on their final score. Candidates with scores above 0.5 are considered as probable and the other are temporarily ignored. Section 8.1 explains how a first graph is built using these probable candidates. Section 8.2 describes how we filter out elements of this graph that are weakly correlated. Then the task is either rejected or accepted as described in Section 8.3.

We illustrate the process at each step with the course *Advanced Data Structures in Java* from Coursera.org [5].

Table 8 displays the probable candidates extracted from the course with their spotlight score which is 1 for each candidate as expected. The entries in bold are clearly relevant for the resource and the other entries are mostly wrong.

8.1 Candidates graph

A first graph of topics and relations is generated using the Wikipedia link structure. A node is associated with each topic and edges are created between two nodes if their corresponding article share a link. The number of links in a Wikipedia article is often more than a hundred and their relevance can vary a lot. For instance, the Wikipedia page about algorithms links to the *Egypt* and to the *Alan Turing* page. The concept of relevance between any two nodes can be expressed as the number of links they share to common neighbors. This is the Wikipedia Link Measure WLM that was first used by Milne & Witten [3], later by Kulkarni et al. [5] and Mihalcea & Csomai [1] in order to quantify the inter agreement of any pair of candidate topics.

[5] https://www.coursera.org/learn/advanced-data-structures

Candidates	Spotlight scores
data structure	1.00
data	1.00
algorithm	1.00
heuristic	1.00
complexity	1.00
computer program	1.00
real number	1.00
wikipedia	1.00
jvm	1.00
golf course	1.00
about.com	1.00
social class	1.00
human	1.00

Table 8: Candidates with their scores from the extraction

We use the WLM adaptation of our previous work [9] which considers both the incoming and outgoing links to a Wikipedia page. The formula considers two nodes a and b, their set of in- out- links A, B and the total number of articles in Wikipedia W.

$$wlm\,(a,b) = \frac{log\left(\frac{|A \cap B|}{max(|A|,|B|)}\right)}{log\left(\frac{min(|A|,|B|)}{|W|}\right)}$$

In practice this adapted score varies between 0 and 16 after handling the corner cases. The score increases as a and b share more common neighbors. In practice we filter the links at 9.5 which removes very weakly related links.

Figure 7 displays the graph generated with candidates from the course *Advanced Data Structures in Java*. There is a clear cluster of highly connected concepts in the top right which fades out with topics less connected such as *Router (Computing)* or *Real Number*. In the bottom left, there are two pairs and one triplet of connected nodes that are irrelevant for the resource.

As expected they form a minor cluster since irrelevant candidates are less likely to be connected between each other. Some topics such as *Indium*, *Golf Course* and *Knot* are not connected at all. There is a clear correlation between the degree of a node and its centrality in the document.

8.2 Core graph

One drawback of the graph model is its computational complexity. There are numerous links on Wikipedia and following links increases the number of nodes exponentially. Wikipedia is known to be a small world graph [12]. The previous example suggested that we can reduce the size of the graph by removing topics

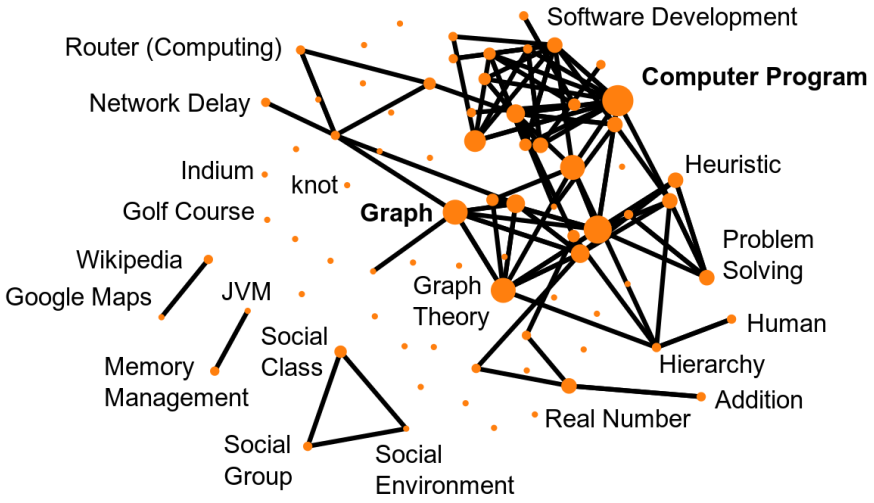


Figure 7: Candidates graph of the course *Advanced Data Structures in Java*

that are clearly irrelevant. The following steps are based on a smaller version of the graph that we call the core graph. Figure 8 shows the procedure used to produce it.

We alternate between a *selection* and a *refine* step until a stop condition is fulfilled. The condition can simply be on the size of the graph or any other condition on the nodes. The *refine* step first removes the nodes that are not connected or connected to only one node. Then it filters the biggest connected component. Figure 7 shows four connected components with their central nodes being respectively *Computer Program*, *Social Group*, *JVM* and *Wikipedia*. We assume that the relevant topics are correlated which makes it intuitive to filter the biggest connected component since it has the best chances to contain this set of relevant topics.

The stop condition indicates if the core graph looks acceptable. In practice we set a condition on the number of nodes since we want to ensure that the graph has a predefined maximum number of nodes. If the stop condition is not fulfilled, the *selection* procedure removes the weakest nodes from the graph before applying the *refine* step again. The *selection* step removes in priority the nodes that correspond to topics extracted from metadata and descriptions. Indeed we consider that some data has more priority such as the table of contents and syllabuses as suggested by Magdy and Darwish [11]. If the priority is the same for two nodes, we keep the most connected node.

Figure 9 shows the core graph. We see that it is a connected graph with many connections. There is one cluster around the *Computer Program* node and one other between the *Graph Theory* and *Algorithm* nodes. The two clusters are connected via the *Data Structure* and the *Data* nodes.

```
function refine(graph)
  remove dangling nodes recursively
  take the biggest connected component

function selection(graph)
  remove the weakest nodes from graph

procedure: core graph extraction
  refine(graph)
  while stop condition is not fulfilled
    selection(graph)
    refine(graph)
```

Figure 8: Core graph extraction procedure

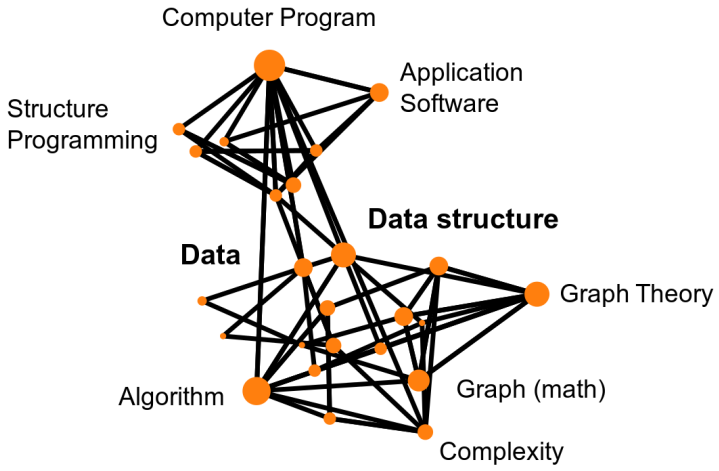


Figure 9: Core graph of the course *Advanced Data Structures in Java*

8.3 Task rejection

There exists several cases where the data is insufficient to produce relevant indexes.

1. The resource is not in the expected language
2. The wikifier fails to recognize too many candidate senses
3. There are too few relevant keywords in the resource

The last case is the most frequent because many authors produce table of contents which discuss more the administrative aspect of the course than its content. Figure 10 displays the table of content from the course *The Data Scientist's Toolbox* [6].

```
| Week 1: week-kind 1
| ..Week 1.1: week-kind 1
| ..Week 1.2: review week-kind 1
| Week 2: installing the toolbox
| ..Week 2.1: week-kind 2
| ..Week 2.2: review week-kind 2
| Week 3: conceptual issues
| ..Week 3.1: week-kind 3
| ..Week 3.2: review week-kind 3
| Week 4: course project submission & evaluation
| ..Week 4.1: course project: setting up accounts
```

Figure 10: Table of content of the course *The Data Scientist's Toolbox*

The candidates graph is shown in Figure 11. *Data* and *Data Analysis* are the two correct candidates that have been extracted from the course title. *Evaluation* and *Project Management* come from the table of content along with 5 other incorrect topics. The relevant concepts are too few and only form a chain which is removed by the *refine* step in the core graph creation since the chain endpoints are alway dangling nodes.

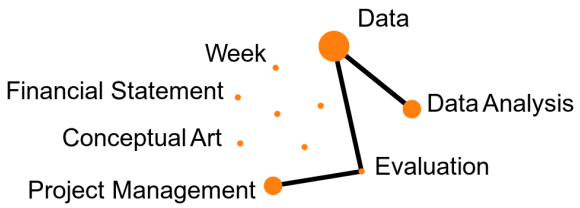


Figure 11: Rejected graph of the course *The Data Scientist's Toolbox*

The core graph becomes empty and the task is rejected. In practice we add the description of each course which is likely to contain enough relevant surface forms for the task to be accepted. Figure 12 contains the description of the course *The Data Scientist's Toolbox*. The candidates and core graphs are displayed in Figure 13. The number of candidates increased but the graph mostly contains chains of relevant information and a triangle which is enough to produce a nonempty core graph. This case shows that most of the relevant

[6] https://www.coursera.org/learn/data-scientists-tools

About this course: In this course you will get an introduction to the main tools and ideas in the data scientist's toolbox. The course gives an overview of the data, questions, and tools that data analysts and data scientists work with. There are two components to this course. The first is a conceptual introduction to the ideas behind turning data into actionable knowledge. The second is a practical introduction to the tools that will be used in the program like version control, markdown, git, Github, R, and Rstudio.

Figure 12: Description of the course *The Data Scientist's Toolbox*

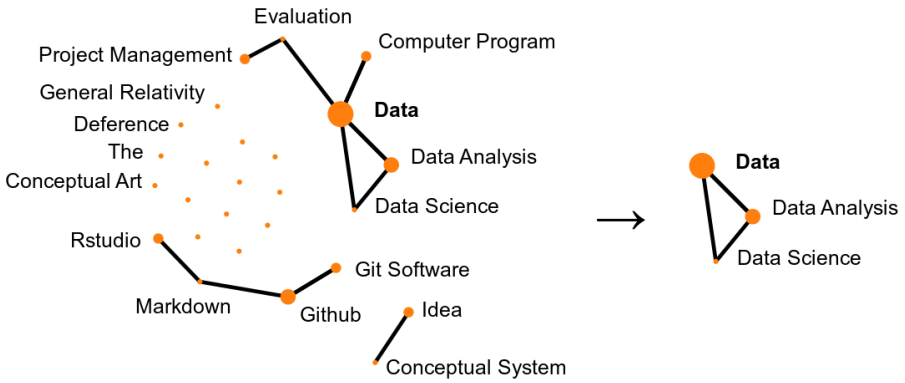


Figure 13: Accepted graph of the course *The Data Scientist's Toolbox*

candidates are not present in the final core graph because they were in the periphery of the main cluster or because the formed chains like *{Rstudio ~ markdown ~ Github ~ Git Software}*.

By definition the core graph should not contain peripheral or chains of nodes because we try to produce a cluster of closely correlated concepts. It is the role of the ranking process which takes a broader view and analyses an extended graph to evaluate how important the peripheral candidates are and rank them.

In our concrete scenario, the algorithm rejected five thousand resources from the total 34,371 resources. This represents 15% of the dataset. The rejection mechanism prevented an important amount of irrelevant indexes to be created.

9 Graph-Based Candidate Ranking

The last step produces a core graph which contains highly correlated topics that are central to the document. There are still two major issues that need to be addressed.

1. The core graph does not contain all the relevant topics that appear in the document.

2. There might be relevant topics that do not appear in the resource and that should be indexed.

The ranking step requires to have a broad view of interconnections between topics that appear in the text and topics that are closely related but did not appear in a previous analysis. It is the best moment to address these two issues and the algorithm does it by building an extended graph from the core graph.

The first issue is addressed in Section 9.1 and the second issue at ranking time as described in Section 9.2.

9.1 Extended graph analysis

The core graph is extended by retrieving two times the wikipedia page links. We first follow every outgoing link from any of the core graph node. Some of the links go to existing nodes and some to new ones. The new nodes are added to the graph. We then retrieve every outgoing link that goes from a new node to any node that is already present. Hence the extended graph contains all the nodes from the core graph and their direct neighbors with their links.

In practice, the extended graph ensures that the new nodes are correlated enough with the core graph nodes by setting a condition on the relevance of the links to follow. The algorithm only follows links with a WLM score above 10.5. We do not provide a picture of an extended graph here since they usually contain more than a thousand nodes.

Conceptually this graph corresponds to the directly related topics which could all be indexed but are not necessarily present in the document. However, we will only keep the central elements as indexes. In this case centrality is defined by considering all the nodes but not equivalently since it also involves whether the node appears in the document or not. There are three types of node.

1. A node which appears in the document and in the core graph.

2. A node which appears in the document and in the new nodes of the extended graph.

3. A node from the extended graph which does not appear in the document.

The centrality analysis will be biased with nodes from the first and second types because they are contained in the document. And *centrality* is by definition related to the *document*. The third type of nodes can be indexed if they are

Existing	Discovered
biopharmaceutical	prescription drug
oncology	epigenetics
clinical research	medicine
pharmacotherapy	
medical research	
pharmacology	
therapy	

(a) *Fundamentals of Pharmacology*

Existing	Discovered
subroutine	discrete math
calculus	software
limit of a function	periodic function
limit (math)	dimension
approximation	continuous function
math optimization	physics
computer science	
taylor series	
computational science	
limit of a sequence	

(b) *Single Variable Calculus - Functions*

Table 9: Example of new indexes discovered on two sample courses

ranked well enough by the graph analysis. We call the associated topics *suggestions* because there were not originated from the document. Table 9 displays some example topics which node is either of the first or the second type. The first column lists topics that exist in the core graph and the second column the new topics that were present in the Spotlight candidates but were not selected to be in the core graph.

Figure 14 shows the proportion of new topics in average in a sample dataset of 600 table of contents. The left boxplot corresponds to table of contents with core graphs of exactly or less than 10 nodes and the right boxplot on the other table of contents. The proportion of new topics does not depend on the number of topics in the core graph and is approximatively 30% of the number of core nodes. This percentage varies between 20% and 40% (the 25^{th} and 75^{th} percentiles).

9.2 Ranking strategy

We use the biased PageRank algorithm from Mihalcea & Csomai [1] that we used in previous works [9]. The centrality $S(n_i)$ of a node n_i depends on the centrality $S(n_j)$ of the nodes that link to it $n_j \in In(n_i)$ normalized by their outdegree $|Out(n_j)|$. Each node n_i has a bias $B(n_i)$ in contrast with the original PageRank algorithm by Brin & Page [7].

$$S(n_i) = (1-p) * B(n_i) + p * \sum_{n_j \in In(n_i)} \frac{1}{|Out(n_j)|} * S(n_j)$$

The random surfer interpretation of the recursive formula considers a random surfer which starts from a node and follows one of the outgoing link with a probability p or teleports to some node in the graph with a probability $(1-p)$.

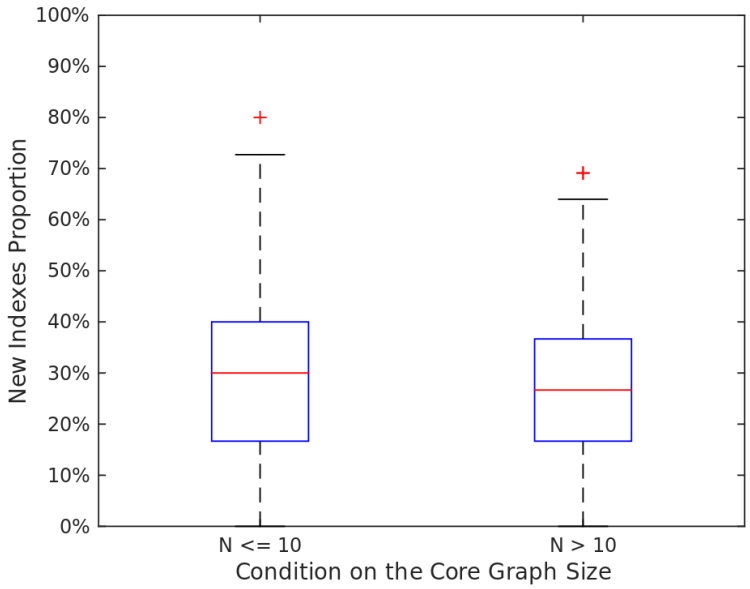


Figure 14: Proportion of new indexes discovered

There is an equal chance of following any of the outgoing links but the likelihood of teleporting to another node n_i depends on its bias $B(n_i)$.

In practice we set a bias of 1 for nodes that belong to the core graph and a bias of 0.5 for nodes that appear in the extended graph (but not in the core graph) and in the document since we are less sure about them. Other nodes have a bias of 0.

The PageRank score $S(n_i)$ is the measure of centrality that we use for each node. We produce indexes for the best N nodes which is set in practice to the number of Spotlight candidates contained in the extended graph. The PageRank scores are divided by $s_{max} = max\{S(n_i)\}$ such that the node with the biggest score has a normalized PageRank score $S'(n_i)$ of 1.

A final score $F(index)$ is attributed to each index which takes into account the centrality of its corresponding node n_i in the extended graph but also how sure we are about the index. In practice, the size of the core graph $size_{coregraph}$ is used as a measure of confidence and the final score is computed using

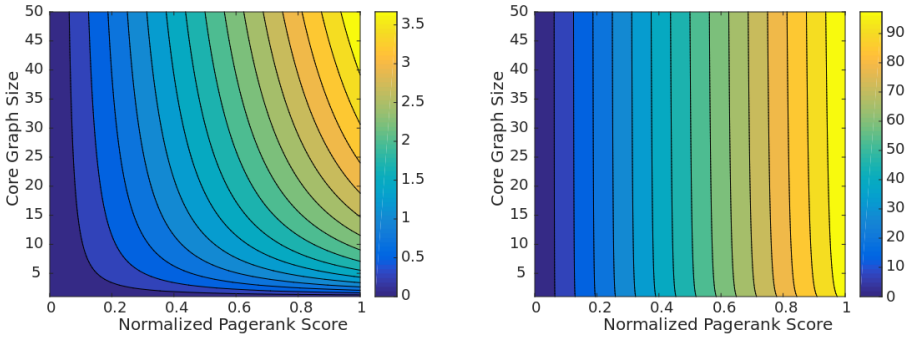
$$F(index) = S'(n_i) * log(size_{coregraph} + c)$$

The function is logarithmic on the size of the core graph and linear on the PageRank score. Its steepness can be controlled with the parameter c. A big c parameter gives a steeper curve and makes the function roughly independent

of the core graph size. Figure 15 shows how the final score varies for $c = 0$ and $c = 100$.

A small core graph of size 5 or below will have its scores between 0 and 1.5 for $c = 0$ and a medium core graph of 20 nodes will have its scores vary between 0 and 3. In contrast, there are no differences in the scores between a small and medium graph for $c = 100$. In this case, the scores will always vary between 0 and 100 independently of the size of the core graph.

In practice we set $c = 0$. Hence the score associated with the most central term (which as a normalized PageRank score of 1) is bigger by a factor of 2 between a core graph of size 5 and 20. In this sense indexes that are produced by small core graphs are disadvantaged since a small core graph is a sign of uncertainty.



(a) $c = 0$ (b) $c = 100$

Figure 15: Index scoring function $F(index)$ for different c parameter values

The ranking strategy allows any topic associated with a node in the extended graph to be selected as an index (i.e. even if it does not appear in the document). By setting the number of indexes created to a greater value than the number of topics which appear both in the document and in the extended graph (these are called *spots*), we are sure to add such nodes.

The multiplication factor parameter controls what is the number of indexes created. A multiplication factor of 1 creates a number of indexes equal to the number of *spots*. A multiplication factor of 2 creates a number of indexes two times bigger than the number of *spots*.

10 Indexes Distribution in Practice

We have seen above the entire process of index creation. We now analyze how the algorithms behave in practice on the dataset described in the beginning.

The algorithm produces in total 5 million indexes for a multiplication factor of 1 and nearly 25 million indexes for a multiplication factor of 5. Figure 16 shows the frequency of each index versus its rank in this frequency list for both multiplication factors. The scale is logarithmic in both rank and frequency and we see that the majority of indexes created are from very popular topics. The most frequent topic is associated with 638,300 indexes for a multiplication factor of 5 and 74,106 indexes for a multiplication factor of 1. However most of the indexes have a rank above ten thousand and are associated with less than a hundred indexes.

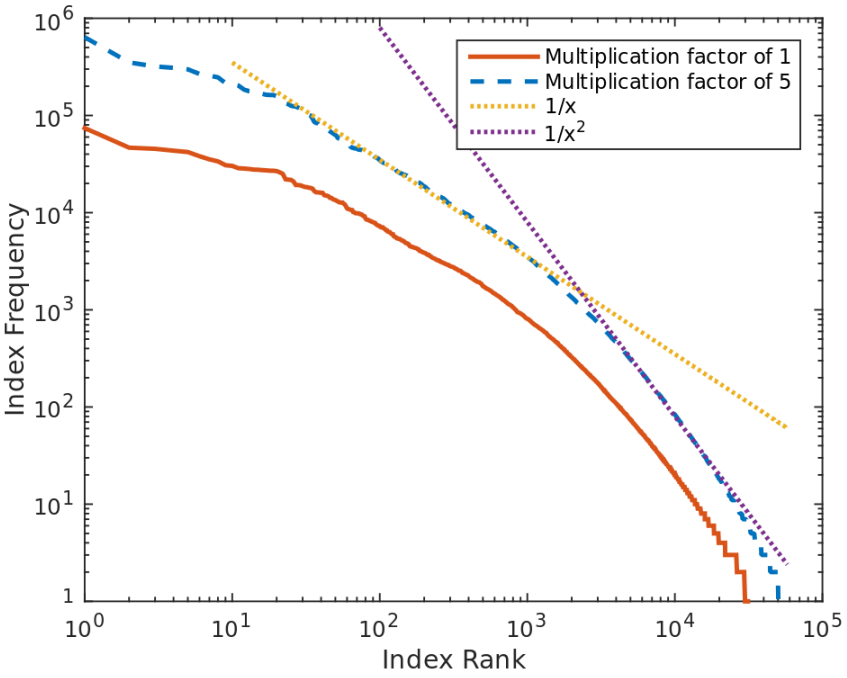


Figure 16: Indexes distribution on a log-log scale

This distribution is very similar to the Zipf distribution with two important parameterizations that appear with two dotted lines in the figure. For the first thousand indexes, the frequency of a topic is inversely proportional to its rank. For indexes with a rank between a thousand and ten thousand, the frequency of a topic is inversely proportional to the square of its rank.

The distribution of the indexes for a multiplication factor of 1(solid curve) is very similar to the distribution of the indexes for a multiplication factor of 5(dashed curve). However, the gap between them is bigger for the top ranked indexes than for the bottom indexes (with a rank bigger than ten thousand). The top ranked indexes are already enough frequent and it is not useful to produce more of them. In this sense a big multiplication factor is not useful since it produces more indexes that are central to many other graphs. This is why we set the multiplication factor to 1 for our experiment.

Table 10 shows the seven top indexes for each multiplication factor. We see that there are no common indexes in the two lists. Indeed, the multiplication factor tends to introduce a massive amount of indexes that are central to many graphs. Overall, the indexes ranking is slightly reordered.

Index	Amount
software	74,106
data	39,259
server_(computing)	39,243
computer_network	38,145
database	37,199
subroutine	28,865
web_application	27,686

(a) Multiplication factor of 1

Index	Amount
operating_system	638,300
programming_language	351,050
microsoft	319,606
microsoft_windows	309,323
java_(programming)	298,171
c_(programming)	267,569
linux	257,397

(b) Multiplication factor of 5

Table 10: Top indexes by frequency

11 Query Processing

We discussed above that the query processing differs in our approach compared to the standard model. Indeed, the user can input a combination of Wikipedia topics and textual terms that are not matched with any Wikipedia entity.

There is another particularity in our approach. When returning the results, we have to decide which part of each document to return. For instance there might be several chapters and sections from a common resource which matches the user input and the task is to return the most relevant chapters or sections. This additional processing is integrated in the standard *document ranking* step.

11.1 Particularities of the approach

Our approach considers the structure of each document. This structure is modeled using a tree. For instance sections from a chapter are leafs of this chapter node. In practice a document is indexed as many times as there are nodes in the tree.

In order to index a particular node in the tree structure, we take this node and its children into consideration. Then we extract only the candidates from this set of nodes. These candidates that are also contained in the extended graph of the resource are named *current spots*.

A root node is added. It represents the entire resource and contains the descriptions, keywords and document title. Figure 17 displays a sample structure with 2 chapters and 5 sections. The root node is added above the two chapters and is linked to the description, keywords and resource title. Indexing this resource should produce 8 sets of indexes. However there is only one core graph for the entire resource and the graph analysis runs on the same extended graph for each of the tree nodes. The only difference between each analysis are the biases attributed to each nodes in the extended graph. The biases depends on the type of nodes as described previously in Section 9.2.

1. Topics that are in the *current spots* and appear in the core graph have a bias of 1

2. Topics that are in the *current spots* but not in the core graph have a bias of 0.5

3. Topics that are not in the *current spots* have a bias of 0

Indexing each node in the tree gives different indexes even if the extended graph is the same because the *current spots* associated with each tree node is different. Hence the graph nodes are assigned different biases. One extended graph for each resource is sufficient to index all the nodes from its tree because the *current spots* of each tree node is a subset of the extended graph nodes by definition.

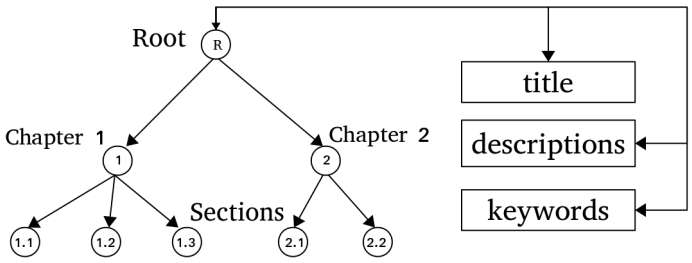


Figure 17: Scheme of a resource structure

11.2 Document ranking

The search engine has a simplified version of the *Boolean* information retrieval model. The user inputs a set of Wikipedia topics and we return the documents that contain at least one of these topics. In practice each document has several indexes that each have a score. For each document we sum the scores of indexes that are contained in the user query. We have seen in the previous section that a resource is indexed once and additionally as many times as the number of nodes contained in its tree structure. For instance a book with only 3 chapters will be indexed 4 times: one time for each of the 3 chapters and one time for the root node which corresponds to the entire resource with optional descriptions. Each time a resource is indexed, we say that the indexes refer to an *entry* of the document. Hence a resource might give indexes for multiple *entries* that all belong to it.

The user should not see multiple entries which belong to the same document separately. We typically want to merge the entries from the same resources. Table 11 displays a simplistic table of content with three entries and their only indexes. All of these entries belong to one common resource. If a user wants to retrieve all the documents that discuss *algorithms* and *sorting*, the system will retrieve chapter 1 and 2. The total score for the first chapter is 2 because the entry has two indexes that match with each a score of 1. The total score for the second chapter is 1 since it only discusses *algorithms* and since the index has a score of 1. However, the system will merge these two entries because they come from the same resource and the result should contain both chapter 1 and 2 since they are relevant to the user query. In practice we group the entries by resources and display to the user one result for this resource with the top three entries ordered by total score.

Figure 18 shows how the results are retrieved. The indexes are separated into two database tables. One contains only the indexes with the minimum information and the other contains the details about the associated resource. For instance, the labels of entries or the location of the resource on the web is contained in the details table.

The system does not know in advanced how many *entries* it needs to retrieve

Entry	Indexes (topic, score)
Chapter 1: Heapsort Algorithm	(Algorithm, 1), (Sorting, 1)
Chapter 2: Dijkstra's algorithm	(Algorithm, 1), (Dijkstra, 1)
Chapter 3: Databases	(Database, 1)

Table 11: Simplistic resource with 3 *entries*

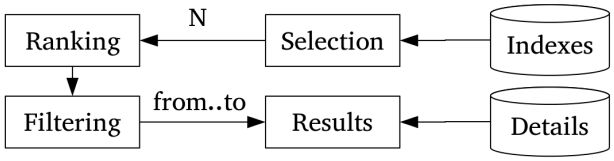


Figure 18: Entire query scheme

because they are later merged by their resource as described above. If a user wants the i^{th} result matching a query, retrieving i results is not enough since the $i-1$ first indexes might contain $n > 0$ entries that will be merged. In this case the system only has the $i - n < i$ first documents and still has to retrieve the next n ones. In practice we want to perform the fewest database queries for efficiency considerations.

The indexes table contains the strict minimum and we retrieve the indexes based on the topics from the user query. We set the primary key of the index table to the topic identifiers column for constant time access. Hence it is faster to retrieve a big number of indexes once than to retrieve multiple times small fragments of indexes.

The first step is to select the indexes that match the user query. The selection condition σ contains the disjunction of the topics from the user query and optional parameters. For instance $\sigma = \{\sigma_{topics}, \sigma_{type}, \sigma_{pagination}\}$ where $\sigma_{topics} = \{algorithms, databases\}, \sigma_{type} = books, \sigma_{pagination} = [1, 3]$ queries the top 3 books that discuss either *algorithms* or *databases*.

The selection step of Figure 18 limits the number of indexes retrieved from the index table to a constant N, bigger than the maximum number of documents that the system can return. The pagination parameter $\sigma_{pagination}$ refers to the ranks of the documents after being merged as discussed previously. This corresponds to the ranking and filtering step in the figure. Finally the details of the documents which correspond to the query are completed with the information from the details table. This approach has two advantages. It only makes 2 queries on two tables with constant time access and we can return the total number of results which makes pagination possible.

12 Performance

The complete process is composed of the eight following steps.

1. Download the pages and scripts associated with each resource
2. Execute the scripts and render each page
3. Extract the information from each page and save it into the JSON format
4. Extract all the candidate Wikipedia topics using SPOTLIGHT
5. Index each resource using our graph-based approach
6. Improve the consistency of each resource. e.g. add missing chapter or section annotations
7. Generate a plan to display each result
8. Save the indexes into the database

Downloading, rendering a page and extracting its information was one of the major costs of the entire process. These first three steps took around ten seconds for each resource and 95 hours in total for the entire dataset. Then we extract all the Wikipedia concepts that may appear in the text using Spotlight. This takes less than one hour in total.

Indexing the resources using our graph-based approach is the second major cost. It includes building the core and the extended graphs. And finally performing the PageRank algorithm for each node in the tree structure of the resource. The indexes are produced at the end of this step and saved in the resource JSON document. By limiting the core graph size to 25 and considering links with a WLM score above 10.5, this fifth step took around 6 seconds per resource. By parallelizing the computations, it took a total of 65 hours for the $34,000$ resources.

In the last steps we improve the look of the indexes when they will be displayed to the users. For instance Figure 19 shows how step 6 adds and corrects table of content chapter or section annotations. The same process is applied to courses and handles the *Module* and the *Week* elements.

Step 7 generates in advance a plan that explains how to display the results to the user. The plan indicates what are the best elements of the document to display given a user query. For instance, if a user queries documents about topics A, B and the best document D is associated with topics A, B, C then we want to display the best chapters and sections of document D that discuss topics A, B but not C. Step 6 and 7 took less than 30 minutes for the entire dataset which do not impact the global performance.

Finally we save the indexes into a database with a primary key on the topic of each index entry. This allows constant time access given a set of topics. The dataset produced five million indexes for a multiplication parameter of 1 and it took around 5 hours to import them into the database.

```
i: Introduction
  About us
  About the field
Advanced topics
  3: algorithms
  4: Advanced data structures
Part iii: Finally
  chapter 5 Discussion
    Controversies

Part i: Introduction
  Chapter 1: About us
  Chapter 2: About the field
Part ii: Advanced topics
  Chapter 3: algorithms
  Chapter 4: Advanced data structures
Part iii: Finally
  Chapter 5: Discussion
    Section 5.1: Controversies
```

Figure 19: Step 6: Improving consistency in the annotations

We estimate the total time to retrieve and index one resource to be around 18 seconds. Half of which is for getting the information and the other half for processing it.

Part III
Evaluation

The search engine is evaluated through a user study on 95 students from different fields in which we compare the results from our approach with the ones from the Bing.com engine and from a standard implementation of a full-text search engine. The search engine was published on the address https://wikichimp.com

Chapter 13 describes the experiment environment and the results are presented in Chapter 14. We conclude by a discussion of future works in Chapter 15.

13 User Study Description

The experiment consists of two steps. We first ask the user to perform some queries and evaluate their results and then to fill a feedback form. Figure 20 shows the results for a query about *Time series* and *Forecast*.

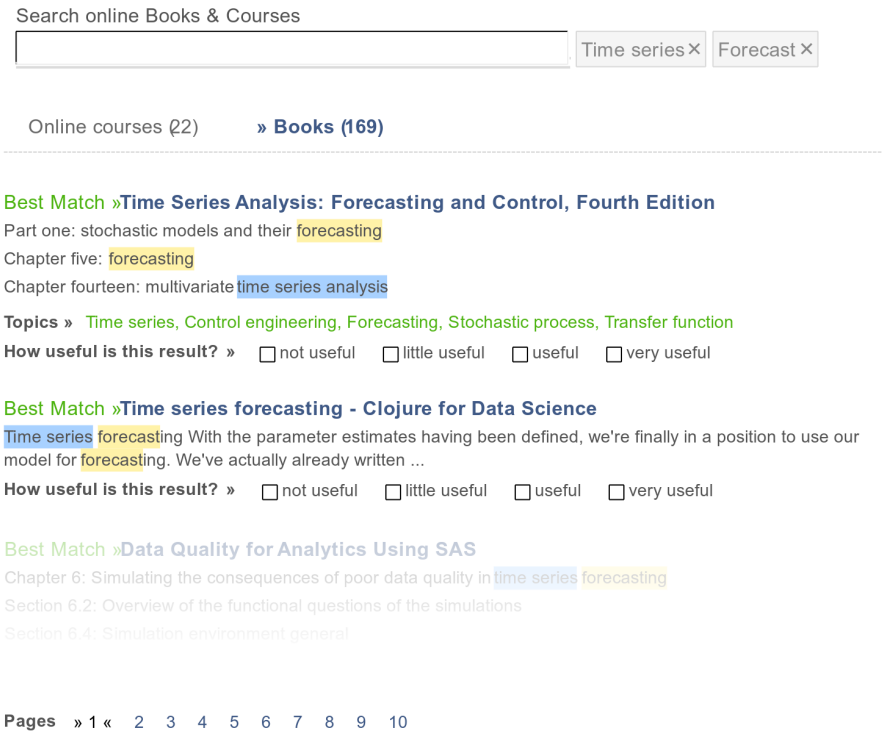


Figure 20: User Interface for browsing search results

The interface uses the resource title as a link which is prefixed by "Best Match" if the resource score is above a limit. Below each title, a snippet of text contains the best entries of the resource. For instance the first result in the figure is a resource named *Time series analysis: Forecasting and control, Fourth Edition* in which part one, chapter five and fourteen are particularly relevant.

The search terms are highlighted in yellow for the topic *Forecast* and in blue for *Time series*. Note that the highlighted text is not necessarily the same as the search term label since one topic is associated with multiple surface forms in our approach. For instance, chapter fourteen talks about "*Time series analysis*" and the engine considers this phrase to be associated with the *Time series* topic. The result is followed by a list of top topics in the resource. These topics correspond to indexes in the resource that have the biggest scores and are hence the most central ones. The last element of each result is a question form that the user

uses to evaluate how useful the result is. We will later refer to this score as the *usefulness* of each result entry.

There are two major types of results interleaved transparently. The first type comes from our graph-based approach and the second type are results from the Bing.com engine. It is important to note that their dataset and knowledge is not the same. We have 30 thousand courses and books from 5 predefined websites in the form of syllabuses and table of contents with optionally a description and some categories metadata. Our knowledge is equivalent to Wikipedia which we use for our analysis and for the Spotlight candidates extraction. In the other hand Bing.com indexes an unbounded number of different websites, type of pages such as PDFs and languages. We restricted Bing to return only English results from the same 5 websites as our dataset by setting appropriate bing API parameters. However we have no finer control on the results that are returned by Bing on these websites. In practice they were mostly equivalent but Bing returns sometimes PDFs results and links to videos.

Each pair of results contains one Bing and one graph-based result. The order is defined randomly but consistently. For instance the first and the second results of any search contains one Bing and one graph-based result but which result is in the first position is not known in advance. However the order is consistent: the same results in the same order are returned for the same query.

There is a third minor type of results which come from the Elasticsearch implementation of full-text search engine. These results complement our graph-based approach and hence they are returned always after our graph-based one. For instance a user query without any Wikipedia topic will not match any results in our engine and the Elasticsearch results will be used instead.

The second result in Figure 20 does not contain a list of top topics such as in the first result because Bing.com does not provide such information. We made two experiments. In the first experiment we hide these top topics and displayed them only in the second experiment.

14 User Study Results

The 95 students who participated to the user study did 259 queries. They evaluated the usefulness of nearly 700 results and clicked on 158 results.

At the end of the experiment, all of them compared the search results to the Google or Bing one. In addition, we asked the users whether the search engine gave enough results and whether they found the interface intuitive. These are two necessary conditions for the experiment to be relevant. 77% of them found that the engine returned enough results in most cases and 91% that the interface was intuitive. This last result shows clearly that the suggestion mechanism presented in Section 5.1 can be easily integrated without affecting the user experience.

In the following sections, we analyze the feedback collected from the user study.

14.1 User queries

One important aspect of the search engine is that it does not exclusively use the graph-based algorithm. Users can input raw texts that cannot be used by our approach because it requires that the user inputs at least one Wikipedia topic. We name such query elements *suggestions* because they are entered via the suggestion - autocompletion feature described in Section 5.1. Figure 21 shows the proportion of queries that contain at least one Wikipedia topic with the proportion of queries that return at least one result from the graph-based approach. The results are consistent between the two experiments with 72% of queries that contain Wikipedia topics and 52% that return results from our graph-based algorithm.

There are around $31,285$ distinct topics that are covered by the indexes for a multiplication factor of 1 and nearly $58,000$ if we set the multiplication factor to 5. This is a very small fraction of the 1.3 million topics available on Wikipedia(without counting categories, disambiguation pages and such). In fact it is less than 1% of the topics and it is surprising that we can answer more than half of the queries (marked with the blue line in the figure). It shows that the indexes produced are very relevant and that there are very few irrelevant topics.

Figure 22 shows the distribution of the queries by length. There is 700 queries that contain a single search term (Wikipedia topic or raw text) and 81 queries with two search terms. Globally, the search term length decreases logarithmically.

We presented in Chapter 10 the distribution of the indexes with respect to their frequency. It is interesting to know if the users search for topics that are frequent in our dataset and if producing more indexes with the multiplication factor helps to produce more results that will be consumed by the users. Figure 23 shows the indexes distribution curves previously seen and adds squares for indexes that match topics that were searched by users during the user study.

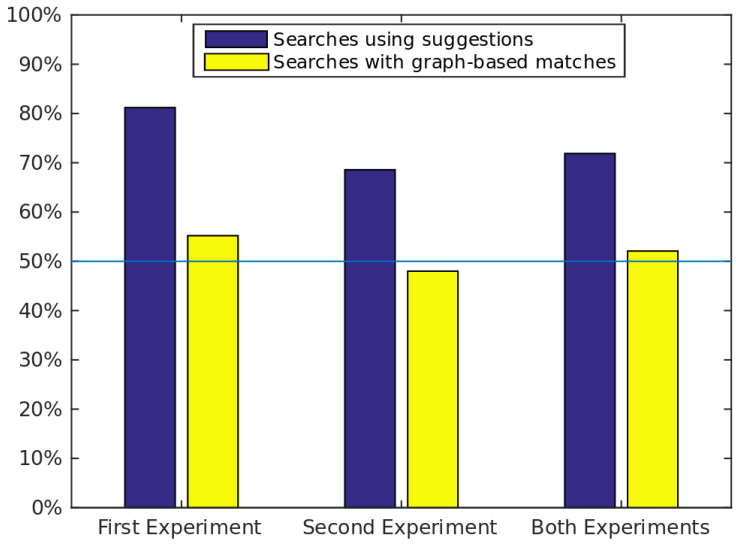


Figure 21: Proportion of users using the suggestions to express their queries

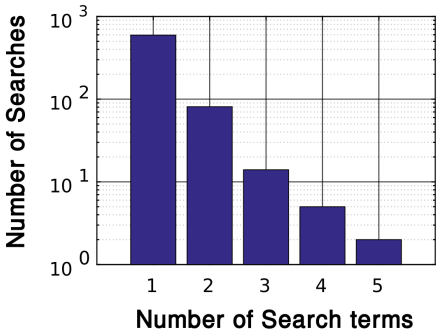


Figure 22: Number of search terms versus frequency on a log scale

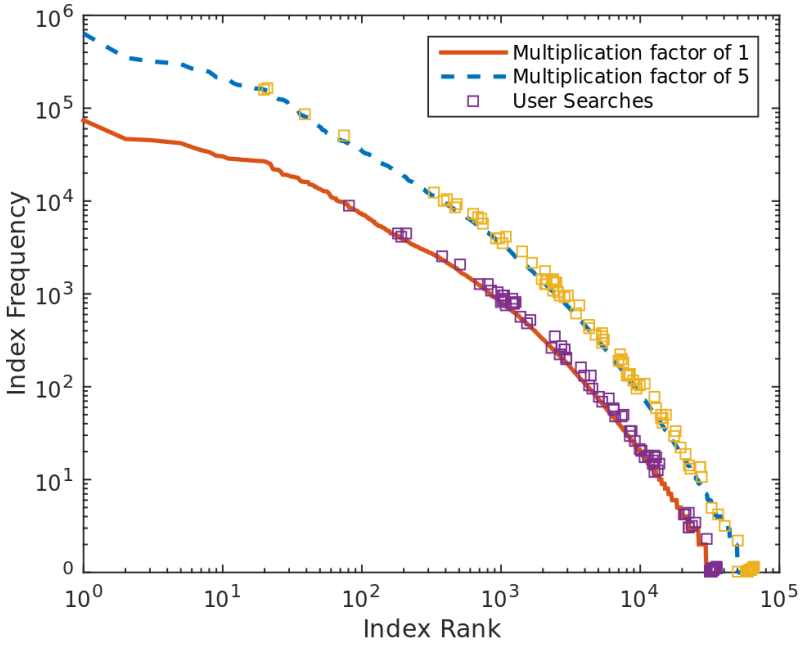


Figure 23: Indexes distribution with searches

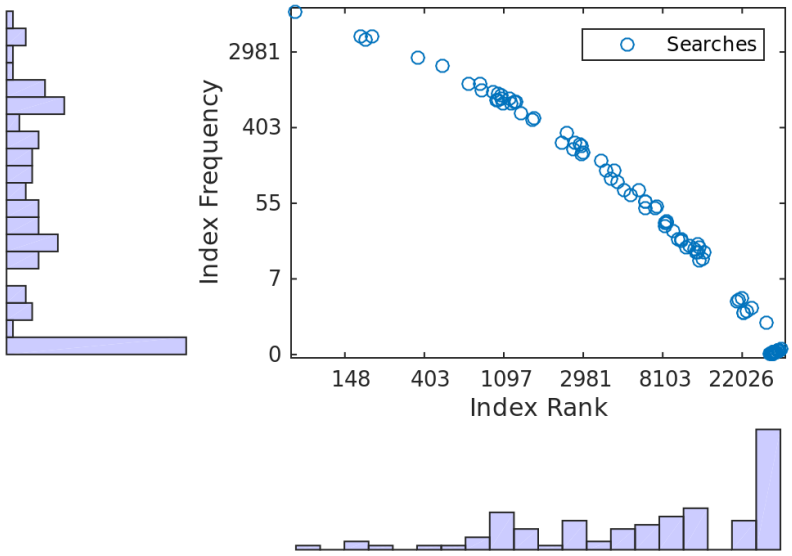


Figure 24: Searches distribution on the factor 1 curve from Figure 23

We see that most of the queries retrieve indexes with 10 to 2,000 indexes in the normal case and 10 to 5,000 for a multiplication factor of 5. The last part of the curve shows that there are many searches that have zero corresponding indexes. This is an important part of the search engine which may return some results that are close or fall back to a standard full-text engine as in our experiment. It is expected because the indexes distribution is heavy tailed: most of the topics have very few or no corresponding indexes.

Figure 24 shows the distribution of the searches with histograms on the previous index curve. We see that the indexes distribution is centered at topics with 300 indexes. This is a good indication that the searches get enough results in general.

Table 12 lists the top searches by popularity. We see that there are many domains covered by the dataset and that most of the searches that do not have any indexes are specific topics that are probably not covered by the dataset at all.

14.2 User clicks

As previously discussed we ran two experiments in which we simply hide, for the first experiment, or display, for the second, the five most central topics for each of our results. These topics are displayed in green in Figure 20. We observed that this minor change affects a lot the number of clicks on the results from the

Search term	Count
Machine Learning	5
Thermodynamics	4
LaTeX	4
Neuroscience	3
Astronomy	2
Biosensor	1

(a) Popular searches with matches

Search term	Category	Count
Brain-com. interface	Neuropsy.	2
Capybara	Animal	2
Game of Thrones	Series	2
AC72	Sailing	1
Alpha-amylase	Biology	1
Perovskite	Chemistry	1

(b) Popular search without matches

Table 12: Examples of user searches

graph-based algorithm. Figure 25 shows that there is nearly two thirds (the blue line) of the clicks that are on the Bing results in the first experiment in which we do not display the top topics. This contrasts with the second experiment in which case most of the clicks were on the graph-based results. Displaying such information does not impact the relevance of the results but it can improve the appeal of the results.

14.3 User results ratings

The users were asked to evaluate the usefulness of each results without knowing from which engine they were generated. Figure 26 shows the proportion of each rating for each type of engine.

There is an important difference in the proportion of not useful results between the engines. The full-text search engine has nearly 80% of them versus 30% for Bing and less than 10% for the graph-based engine. It is important to note that the full-text and the graph-based engine use the exact same dataset. We have observed a major difference between the full-text and the two other search engines. Bing and the graph-based engines seem to understand better the query. For instance "*time series*" will produce two tokens for the full-text engine and return results about "*time*" and/or about "*series*". We did not observe that behavior with the Bing engine which understands "*times series*" as a whole. Our graph-based approach will also not fail in this case because it knows the concept of *times series* since there exists a Wikipedia page for them. The difference between the Bing and our graph-based approach is the number of not useful results which are less frequent in our algorithm. These became *little useful* results for the graph-based results.

We discussed previously how the index scores are computed using a combination of the PageRank score and the size of the core graph. Figure 27 shows that there is a correlation between the usefulness score given by the user and the index score. Results that are better evaluated by the users have bigger index scores. This shows that the index score is a relevant parameter for ranking the results.

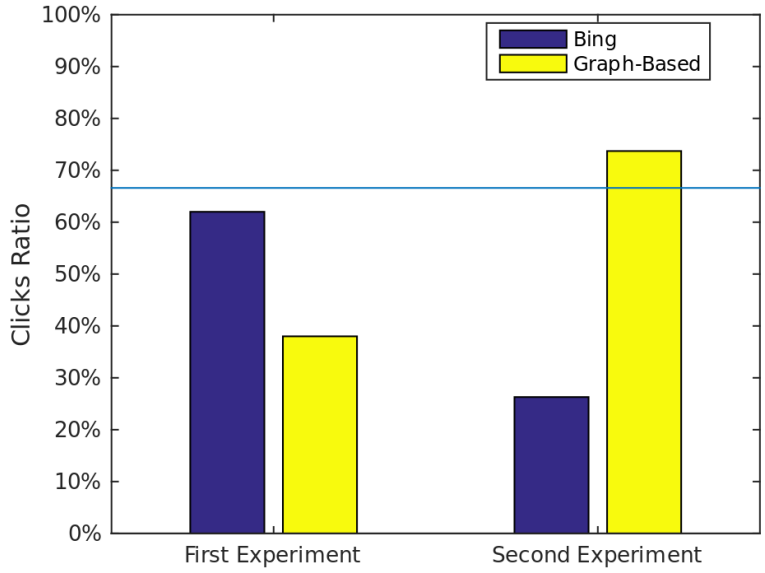


Figure 25: Proportion of clicks on the different types of results

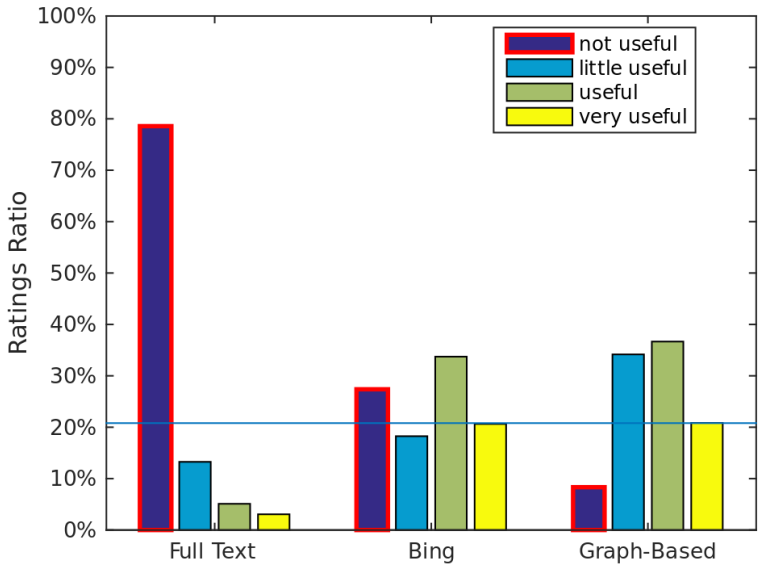


Figure 26: User feedback of the different type of results

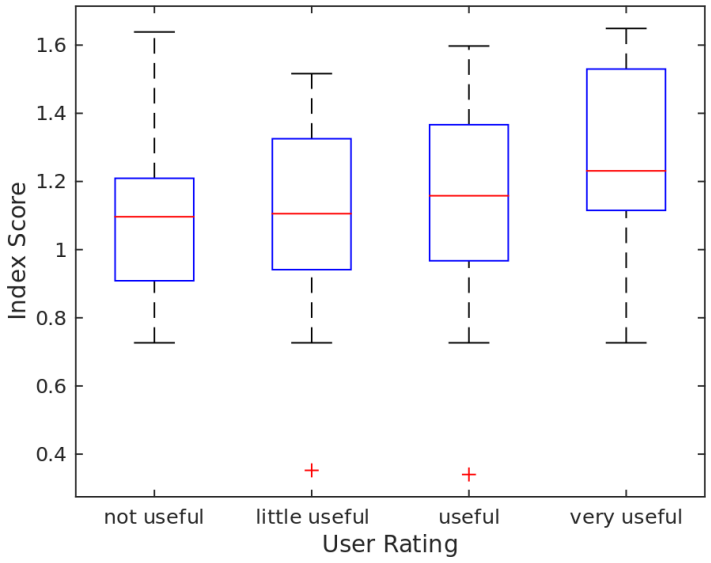


Figure 27: Correlation between the result scores and the user ratings

14.4 User satisfaction

At the end of the user study, we asked the users to evaluate the results of our search engine compared to most common other ones. The question was "When the website gives you results, are they good?" to which the user could answer

1. No, it is often worse than Google or Bing
2. It is equivalent
3. It is sometimes better, but not worse

Each user inputs different queries and hence gets different proportion of results from each engine. For instance if some query produces 3 pages of 10 results each. The first one might contain only 3 results from the graph-based algorithm and the rest from the Bing engine or the contrary might happen if Bing does not find many results for this search. Figure 28 shows the proportion of results that were returned to the user grouped by answers from the question above. We found that a majority of results (the blue line), for users that found that the results were worse than Google or Bing, were produced by the full-text search engine. In contrast users that find the results better than in Google or Bing had a majority of results from our graph-based approach. We see that better evaluations of the results correspond to less full-text results and more graph-based ones.

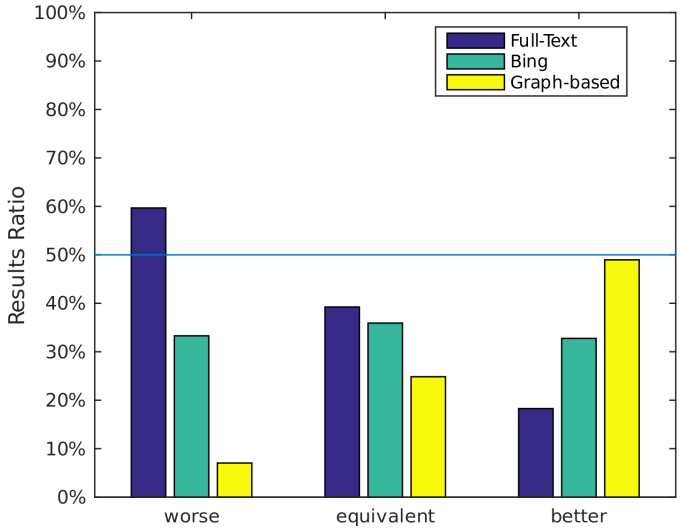


Figure 28: Proportion of search results by type versus user satisfaction

15 Future Work

The algorithm does not take into account any user feedback such as the usefulness scores or simply the clicks. Ideally we want the best results to be displayed in the top and the ordering to be dynamic in order to include the user feedback and raise in the top the most appreciated results. The ranking is based on the PageRank score and the nodes biases. Further analysis could investigate how the biases from the extended graph can be adapted to improve the indexes ranking using the user feedback.

We have seen that most of the queries only contain one search term. In addition, the users do not want necessarily to express their queries with an extensive list of Wikipedia topics. The boolean information retrieval model used by our approach cannot retrieve results that do not have indexes matching at least one of the Wikipedia topics from the query. There exists other models such as the vector space model which assigns weights to search terms. Further studies could analyze how to use the Wikipedia graph to transform any number of search terms into a query vector.

Finally, we observed that more than two thirds of the queries were expressed using the suggestion system. However the rest of the queries cannot be used, without preprocessing, in our graph-based approach because it relies on Wikipedia topics rather than on raw sentences. The use of actual wikifiers on the user query does not seem appropriate because user queries are typically short and lack context. On the other hand the vector space model could handle some wrong elements in the query if there are enough relevant topics to counter them. Further studies could investigate the generation of query vectors composed by weighted Wikipedia topics from a raw text user query.

16 Conclusion

We presented a graph-based approach to tag documents with topics from Wikipedia which is particularly adapted for structured documents with few textual context. The approach can index a reasonable amount of resources for use in a search engine with an adapted query interface which transparently transforms the queries into topics and concepts that appear in Wikipedia. We demonstrated that such an interface can translate the majority of the user queries in a concrete scenario.

The user study demonstrated that the results from our graph-based approach are equivalently useful as the top public search engines such as the Bing one. In some cases it produces better results and can display relevant information such as the top topics associated with each document.

Furthermore the algorithm was successfully able to rank the results accordingly to their relevance in the context of their document and reject the task when not enough correlation is detected. The graph-based approach can be used on any ontology and its speed can be adapted by setting the size of its underlying analysis graphs.

Once the documents analyzed, they are represented by a set of concepts which simplifies further analysis such as clustering, recommendation or indexing like in our search engine scenario.

List of Figures

List of Tables

References

[1] Mihalcea, R., & Csomai, A. (2007, November). Wikify!: linking documents to encyclopedic knowledge. In Proceedings of the sixteenth ACM conference on Conference on information and knowledge management (pp. 233-242). ACM.

[2] Medelyan, O., Witten, I. H., & Milne, D. (2008, July). Topic indexing with Wikipedia. In Proceedings of the AAAI WikiAI workshop (Vol. 1, pp. 19-24).

[3] Witten, Ian, and David Milne. "An effective, low-cost measure of semantic relatedness obtained from Wikipedia links." Proceeding of AAAI Workshop on Wikipedia and Artificial Intelligence: an Evolving Synergy, AAAI Press, Chicago, USA. 2008.

[4] Milne, D., & Witten, I. H. (2008, October). Learning to link with wikipedia. In Proceedings of the 17th ACM conference on Information and knowledge management (pp. 509-518). ACM.

[5] Kulkarni, S., Singh, A., Ramakrishnan, G., & Chakrabarti, S. (2009, June). Collective annotation of Wikipedia entities in web text. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 457-466). ACM.

[6] Coursey, K., Mihalcea, R., & Moen, W. (2009, June). Using encyclopedic knowledge for automatic topic identification. In Proceedings of the Thirteenth Conference on Computational Natural Language Learning (pp. 210-218). Association for Computational Linguistics.

[7] Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank citation ranking: bringing order to the web.

[8] Haveliwala, T. H. (2002, May). Topic-sensitive pagerank. In Proceedings of the 11th international conference on World Wide Web (pp. 517-526). ACM.

[9] Ouwehand, F., “Linking Pedagogical Content to Wikipedia”

[10] Mendes, Pablo N., et al. "DBpedia spotlight: shedding light on the web of documents." Proceedings of the 7th International Conference on Semantic Systems. ACM, 2011.

[11] Magdy, W., & Darwish, K. (2008, October). Book search: indexing the valuable parts. In Proceedings of the 2008 ACM workshop on Research advances in large digital book repositories (pp. 53-56). ACM.

[12] Dolan, S. "Six Degrees of Wikipedia." Retrieved June (2008).

[13] Baeza-Yates, R., & Ribeiro-Neto, B. (1999). Modern information retrieval (Vol. 463). New York: ACM press. See Figure 1.3 “The process of retrieving information” of Section 1.4 “The retrieval process”.