Extending WASP: providing context to a personal web archive

From re-finding to finding

Gijs Hendriksen

July 1, 2019

Supervisor:
Prof.dr.ir Arjen de Vries
`a.devries@cs.ru.nl`

Second reader:
Chris Kamphuis
`c.kamphuis@cs.ru.nl`

Radboud University

Abstract

Using WASP, a system for archiving, revisiting and searching through previously visited websites, users can have a better overview of their browsing behavior and find pages they remember visiting more easily. In this thesis, we introduce an extension to WASP that not only allows for searching through your browsing history, but also through pages related to your interests. It does so by crawling possibly relevant pages in the background, and indexing them if they are deemed relevant enough.

Contents

Chapter 1

Introduction

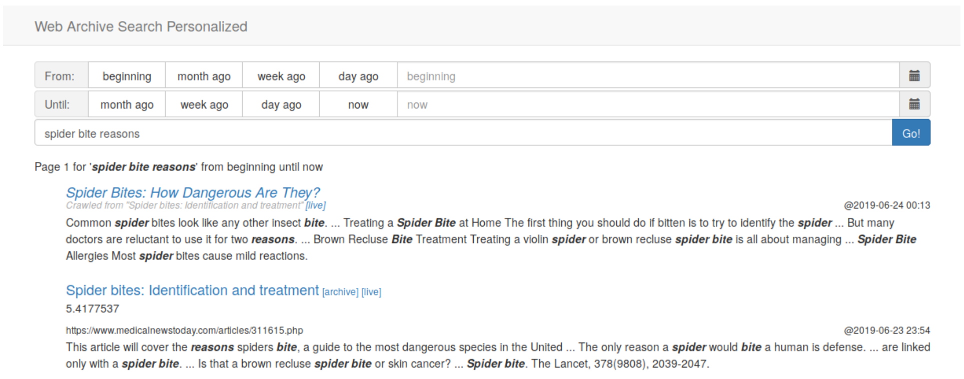
With the rise of popular search engines like Google, search has become better and better. We are almost at a point where Google knows what you want to search before you do. However, as this technology developed, the need for data protection arose with it. With large companies in control of, among many other data, your browsing behavior, the matter of privacy is addressed more and more.

In 2018, WASP (Web Archiving and Search Personalized) was developed, mainly to provide more insight into your browsing behavior. It allows the user to archive visited websites in a local database, and search and revisit these pages later on. As it stands now, it mainly fulfills a purpose lacking in mainstream browsers: full-text search in your browsing history. Google Chrome, Mozilla Firefox and Microsoft Edge - the most popular modern web browsers - only store some metadata of the page, like the title, URL, and the date on which the web page was visited. The functionality was even removed intentionally from Chrome in 2013, due to performance issues.[1]

Since WASP only searches through local, offline archives of visited web pages, its functionality can be compared to that of desktop search, albeit with a limited purpose. Where regular desktop search is concerned with all kinds of data sources on the user's computer, like files, applications and emails, WASP limits itself to the user's browsing behavior in a more comprehensive and insightful manner.

In this thesis, we introduce an extension to WASP which allows the user to search through other documents on the Internet relevant to their interests. It does so by using the user's archived browsing history to crawl and index new,

[1] `https://bugs.chromium.org/p/chromium/issues/detail?id=247415`


Web Archive Search Personalized

From:	beginning	month ago	week ago	day ago	beginning
Until:	month ago	week ago	day ago	now	now

spider bite reasons

Page 1 for '***spider bite reasons***' from beginning until now

Spider Bites: How Dangerous Are They?
Crawled from "Spider bites: Identification and treatment" [live] @2019-06-24 00:13
Common ***spider*** bites look like any other insect ***bite***. ... Treating a ***Spider Bite*** at Home The first thing you should do if bitten is to try to identify the ***spider*** ... But many doctors are reluctant to use it for two ***reasons***. ... Brown Recluse ***Bite*** Treatment Treating a violin ***spider*** or brown recluse ***spider bite*** is all about managing ... ***Spider Bite*** Allergies Most ***spider*** bites cause mild reactions.

Spider bites: Identification and treatment [archive] [live]
5.4177537
https://www.medicalnewstoday.com/articles/311615.php @2019-06-23 23:54
This article will cover the ***reasons*** spiders ***bite***, a guide to the most dangerous species in the United ... The only reason a ***spider*** would ***bite*** a human is defense. ... are linked only with a ***spider bite***. ... Is that a brown recluse ***spider bite*** or skin cancer? ... ***Spider bite***. The Lancet, 378(9808), 2039-2047.

Figure 1.1: A screenshot of WASP with the added functionality. The first result was crawled in the background, and the second result is a web page the user visited themselves. Note that the crawled result mentions it was found by crawling from the other document.

potentially relevant documents. In doing this, the modified version of WASP allows users to not only re-find documents they have visited previously, but to find other relevant documents entirely.

An example of the extended functionality can be found in figure 1.1. It shows two results of a search query, one of which was visited by the user and one that was crawled in the background. Since it might be useful to the user to know from which page a new document was crawled, we also display the origin of such a crawled document whenever it is returned as a search result.

By introducing this extension, we enable WASP to use online functionality as well, by retrieving and indexing other pages on the Internet. As a result, the modified version of WASP is no longer purely an offline system like desktop search. Rather, it has become a hybrid of both offline and online search, as some of the documents it serves are archived locally and other documents are retrieved online. Most importantly, all the data on the user's browsing history is still saved locally, and used to determine which documents could prove useful to the user's interests. This way, users can enjoy predictive search from a localized search engine, without having to depend on a third party to gather and analyze your browsing history.

Chapter 2

Preliminaries

2.1 WASP

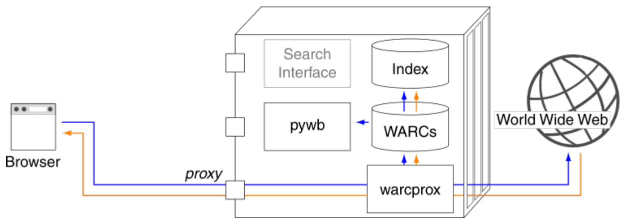
The most important part of this research is built on top of the existing infrastructure of WASP (Web Archiving and Search Personalized) [4]. Currently, WASP enables three core functionalities: archiving visited pages, searching through archived pages and revisiting these pages. Figure 2.1 shows the infrastructure of WASP and how it enables said functionalities.

Initially, WASP is used as a proxy service, silently reading all HTTP requests and responses. These responses are archived in WARC format and indexed in an Elasticsearch[1] index. Then, when the user wants to search their archive, they can head to WASP's simple search interface, through which they can query the Elasticsearch index for contents and archive date. Finally, for each of the query results, the user can choose to open their archived version, which is served by a `pywb`[2] service.

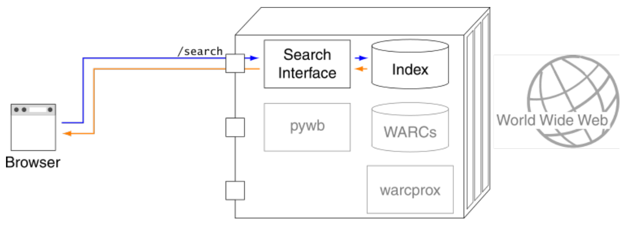
2.2 Crawlers

Crawling is the process of starting with a set of seed URLs, downloading the corresponding web pages, extracting all URLs out of these crawled pages and then following these links recursively. Much research has been done in the field of crawling, ranging from efficiency to scalability and even politeness.

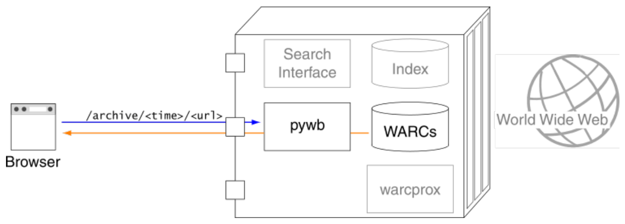
[1]`https://www.elastic.co`
[2]`https://pypi.org/project/pywb/`



(a) Archiving a visited page



(b) Searching through archived pages



(c) Revisiting an archived page

Figure 2.1: The infrastructure of WASP. (a) depicts the proxy functionality, where any website that is visited is also archived and indexed. (b) shows the search functionality, where a user can search their browsing history in the index of archived pages. (c) depicts how users can use `pywb` to revisit archived pages.

For the extension of WASP proposed in this thesis, the most important concept to keep in mind is the *crawl ordering policy.* The crawl ordering is the ordering in which the crawler follows the retrieved URLs. There are many ways to define such an ordering. The most simple and naive way is to use a basic breadth-first search to crawl pages in the order they are discovered. However, there are other, more effective crawl orderings that ultimately lead to a corpus better suited for a certain use case.

For instance, crawlers that are most interested in highly reliable pages can order the pages to crawl based on the highest PageRank [8], or the page with the most incoming links, both of which perform better than the simple breadth-first strategy [1]. In the case of topical crawlers that should only crawl pages on a certain topic, links promising to be relevant to this topic can be crawled first. In [9], several methods are explored to determine effectiveness of different crawl ordering policies for topical crawlers. Most of these depend on the context of a certain link, i.e. the words surrounding it, and feed this context into a classifier trained on the topic at hand. This way, they can predict the relative usefulness of a given link, before actually retrieving the corresponding document.

2.3 Document similarity

There are many ways to compute the similarity between documents. These methods can be separated into two main categories: lexical analysis and semantical analysis.

2.3.1 Lexical analysis

In lexical analysis, documents are defined by the actual words in it. In the well-known bag-of-words model, a document is represented as a 'bag' of all words in it, combined with the amount of times each word occurs in the document. The order of these words is disregarded. When comparing the similarity between two documents, one only has to compare the similarity between their bags of words. This can be done naively by, for instance, computing the Jaccard similarity between the two bags (the fraction of words that the two documents have in common).

A better approach is to extract certain features out of the bag of words, for instance by selecting the words that tell most about the document at hand. This can be done by computing the *tf-idf* value of each term, which is the combination of a word's term frequency (*tf*) and its inverse document

frequency (*idf*). The term frequency describes how often the given word occurs in the current document, favoring the more frequent terms. The inverse document frequency measures the inverse of the amount of times the given word occurs in all available documents. Thus, it scores highest for terms occurring rarely, and lowest for terms occurring in each document (most notably, stop words like 'the', 'and'). By combining the two, you can find the most important or defining words of a document, as a high *tf-idf* means a word occurs very frequently in the current document, but is not common in other documents.
Using *tf-idf* weighting, you can extract the N most important terms out of each document (i.e. the terms with the highest *tf-idf* value) and represent them as an N-dimensional vector. By computing the cosine similarity between two of these term vectors (the angle between both vectors), we can measure how similar the term vectors of two documents are. Consequently, we can give an estimate on how similar the documents are.

Lexical analysis can be extended by taking all bigrams or, generally speaking, n-grams in a document. An n-gram is a sequence of n consecutive words in a document. Therefore, n-grams allow for more context with each word, making it easier to distinguish small semantic differences between different usages of the same word. The total set of n-grams of a document is also called a w-shingling.

Finally, another key concept of lexical analysis is the process of 'stemming'. With stemming, each word is reduced to its 'stem'. For instance, 'working' should be reduced to 'work', 'books' to 'book', and 'are', 'am' and 'is' should all be reduced to 'be'. This way, the fact that a word occurs in a different form (singular or plural, etc.), does not impact the comparison between different bags of words or term vectors.

2.3.2 Semantical analysis

The main drawback of lexical analysis is the inability to distinguish meaning between phrases. For example, look at the phrases 'I read a book' and 'I book a flight'. Lexical analysis would not be able to differentiate between the noun 'book' in the first sentence and the verb 'book' in the second. In order to distinguish meaning between different usages of the same word, we have to take a look at the semantics of words in a given context. That is where semantic analysis comes in.

Semantic analysis focuses on finding the meaning of a word, phrase or document. This can be done in several different ways. One of the first things that comes to mind when determining the meaning of a term is the role it plays in the phrase. For instance, in the example phrases above, does 'book' occur as a verb, a noun, or something else? Assigning such a word class to each term in a phrase or document is called *part-of-speech* tagging. Several good implementations of part-of-speech taggers exist, like the implementation using Hidden Markov Models as described in [5].

Another common concept is to somehow represent the meaning of a word or phrase as a vector of numbers. An example of this is the representation as *word embeddings* and their implementation in Word2vec [7], where a neural network is trained to predict one or more words given a certain context. Later on, any new word can be fed into this neural network, along with its context, and the network's output features make up the word's vector representation.
This analysis has been extended to full phrases, as well, where the meaning of a sentence is determined by looking at the sentences surrounding it. And, by taking paragraphs into account, even entire documents can be reduced to a single vector encapsulating the document's meaning [6].

Chapter 3

Research

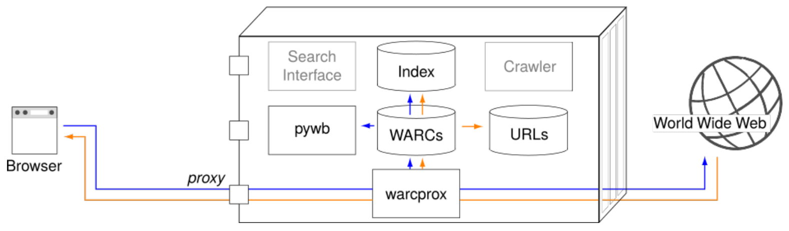
The final result of this research is an extension to WASP, which will crawl and index documents relevant to the user's interests in the background. In order to achieve this, we slightly amended the infrastructure of WASP and the way it handles any newly indexed pages. The new infrastructure can be found in figure 3.1.

The extension we developed consists of two main parts. The first part is the crawl selection, which decides what URLs will be crawled and in what order. This process will be explained in more detail in section 3.1. The second part is the relevance measurement, i.e. the part that actually retrieves the documents and determines whether a retrieved document is relevant to our user's interests. The relevance measurement will be handled in section 3.2.

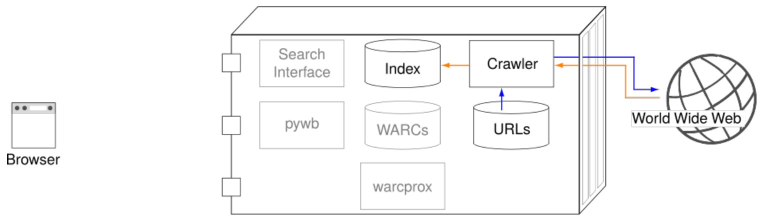
3.1 Selecting pages

As mentioned, the first part of our extension to WASP is determining which pages to crawl and compare to our index. In the same way that a normal web crawler (in the case of a general search engine, for instance) has one or several starting points from which to crawl, our own application should somehow know which pages to process in order to find relevant documents.

To do this, we assumed the internet has a certain degree of data locality. In other words, we used the documents we had already indexed as starting points for our crawling operation, under the assumption that web pages referenced in a certain document are in some way related to said document. Logically speaking, this makes sense, as we see many examples of references



(a) WASP now also extracts URLs out of each document and stores them



(b) Crawling possibly relevant pages in the background and indexing them

Figure 3.1: The new infrastructure of WASP. (a) depicts the extra step when indexing new documents, where any URLs and their contexts are extracted and stored for later use. (b) depicts the new background crawling functionality, where the previously stored URLs are crawled, checked for relevance and possibly indexed into Elasticsearch.

to relevant pages in different areas of the internet: answers to a question posed on any forum might contain links to background information or references to other relevant questions or answers, wiki articles reference other articles in the same category under the header "See Also", and blog posts sometimes mention other blog posts about the same topic. As a result, we added an extra operation to the indexing step, which extracts all hyperlinks from the document's anchor tags and stores them for further usage, along with a reference to the document from which they were extracted. This extra operation is visualized in Figure 3.1a.

However, the number of resulting links grows increasingly fast when we add more documents to our index. Consequently, it is near impossible to retrieve all these pages, compare them to our current data and determine whether they are similar enough to keep in the index. Moreover, WASP was designed to run locally on a single computer with basic hardware, meaning we have limited memory, processing power and bandwidth available. Taking this into account, we experimented with different heuristics for selecting the most promising link. In order to find this most promising link, we used the context in which the link was found in one of our indexed documents. This context consisted of the words inside and surrounding the anchor tag of said link. Using the Jericho HTML parser we could easily extract this surrounding text and store it alongside the other information mentioned above.

In every iteration, our background crawler reads from this list of possibly relevant links and matches the corresponding context to the existing index. The entry that scores the highest in this query is the entry that will be used for further retrieval and evaluation. By entrusting the actual scoring to Elasticsearch, we make sure the context words are preprocessed in the same way as the indexed documents, allowing for a better search performance. After experimenting with different context lengths, we found that we can predict which documents are likely to be relevant the best when we use a context of 60 words. More on these experiments can be found in section 5.2

3.2 Determining relevance

Now that we have a way to find the most promising link, we have yet to decide whether the corresponding page is relevant or similar enough to our existing index, before we decide to keep it. We start by actually retrieving the document and making sure it is an actual HTML document instead of another web resource like an image or a PDF document. After all, WASP is meant to store visited and relevant web pages, meaning we only want to parse and store HTML documents.

The next step is the removal of any `script` or `style` tags from the document, as they do not provide any information on the document's contents. They merely indicate how it handles user interaction and how the page is formatted, respectively. We remove all content from the document that is invisible to the user, such as hyperlinks, images, page structuring and all other HTML tags, in order to give us the raw content visible to the user.

Finally, we match this raw content to the current index by using Elasticsearch's `more_like_this` (MLT) query[1]. The purpose of the MLT query is, simply put, to find one or more documents in the index like the query document. It does so by computing the terms of the query document with the highest *tf-idf* value and using these as a query on the index. The main advantage we obtain by using an out-of-the-box Elasticsearch functionality is that Elasticsearch takes care of the tokenization of the document into words, stemming, removal of stop words, and any other basic preprocessing we might want to do before querying against our index.

[1]`https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-mlt-query.html`

However, even though this is a step in the right direction, finding documents comparable to our query document does not necessarily mean the query document is relevant to our interests. Take, for instance, the situation in which our index consists of a large amount of Wikipedia articles, and our query document is a Wikipedia article as well. In this case, the MLT query will see the other articles as (remotely) similar and return them, but their contents might be far from relevant to our query document.

The score value returned by the MLT query is unfortunately not normalized, meaning it does not allow us to draw a conclusion about our query document's degree of relevance. Instead, if we want to set a certain threshold for 'similarity', i.e. a particular minimum score for the most similar documents, we have to compute a custom, normalized score ourselves. We do so by selecting the most relevant results returned by the MLT query, and computing the cosine similarity between each of these documents and the query document. The actual threshold and amount of MLT results to use were determined after experimenting with different values, which is explained in detail in section 5.1. The cosine similarity is defined using the following formula:

$$sim(A,B) = \frac{A \cdot B}{||A||\,||B||} = \frac{\sum\limits_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

In our case, A and B correspond to the term vectors of the query document and the document it is compared against. These term vectors consist of the ten terms in the document with the highest *tf-idf* value, where the *tf-idf* value is defined as follows:

$$\begin{aligned} tf(t,d) &= \sqrt{freq(t \in d)} \\ idf(t) &= 1 + \log\left(\frac{|D|}{|\{d \in D \mid t \in d\}| + 1}\right) \\ tf\text{-}idf(t,d) &= tf(t,d) \cdot idf(t)^2 \end{aligned}$$

The *tf-idf* weighting scheme used here is taken from the Lucene Practical Scoring Function[2]. By choosing this weighting scheme, we make sure our custom scoring functions are in line with the scoring functions used by Elasticsearch.

[2]`https://lucene.apache.org/core/7_2_1/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html`

As we did with the context lengths, we also experimented with the amount of MLT results to use when performing our custom similarity measurement. On top of that, we also investigated whether to compare the average cosine similarity for the set of MLT results to a certain threshold, or whether we wanted all of the similarities to be above this threshold. More on this can be found in section 4.2.

Finally, after computing the similarity degrees of each of the Elasticsearch MLT results and making sure they are all above the preset threshold, we index the crawled document so it can be found by the user. However, it might still occur that this document is not exactly in line with the user's interests due to a false positive slipping through our similarity detection. Therefore, we do not extract any URLs out of the crawled document for further crawling. For the same reason, we do not include the crawled documents when comparing a new candidate to the index, so the results are not biased towards a topic only found in crawled documents.

Only when the user actually visits the corresponding web page will the document be fully indexed and archived, as a page visit indicates the user's actual interest. This is also the moment that the document is first available through `pywb`.

Chapter 4

Experimental Setup

As we discussed before, there are two main parts to this extension to WASP: the crawler and the relevance measure. Both of these have been tested extensively to verify their effectiveness. In this chapter, we will discuss the data sets on which we tested both components and the metrics we used to evaluate their performance. In the next chapter, we present our findings and draw the corresponding conclusions.

4.1 Data sets

4.1.1 Custom data set

In order to mimic actual user behavior as closely as possible, we created a small sample set supposed to represent actual user interests and pages on these topics. As a simple proof-of-work, we took a data set of ten topics, ranging from 'Soccer' to 'Ducks' and found ten documents on each of these topics, ranging from Wikipedia pages to news articles and Q&A forums. This data set was mainly used to show that our proposed solutions actually worked, but it was clearly too small to derive any actual statistics from. As we thought it was infeasible to expand this data set to the point it was actually large enough, we decided to look into some other, already existing data sets as well.

4.1.2 ODP data set

The Open Directory Project data set[1] contains a large amount of URLs, all annotated with the corresponding category. For instance, there is a category `Top/Sports/Soccer`, which contains all pages talking about soccer. Note that this notion of category differs slightly from the subject of a document, which we discussed earlier. However, we argued that the principle of similarity checking is the same in both instances. In other words, a document in a specific category is likely to be similar to another document in that same category, just like documents with specific subjects are likely to be similar to other documents on that subject. Therefore, we used this data set and assumed the results of this experiment would be comparable if we executed it with subject annotations instead of categories.

Some of the categories in the ODP data set only contain a very small set of documents, for instance when the category is very specific. An example is the category `Top/Sports/Soccer/UEFA/England/Non-League/Local_Leagues/`. Evidently, this category is unlikely to contain many documents, as it is concerned with a very specific group of websites. However, in the context of document subjects, we would rather have a broader subject such that we can actually compare different documents on the same subject. Therefore, we made sure any categories we select during our experiment are broad enough. This was done by merging any categories with too few documents and a common parent category into this common category. This way, the category on English local soccer leagues mentioned above would be merged with, for instance, `Top/Sports/Soccer/UEFA/England/Clubs` into a common category `Top/Sports/Soccer/UEFA/England/`. And if this new, combined category is still too specific, we can merge it even further up the category hierarchy, ultimately resulting in the final category `Top/Sports/Soccer/UEFA`.

Eventually, we ended up with a list of all categories containing at least a thousand documents, as we figured this number would allow us to perform a good statistical analysis. Finally, we also had the problem that some categories were too broad. Take, for example, the category `Top/Sports`. Many different documents would fall in this category, even though, for instance, a soccer fan might not necessarily like hockey. In this case, though, both subjects would be labeled relevant. Therefore, we decided to drop the categories that were too large, setting the limit at 10.000 documents per category.

In the end, we did not use the full ODP data set for our experiments. Instead, for each experiment we extracted a random subset of 50 categories

[1]Formerly hosted at `http://www.dmoz.org`. However, as of March 2017, the project is no longer maintained. The data set we used was collected through an archived version of the website.

of the ODP data set and a random set of 100 documents per topic. The main reason for doing this is to keep computation time down, as all documents had to be crawled first (the ODP data set only contains URLs, not the actual documents) and all experiments were executed on a single machine. Besides, the first set of experiments was only used as a proof-of-concept that the similarity measure worked. After confirming this, we could still scale up the experiments to prove the effectiveness further.

4.1.3 TREC Web Track data set

The final data set we used was the TREC (Text REtrieval Conference) 2014 Web Track data set[2]. The TREC data set contains several queries like 'identifying spider bites', along with the URL of web pages on which the answer can be found. Since different pages that answer the same question discuss the same subject, this data set is a lot more comparable to the actual usage data we are trying to recreate than the ODP data set. Thus, it should be a lot more suitable to use in our experiments. Moreover, we know for a fact that the pages we use contain some content on the subject at least, since they discuss an answer related to said subject.

The 2014 Web Track data set contained roughly fifty topics, with a total of more than 1.500 URLs over all topics. As this was a manageable number, we were able to use all topics and documents in this data set. And, since it only contained URLs and not the actual documents, we had to crawl all web pages before running our experiments, just like we did with the ODP data set.

4.2 Measurement

In order to measure how well our relevance measurement and crawl ordering performed, we mocked the corresponding functionalities of our crawler on a sample test set of URLs and corresponding topics and contexts. Half of the topics in our data set were then marked as relevant to the user's interests, meaning we had a sample of URLs corresponding to pages the user is supposedly interested in, and another sample of URLs corresponding to irrelevant pages. Each topic was also split into a training set and a data set, containing three quarters and a quarter of the URLs, respectively. The training documents for all relevant topics were indexed in an Elasticsearch index, just as WASP itself would do.

[2]`https://trec.nist.gov/data/web2014.html`

With this index of 'visited' pages, we could compute the relevance metric (i.e. the combined cosine similarities of term vectors) for each URL in the testing sets of each of the topics in our data set. Then, we quantified the effectiveness of the relevance metric, by sorting the test documents in descending order of relevance and the computing the Normalized Discounted Cumulative Gain (nDCG) [3]. The nDCG is a value that measures the ranking quality of this sorted list of documents. For each of the documents d_i, we assign $rel_i = 1$ if the document was marked 'relevant', and $rel_i = 0$ otherwise. Using these relevance values, we can define the standard Discounted Cumulative Gain (DCG) for the first p documents in our sequence as:

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

As can be seen from the given formula and the binary relevance values we assigned, the DCG counts the relevant documents, divided by their position in the sequence of all documents. Therefore, the DCG will be higher if relevant documents are assigned a higher relevance score and irrelevant documents a lower one. Likewise, if relevant documents receive a low score and irrelevant documents are scored higher, the DCG will turn out lower. Consequently, we can use the DCG to measure how well our relevance score represents the actual relevance of a document.

However, if we want to compare DCGs between different experiments or relevance metrics, we will have to normalize the result, as the amount of documents in our training set now influences the DCG. This is done by dividing the DCG by the maximum possible DCG, also known as the Ideal Discounted Cumulative Gain (IDCG). The IDCG is the DCG for a set of optimally ordered documents. In other words, we can take a sequence of our 'relevant' documents first, followed by all irrelevant documents, and then taking the DCG of this sequence. If we define N_r as the amount of relevant documents, we can define the IDCG and the nDCG formally as:

$$\begin{aligned} IDCG_p &= \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} \\ &= \sum_{i=1}^{\min(N_r,p)} \frac{1}{\log_2(i+1)} \\ nDCG_p &= \frac{DCG_p}{IDCG_p} \end{aligned}$$

To gain even more insight into the effectiveness of our relevance measure, we decided to compute the F-measure [10] for each experiment, as well. The F-measure is defined as such:

$$\begin{aligned}
precision &= \frac{|D_{relevant} \cap D_{retrieved}|}{|D_{retrieved}|} \\
recall &= \frac{|D_{relevant} \cap D_{retrieved}|}{|D_{relevant}|} \\
F_\beta &= \frac{(1+\beta^2) \cdot precision \cdot recall}{(\beta^2 \cdot precision) + recall}
\end{aligned}$$

In these formulas, precision measures how many of the retrieved documents are actually relevant, and recall measures the fraction of relevant documents that are actually retrieved. The F-measure combines these values, where the value β decides which of the two is more important. With $\beta < 1$, precision is weighed more, while recall is deemed more important with $\beta > 1$. A special case is when $\beta = 1$, where the value is the harmonic mean of the two values.

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

In the scope of WASP, we mainly want to eliminate false positives, whereas the occasional false negative is not that important. Therefore, we used values of β less than 1, to stress the importance of precision in our setup.

In the end, the F-value is computed the same as the nDCG for a set of documents; by sorting the documents given a certain score, and computing the F-value for different sample sizes for this ordering.

Finally, using the nDCG and F-value for each experiment, we could compare the effectiveness of our relevance detection measurement across different data sets and different amounts of documents, and we could compare differences between several parameter choices in order to find the optimal setup for our final system.

Chapter 5

Results

5.1 Relevance estimation

5.1.1 Comparison between data sets

With each of the data sets described in section 4.1, we performed the following experiment:

- Take a random set containing half of the topics, for each of which we mark all documents as relevant.
- Split each topic into a training set and a testing set, at a ratio of 3 to 1.
- Index each document in the training set of a relevant topic.
- For each document in a testing set (both relevant and irrelevant), compute the different relevance measures (section 3.2).
- Sort all test documents in descending order of relevance and compute the nDCG and F-measure (section 4.2).

In order to find the ideal setup, we computed the relevance measure in many different ways and tried to determine the best way to compute it. We did this by varying in the amount of More Like This results we asked Elasticsearch to give us (1, 2, 5 or 10) and the way we combined the results of the cosine similarity between the target documents and each of the documents returned by Elasticsearch (should all cosine similarities be above a certain threshold,

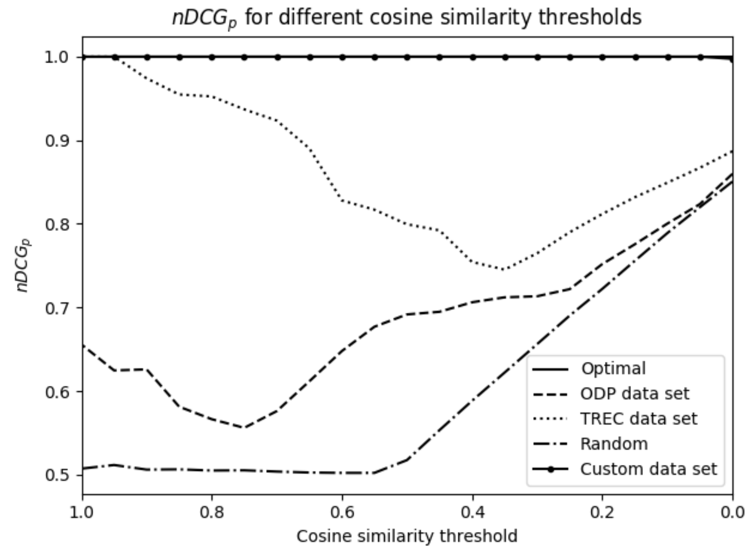


Figure 5.1: Average $nDCG$ for several cosine similarity thresholds. For each threshold, we computed the $nDCG_p$ for each combination method, with p being the number of entries that score above the given threshold. For each point, we used the combination method resulting in the highest average $nDCG$.
Ideally, we would have a line at $y = 1$ as indicated by the optimal line, since that would mean our relevance measure ranks the documents perfectly. The random line is based on a sample of randomly ordered documents.

or only the average value). In the end, we took the combination that resulted in the highest nDCG value. For extra redundancy, this experiment was repeated multiple times for each data set. The results can be found in figure 5.1, where the $nDCG_p$ is plotted against different cosine similarity thresholds.

As can be seen from this figure, the nDCG line of our custom-built data set is almost equal to the optimal line at $y = 1$. Clearly, even though this set contains too few documents to prove anything statistically, our relevance measure performs very well for this set of hand-picked topics and documents.

For the ODP data set, our measure performed slightly better than a random ordering, but not as much as we had hoped. Certainly not well enough to give any meaningful relevance estimate, proving its applicability in practice. However, with the knowledge that our custom data set performed as well as it did, we concluded that the ODP data set is not quite comparable to the data we would expect from regular user behavior after all. As a result, we did not use it for any further experiments, as it proved to be an unsuitable data set. We assume this has to do with the fact that the ODP data set mainly refers to landing pages of each website, which are not necessarily content heavy. For instance, a news article on a soccer match will be more content intensive than the landing page of a random English soccer club.

In the case of the TREC data set, our similarity measure performs a lot better, especially if we consider the higher cosine similarity thresholds. Consequently, we know now that we have a setup that performs significantly better at determining the most relevant documents in a sample set than a random ordering. Moreover, we know the TREC data set is somewhat representative of the usage we are trying to mock, meaning we can safely use it in our experiments. Therefore, all experiments from here on out were done using the TREC data set.

5.1.2 Finding the optimal parameters

In figure 5.1, we plotted the $nDCG_p$ against the cosine similarity threshold, to see how well our system could perform. However, in each point, we used the maximum value available from all combination methods (amount of Elasticsearch results to use, and how to combine them), meaning we cannot say anything useful yet on the best combination method or which threshold to use in the final system. The plot presented there can only be used to compare different data sets against the optimal ordering, random ordering and each other.

Since we also want to find out the best way to combine our Elasticsearch results, we need to compute each of the values separately and compare them against each other. Therefore, we present the $nDCG_p$ for each combination method and a couple of thresholds in table 5.1. From these results, we can at the very least conclude that all of the combination methods outperform a random ordering. On top of this, we can easily see that combinations that use more of the Elasticsearch results perform better at separating the relevant documents from the irrelevant documents, as they have a higher $nDCG_p$ score. We can conclude that using the minimum cosine similarity value of ten results performs the best, meaning this is the one we should use in our final system. Note that requiring the minimum value to be above a certain threshold is equivalent to requiring that all values are above that threshold. So, in other words, we will use ten Elasticsearch results and require that the document we are currently processing is similar enough (i.e. above a certain cosine similarity threshold) to each of these ten documents.

Finally, now we know how to compute the final cosine similarity, we still have to know when to accept it as 'relevant'. In other words, we have to

Threshold	0.6	0.7	0.8	0.9
Random	0.50	0.50	0.50	0.50
Min. of 1 result	0.65	0.66	0.69	0.70
Avg. of 1 result	0.65	0.66	0.69	0.70
Min. of 2 results	0.67	0.71	0.71	0.71
Avg. of 2 results	0.66	0.70	0.72	0.71
Min. of 5 results	0.79	0.82	0.81	0.86
Avg. of 5 results	0.76	0.79	0.81	0.83
Min. of 10 results	0.83	0.92	0.95	0.97
Avg. of 10 results	0.83	0.85	0.83	0.88

Table 5.1: nDCGs for each combination of Elasticsearch More Like This results, computed for several cosine similarity thresholds.
Each row lists the $nDCG_p$ values if we sort our result list according to the combination method mentioned in the first column. This combination method varies in the amount of Elasticsearch MLT results we use (1, 2, 5 or 10) and the way we combine their cosine similarities (whether we take the minimum value or the average of all the cosine similarities). For p, we took the amount of results that score above the given threshold.

find a certain threshold above which we can label the document relevant. To find this threshold, we plotted the $nDCG_p$, $F_{0.1}$ and $F_{0.5}$ values in figure 5.2. As mentioned in section 4.2, an F_β measure will value precision more and more as β goes to 0. That is also the reason we plotted both $F_{0.1}$ and $F_{0.5}$. In both cases, precision is more important than recall, as it is more important in the situation of WASP to limit false positives than false negatives. That is where the $F_{0.1}$ measure is useful; it still values recall somewhat, but it mostly cares about precision. However, in some cases, a user might be more tolerant of false positives, and they might be interested in limiting the false negatives a little more. In this case, they might use the $F_{0.5}$ measure instead, as it provides more insight in the recall and thus in the amount of false negatives.

By looking at both the nDCG curve and one of both F_β curves, we can determine what threshold is useful for a specific use case. For instance, both the nDCG and the $F_{0.1}$ value are very high around the threshold of 0.9, meaning that would be a good threshold if the aim is to limit false positives to an absolute minimum. The lower you choose the threshold, the more relevant results we will actually accept, as indicated by the rising $F_{0.5}$ curve. However, at the same time, the decreasing nDCG and $F_{0.1}$ values indicate that a lower threshold will also allow for more false positives to

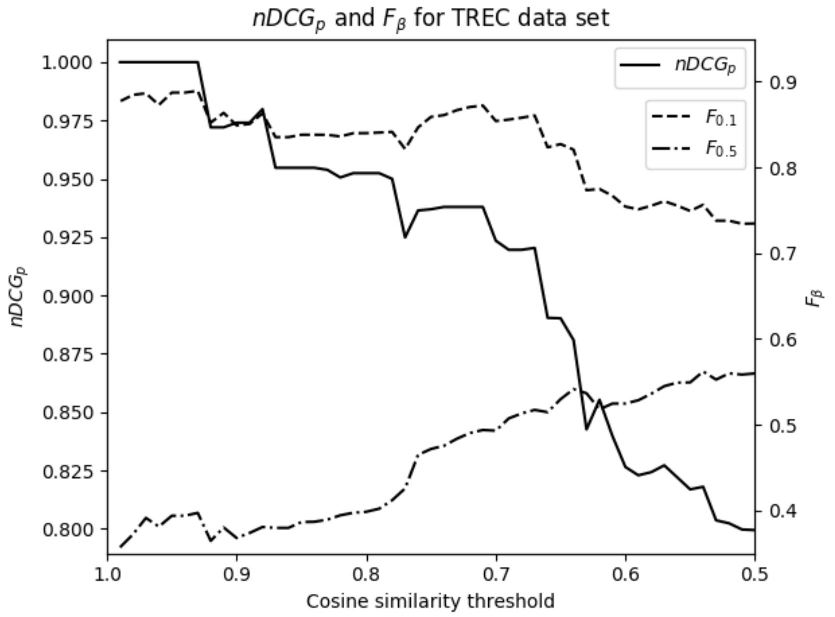


Figure 5.2: nDCG, $F_{0.1}$ and $F_{0.5}$ for different cosine similarity thresholds, computed from the TREC data set.

pass through the relevance filter. Since the initial purpose of this research is to limit the false positives as much as reasonably possible, we set the threshold to 0.9. This value is user customizable, though, meaning it can be tailored to different users with varying use cases.

5.2 Crawl ordering

As mentioned in section 3.1, we wanted to sort the URLs to crawl on their predicted relevance, as it might be infeasible to crawl all of the URLs in a reasonable amount of time. We decided to adopt the approach from [9], where we use the context (i.e. surrounding words) of an anchor tag to determine how likely a document is to be relevant. However, as they did in their research, we still had to determine how many surrounding words we had to use in order to reach optimal efficiency.

In order to test this, we first had to find a set of documents, labeled with topics, and corresponding context for anchor tags referencing these documents. For the set of documents and corresponding topics, we used the TREC data set again, as it proved useful and suitable in the other experiments. Then, we used the Moz Link Explorer[1] to find backlinks to the documents in our test set. In other words, using this Link Explorer we were able to find the URLs of web pages that contained an anchor tag mentioning a document in our data set. Using these backlinks, we were able to crawl these pages and extract the context for the corresponding anchor tag. Unfortunately, the crawl from which the backlink statistics came was slightly out of date, meaning some pages no longer contained the hyperlink we were looking for and other pages were taken offline entirely. However, in the end we had access to a set of over 500 context samples for different documents in the TREC data set.

Using this newly obtained data, we performed roughly the same experiment as we did with the relevance measurement:

- Take a random set containing half of the topics, for each of which we mark all documents as relevant.
- Split each topic into a training set and a testing set.
  Since we could only use documents for which we found a backlink and context in the test set, we decided to use all documents without context as training set, and all documents with context as the test set. For the purposes of this experiment, we assumed this would not lead to any bias as there is no real underlying reason we could not find the backlinks and contexts. The ratio between these sets was approximately 3 to 1, as it was in our previous experiments.
- Index each document in the training set of a relevant topic.
- For each document in a testing set (both relevant and irrelevant), query the context against the Elasticsearch index and determine the score.
- Sort all test documents in descending order of determined score and compute the nDCG.

Since we wanted to figure out which context length was most suitable for our crawl ordering, we varied with different context lengths. We used the values that were used in [9] (5, 10, 20 or 40 words), and tried some other, larger values as well (60 or 80 words, or the full document). We also experimented with two different ways to combine the results Elasticsearch gave us. The first was to use the highest score found, and the second was to take the sum of all the returned scores. Again, the experiments were repeated many times to ensure they were not influenced by the set of randomly selected relevant documents. The results of these experiments can be found in table 5.2.

[1] `https://moz.com/link-explorer`

Context length	Combination method	1	2	3	4	5	6	7	8	9	10
5	Max	0.46	0.48	0.56	0.61	0.62	0.63	0.65	0.65	0.65	0.67
5	Sum	0.51	0.50	0.56	0.56	0.57	0.61	0.62	0.64	0.65	0.67
10	Max	0.59	0.55	0.55	0.57	0.60	0.60	0.62	0.63	0.63	0.64
10	Sum	0.56	0.57	0.55	0.58	0.60	0.61	0.63	0.64	0.65	0.65
20	Max	0.56	0.53	0.55	0.56	0.58	0.60	0.62	0.62	0.63	0.65
20	Sum	0.59	0.55	0.58	0.57	0.60	0.62	0.63	0.64	0.64	0.65
40	Max	0.56	0.53	0.56	0.58	0.60	0.61	0.63	0.63	0.65	0.66
40	Sum	0.56	0.55	0.57	0.59	0.61	0.64	0.66	0.67	0.68	0.68
60	Max	0.61	0.61	0.61	0.66	0.69	0.71	0.73	0.73	0.73	0.73
60	Sum	0.63	0.63	0.65	0.68	0.70	0.71	0.72	0.73	0.73	0.74
80	Max	0.61	0.62	0.62	0.66	0.68	0.70	0.71	0.72	0.72	0.73
80	Sum	0.63	0.62	0.65	0.68	0.70	0.71	0.72	0.73	0.74	0.74
All	Max	0.61	0.65	0.65	0.66	0.68	0.69	0.70	0.70	0.70	0.70
All	Sum	0.61	0.65	0.66	0.67	0.69	0.70	0.71	0.71	0.71	0.72

Table 5.2: $nDCG_p$ for different values of p, calculated for each of the performed experiments. Each row depicts the average values of $nDCG_p$ for p from 1 through 10, computed when using the mentioned amount of context words and combination method. The combination method indicates how we combined the scores returned by Elasticsearch, i.e. whether we take the maximum score or the sum of all scores.

From these values, we can see that a larger context window results in a better crawl ordering, but only to a certain extent. Using the full document a link is found in as its context performs slightly worse than using a context of 60 or 80 words surrounding the link. Taking into account that we also want to limit the storage space used up by the links and contexts, we concluded WASP should store the 60 words surrounding a URL when storing it for further crawling. The scores returned by Elasticsearch are then summed up to obtain the best crawl ordering.

However, in these experiments we were limited by the data we had access to. As we mentioned, we crawled backlinks of documents known to be relevant or irrelevant, and extracted the context of the corresponding links in order for us to perform this experiment. In the case of larger context windows, though, it is very likely that the extracted context surrounds other hyperlinks, too. Unfortunately we do not know whether these other documents are relevant or irrelevant, which degrades our experiment somewhat.

For instance, if the user has visited a web page containing a news article, that page might contain a list of other popular articles in a sidebar. If we take a large number of context words, all of these closely grouped links will have roughly the same context. As a result, each of these links will receive approximately the same score, even though some might clearly be relevant and others might not seem even remotely similar.

In such a situation, it is probably better to use a smaller context size, such that the context is limited to the corresponding anchor tag only, and not to other links in the neighborhood. Unfortunately, in the data set we have, we only know for one link in each document whether it is relevant: the document of which we retrieved the backlink. Thus, we miss out on the subtleties mentioned above, meaning the larger context windows perform better according to our experiments.

In order for this experiment to be improved, a more comprehensive data set is necessary. Besides the current set of queries and corresponding web pages, we would also need to have one or more documents referencing each of the web pages in our data set. And, for each of the web pages referenced by these documents, we would require a relevance assessment or some other form of classification. Using this larger and more comprehensive data set, we could repeat the experiment as described above, where the smaller context windows will probably outperform the larger ones. Creating such a data set and corresponding relevance assessments takes a lot of time and effort, though, as relevance assessments often have to be made manually. Therefore, we decided not to do so in this thesis, and leave the fine tuning of the context lengths up for future research.

Chapter 6

Related Work

6.1 Topical crawling

There is not yet a solution for focused crawling in which the topic of the crawl is dynamic. Multiple papers have been dedicated to focused crawling, but the common denominator between each of these papers is that they first establish a topic to crawl, feed it to the crawler and then let it run.

For instance, [2] describes a way of focused crawling in which the topic is already known beforehand, and relevant keywords are extracted from pages known to be relevant. These keywords can then be used as a similarity query on any new pages crawled, indicating whether they are relevant to the given topic. Or, they can be compared against the context of a given link, to determine whether the crawler should follow that link.

In our case, though, the topic depends entirely on the contents of the WASP index, which solely consists of pages visited by the end user, meaning their topic can be anything. Therefore, we cannot extract relevant keywords to query new pages against beforehand. The closest we can come to this is computing the relevant keywords on the fly, whenever we want to compare a document to our existing set of relevant documents, but this is actually exactly what our proposed relevance measurement does.

Another common approach to topical crawling is to train a classifier on the topic of the crawl, and use this classifier to determine whether a crawl candidate is relevant to the given subject. [9] poses an example of such a setup. However, like the method mentioned above, this approach suffers from the issue that subjects in our WASP index are not fixed beforehand. Since the topic of the crawl can be any possible combination of subjects, training a classifier based on these subjects beforehand is quite impossible.

Moreover, classifiers need a lot of data to train on before their results are really trustworthy. Since our index only consists of pages visited by the user, the amount of data completely depends on the amount of pages the user visits. Without going into the specific numbers, it is quite unlikely that the average user visits enough pages to fully train a classifier in a reasonable amount of time.

Finally, another reason that classifiers would be hard to use with our current setup is that we only have examples of documents that are relevant to our subject. Training a classifier would require a set of irrelevant documents, too, as the classifier has to somehow find the features that distinguish between 'relevant' and 'irrelevant'.

6.2 Similarity measurement

As mentioned in section 2.3, there are many different ways to measure the similarity between two documents. And, even though semantical analysis performs better in most tasks than lexical analysis, we have still chosen to adopt the simple bag-of-words model in this extension to WASP. Again, this is mostly due to the basic reason that we are unable to use classifiers (or any other trained network for that matter) in our current use case. A regular user does simply not have enough data to train a network on. Another option could be to use a pre-trained classifier, but that would mean we know the set of subjects a user is interested in beforehand, which is also not the case.

But even if we look past the issues that arise if we want to train a classifier, even with a trained network we might not be able to use the top-of-the-line similarity measures in this context. For instance, the Doc2vec measure [6] seems to do what we want, as it represents the meaning of a document in a vector. However, because of the way the vector is computed, the location of each paragraphs is embedded in this vector, alongside the paragraph's computed 'meaning'. However, in our case, the structure of the document is irrelevant; we are only concerned with the actual topic of the document. Thus, we don't want the structure of two documents to impact the similarity between them. And that is exactly why the bag-of-words model might be more suitable in this case, even though it is less sophisticated.

Chapter 7

Conclusions

In this paper, we introduced an extension to WASP that enables the user to search through pages they did not necessarily visit themselves. We do this by storing all URLs found in visited documents, along with the context surrounding the anchor tags they are found in. With a set of experiments we found that we should use a context of 60 words for a simple crawl ordering, and that this crawl ordering makes sure the documents more likely to be relevant are crawled first. However, we also acknowledged that these experiments miss some subtleties due to a lack of relevance assessments for certain groups of documents. This causes the results of our context experiments to be inconclusive, as it might turn out later that smaller context windows outperform the 60 word window.

When reviewing a crawled document for its relevance, we found the simple bag-of-words model with a cosine similarity measure to be the most suitable measurement. With extensive testing, we have shown that results are quite accurate, especially if we use a higher cosine similarity threshold. We also noticed that a different use case of the system might call for a different cosine similarity threshold. For instance, when false positives are not much of an issue, we could increase the recall, or amount of relevant documents we find, by lowering the threshold.

The final extension to WASP, in which we implemented the findings from our experiments, can be found on GitHub[1].

[1]`https://github.com/gijshendriksen/wasp`

Chapter 8

Future work

8.1 User testing

In this research, we mentioned several well-known metrics and used them to validate the accuracy of the extension we built. From these metrics, it followed that the theory behind our extension is valid, and that the extension should work as expected. However, for these experiments, we used a couple of existing data sets and one smaller, custom data set. And, although these data sets were selected to mimic user interests in the form of subjects or categories, it is infeasible that these data sets and the random samples we extracted from them accurately represent actual user behavior.

In order to actually verify this new system works accurately for actual, real life users, a different type of experiment should be executed. For instance, a larger scale experiment with multiple users using WASP as a part of their daily browsing behavior, who can verify whether the suggestions for related documents are actually relevant to their interests. This way, we test the performance of WASP on actual user data, and the relevance of documents is determined by the user whose interests we are trying to predict.

8.2 User interaction

As it stands right now, suggestions made by WASP cannot be removed by the user, nor can they be accepted or assessed in any other way. Currently, it is built to not include these suggestions when determining the relevance of other crawled documents, meaning the system will not recommend documents similar to other suggestions. However, by providing the user with the means to rate a recommendation's relevance, we can get more insight into their interests and get a larger sample of positive documents to compare new documents against. The most interesting development, though, could be that we gain access to a set of negative examples (i.e. irrelevant documents), which could allow us to predict the relevance of new documents more accurately.

8.3 Performance

Mostly concerned with the technical possibilities, this research does not cover the performance of the new system all too much. Since WASP is designed to run locally on a normal computer with regular hardware, one of its key requirements is that running WASP is not too taxing on the user's computer. Concretely, this means it should not take up too much of the computer's processing time, memory or bandwidth. Therefore, a possibility for further research is finding out how this extension to WASP influences these parts of the user's computer, for instance by measuring memory or network usage over time and finding the optimal interval between crawling operations.

8.4 Slow search

The current crawler and relevance metric, as they are introduced in this paper, are both quite basic. This was mostly done deliberately to limit the impact WASP has on the performance of the computer it runs on. However, since the crawler runs in the background, there is no need for it to hurry when searching for relevant documents. Therefore, in the spirit of 'slow search' [11], we could also decide to spend more resources on determining the relevance of a certain document using more difficult measures than the simple bag-of-words model with a cosine similarity. This would mean we process fewer possible relevant documents within a given time span, in return for more accurate relevance predictions. And by assigning a relatively low priority to the crawler process, we could still keep the impact on the normal performance of the computer in check.

Bibliography

[1] Dennis Fetterly, Nick Craswell, and Vishwa Vinay. The impact of crawl policy on web search effectiveness. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 580–587, New York, NY, USA, 2009. ACM.

[2] Emil Gatial, Zoltan Balogh, Michal Laclavik, Marek Ciglan, and Ladislav Hluchy. Focused web crawling mechanism based on page relevance. *Proceedings of ITAT*, pages 41–46, 2005.

[3] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002.

[4] Johannes Kiesel, Arjen P. de Vries, Matthias Hagen, Benno Stein, and Martin Potthast. Wasp: Web archiving and search personalized. In *DESIRES*, 2018.

[5] Julian Kupiec. Robust part-of-speech tagging using a hidden markov model. *Computer Speech & Language*, 6(3):225–242, 1992.

[6] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1188–II–1196. JMLR.org, 2014.

[7] Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

[8] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[9] Gautam Pant and Padmini Srinivasan. Link contexts in classifier-guided topical crawlers. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):107–122, Jan 2006.

[10] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[11] Jaime Teevan, Kevyn Collins-Thompson, Ryen W. White, Susan Dumais, and Yubin Kim. Slow search: Information retrieval without time constraints. ACM - Association for Computing Machinery, October 2013.