CiteWise

The Citation Search Engine

Master Thesis

Aliya Ibragimova

University of Fribourg

Faculty of Natural Sciences
University of Bern

June 2015

Prof. Dr. Oscar Nierstrasz

Haidar Osman, Boris Spasojević

Software Composition Group
University of Bern, Switzerland

Abstract

Nowadays the number of documents in the World Wide Web grows at extremely fast rate[1]. Tools that can facilitate information retrieval (IR) present a particular interest in the modern world. We believe that considering meta information helps us to build enhanced search systems that can facilitate IR. Particularly, we target an IR task for scientific articles. We consider citations in scientific articles to be important text blocks summarizing or judging previous scientific findings, assisting in creating new scientific work.

We propose *CiteWise*, a software system that automatically extracts, indexes and aggregates citations from collections of scientific articles in a PDF format.

We evaluated the capabilities of *CiteWise* by conducting user evaluation experiments that compare it with alternative approaches. In the first set of experiments, we measured the efficiency of our system, i.e. how fast users can find relevant results in comparison with *Google Scholar*. We found that CiteWise performs equally well as Google Scholar. Secondly, we developed a citation aggregation feature to create automatic summaries of scientific articles and asked domain experts to evaluate summaries created by CiteWise and TextRank algorithms. We found that CiteWise outperforms TextRank algorithm in generating article summaries.

[1] `http://googleblog.blogspot.ch/2008/07/we-knew-web-was-big.html`

Contents

1
Introduction

1.1 Thesis statement

The increasing amount of research literature produced by the scientific community poses a number of challenges to the task of finding relevant prior art to newly written papers and filed patents. Thus, searching for prior related work becomes an extremely hard task for an individual or a small group of researchers, which has been studied for cases such as patent search [11]. Other works [26] even claim that this will result in a “fundamental phase transition in how scientific results are obtained, represented, used, communicated and attributed”, and propose their own system to organize and navigate scientific knowledge [1].

This work focuses on the specific problem of finding relevant citations for claim statements. During the process of writing of a paper, one of the main difficulties is to validate proposed claims with the right citations. The claims are required in many situations to construct a valid argument, but only if supported by appropriate citations. Current solutions to the problem, such as full-text search engines, are typically based on keyword search, and thus do not always work well for the case of finding relevant citations. This happens due to the fact that they do not take document structure into account, i.e, that some sentences in a document are more likely to contain claims than others. Therefore, such systems return a large amount of irrelevant results. A more suitable approach is to look at what other people used in their papers as references for their claims. In other words, if we have a previous paper using certain claims, we can see what citations the authors used to support those claims.

We address the described problem by introducing CiteWise, a novel search engine for scientific literature based on citations. In contrast to ordinary information retrieval (IR) systems that index entire content of articles, we focus on indexing citations extracted from articles. We study the structure of citations and design an algorithm that aggregates citations referring to the same source. We use the aggregation mechanism of CiteWise to generate automatic summaries of papers. CiteWise provides a web search interface that supports the following use cases: 1) finding relevant citations based on statements and 2) searching for bibliographic entries using meta-information, such as author names and venues. Additionally users can look up all citations of a given article in other articles.

1.2 Contributions

The following are the main contributions of this work:

- A novel IR system for scientific articles based on citations.
- A search interface to discover relevant scientific results based on a statement query.
- A new method of summary generation by means of citation aggregation.
- An empirical evaluation of the system by means of user study experiments.

1.3 Outline

The rest of the paper structured as follows:

chapter 1 gives a high overview of the architecture of a typical web search engine. It describes the main steps to construct an inverted index.

chapter 3 surveys the research related to citations in scientific publications. It overviews two popular academic search engines: Google Scholar and CiteSeer.

chapter 4 describes the design of CiteWise. It first shows overall architecture of the proposed system and then shows details of implementation of each component.

chapter 5 describes user evaluation experiments and analyzes the results.

chapter 6 concludes the work.

chapter 7 describes potential future work.

appendix provides a user guide for the CiteWise deployment.

1.4 Glossary of Terms

Citation A *citation* is a piece of text (usually a claim, within the body of an article), including a (bibliographic) *link* to a *bibliographic reference* (in a references section of the article), that identifies a *source text* (another work) justifying that claim.

Bibliographic link or link *Bibliographic link* is a link to a *bibliographic reference*. It consists of a unique identifier of the *bibliographic reference*, normally within square brackets (e.g. "[23]", "[Giles97]").

Bibliographic reference is a bibliographic entry in a references section of the article identifying another work.

Document a broader term, having multiple meanings. In this work we can use a term *document* to refer to a single file, i.e a PDF article. A *document* term can be used to refer to a basic storage unit, i.e basic storage unit of an Indexes storage or a MongoDB database.

2

Technical background

2.1 Typical Web Search Engine

Figure 2.1 illustrates a high level architecture of a standard web engine. It consists of three main components:

- Crawler
- Indexer
- Index Storage
- Search interface

A Web Crawler is a program that browses the World Wide Web reading the content of web pages in order to provide up-to-date data to the Indexer. The Indexer decides how a page content should be stored in an index storage. Indices help to quickly query documents from the index storage. Users can search and view query results through the Search Interface. When a user makes a query the search engine analyzes its index and returns best matched web pages according to specific criteria.

Web crawlers that fetch web pages with the content in the same domain are called focused or topical crawlers [7]. An example of a focused crawler is an academic-focused crawler that crawls

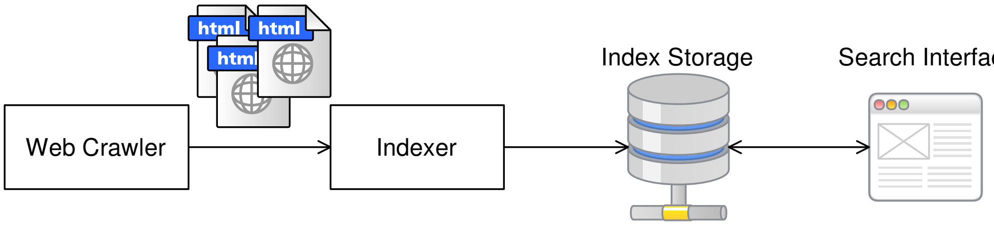


Figure 2.1: A high-level architecture of a typical web search engine

scientific articles. Such crawlers become components of focused search engines. Examples of popular academic search engines are Google Scholar[1] and CiteSeer[2]. Chapter 3 gives an overview of these search engines.

2.2 Inverted Index

Search engines like CiteSeer or Google Scholar deal with a large collection of documents. The way to avoid scanning the text of all documents for each query is to *index* them in advance. Thereby we are coming to the concept of *inverted index*, which is a major concept in IR. The term *inverted index* comes from the data structure storing a mapping from content, such as words or numbers, to the parts of a document where it occurs. Figure 2.2 shows an example of an inverted index. We have a dictionary of terms appearing in the documents. Each term maps to a list that records which documents the term occurs in. Each item in the list, conventionally named as *posting*, records that a term appears in a document, often recording the position of the term in the document as well. The dictionary on Figure 2.2 has been sorted alphabetically and each posting list is sorted by document ID. A document ID is a unique number that can be assigned to a document when it is first encountered. The construction of the inverted index has the following steps:

1. Obtaining a document collection (usually performed by the crawler);
2. Breaking each document into tokens, turning a document into a list of tokens;
3. Linguistic preprocessing of a list of tokens into normalized list of tokens;
4. Index documents by creating an inverted index, consisting of a dictionary with terms and postings.

[1] `https://scholar.google.ch/`
[2] `http://citeseerx.ist.psu.edu/index`

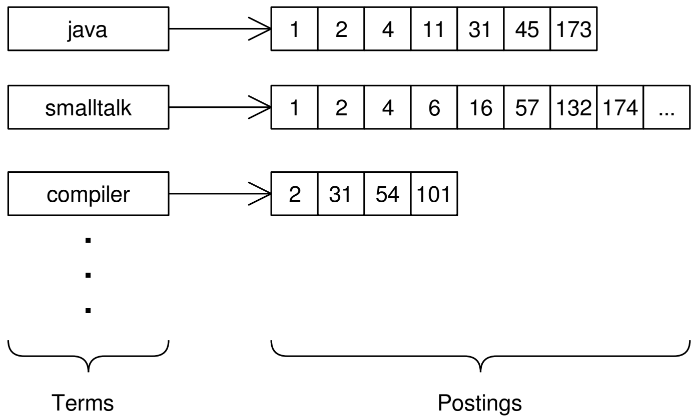


Figure 2.2: Example of an inverted index. Each term in a dictionary maps to a posting list consisting of document IDs, where this term occurs. Dictionary terms are sorted alphabetically and posting lists are sorted by document IDs

Once all documents are collected (for example, by the crawler), one can begin to build an inverted index.

We begin the index construction by breaking up each document into *tokens*. Tokens can be thought of as the semantical units for processing. For example, it might be a word or a number. During tokenization, some characters, such as punctuation marks, can be thrown away. An example of the tokenization process is shown below:

Input: Sometimes, I forget things.
Output: $\boxed{\text{Sometimes}}$ $\boxed{\text{I}}$ $\boxed{\text{forget}}$ $\boxed{\text{things}}$

The next step in the index construction is normalization. Consider an example of querying the word *co-operation*. A user might also be interested in getting documents containing *cooperation*. *Token normalization* is a process of turning a token into a canonical form so matches can occur despite lexical differences in the character sequences. One way of token normalization is keeping relations between unnormalized tokens, which can be extended to manual constructed synonym lists, such as *car* and *automobile*. The most standard way of token normalization however is creating *equivalence classes*. If tokens become identical after applying a set of rules then they are in the equivalence classes. Common normalization rules are:

Stemming and Lemmatization Words can be used in different grammatical forms. For instance, *organize*, *organizes*, *organizing*. However in many cases it sounds reasonable for one of these words to return documents that contain other forms of the word. The goal of stemming and lemmatization is to reduce the form of the word to a common base form.

Here is an example:

am, are, is → be

car, cars, car's, cars' → car

The result of applying the rule to the sentence:

three frogs are flying → three frog be fly

Stemming and lemmatization are closely related concepts however there is a difference. *Lemmatization* usually refers to finding a *lemma*, common base of a word, with the help of a vocabulary and morphological analysis of a word. *Lemmatization* may require understanding the context of a word and language grammar. *Stemming* however refers to reducing inflected (or sometimes derived) words to their word *stem*. The word's *stem* is not necessarily identical to its *lemma*.

Here is an example:

better → good, can only be matched by lemmatization since it requires dictionary look-up

picked → pick, can be matched by both lemmatization and stemming

meeting → meeting (noun) or to meet (verb), can be matched only by lemmatization since it requires the word context

In general, stemmers are easier to implement and run faster. The most common algorithm for stemming is *Porter's* algorithm [25].

Capitalization/Case-Folding A simple strategy is to reduce all letters to a lower case, so that sentences with *Automobile* will match to queries with *automobile*. However this approach would not be appropriate in some contexts like identifying company names, such as *General Motors*. Case-folding can be be done more accurately by a machine learning model using more features to identify whether a word should be lowercased.

Accents and Diacritics Diacritics in English language play an insignificant role and simply can be removed. For instance *cliché* can be substituted by *cliche*. In other languages diacritics can be part of the writing system and distinguish different sounds. However, in many cases, users can enter queries for words without diacritics.

The last step of building the inverted index is sorting. The input to indexing is a list of pairs of normalized tokens and documents IDs for each document. Consider an example of three documents with their contents:

- Document 1: Follow the rules.

- Document 2: This is our town.
- Document 3: The gates are open.

After applying tokenization and normalization steps of the listed documents the input to the indexing is shown in Table 2.1. The indexing algorithm sorts the input list so that the terms are

Term	**DocumentID**
follow	1
the	1
rule	1
this	2
be	2
our	2
town	2
the	3
gate	3
be	3
open	3

Table 2.1: Input to the indexing algorithm is a list of pairs of a term and document ID, where this term occurs.

in alphabetical order as in Table 2.2. Then it merges the same terms from the same document

Term	**DocumentID**
be	2
be	3
follow	1
gate	3
open	3
our	2
rule	1
the	1
the	3
this	2
town	2

Table 2.2: Indexing algorithm sorts all terms in a alphabetical order. The result is a list of sorted terms with document IDs

by folding two identical adjacent items in the list. And finally instances of the same term are grouped and the result is split into a dictionary with postings, as shown in Table 2.3.

Term	Postings
be	2 3
follow	1
gate	3
open	3
our	2
rule	1
the	1 3
this	2
town	2

Table 2.3: Indexing algorithm groups the same terms with creating postings. The result is a dictionary with terms as keywords and values as postings.

The described above index construction algorithm is an application of the MapReduce framework, a general architectural pattern for distributed computing[3]. Normally web search engines work with very large collections of documents and, therefore, use distributed indexing algorithms for the index construction.

2.3 Dynamic Indexing

So far we assumed that the document collection is static. However there are many cases when the collection can be updated, for example, by adding new documents, deleting or updating existing documents. A simple way to deal with dynamic collections is to reconstruct the inverted index from scratch. This might be acceptable if the changes made in the collection are small over time and the delay in making new documents searchable is not critical. However if one of the aforementioned conditions is violated, one might be interested in another more dynamic solution like keeping an auxiliary index. Thus we have a large main index and we keep an auxiliary index for changes. The auxiliary index is kept in memory. Every time a user makes a query the search runs over both indexes and results are merged. When the auxiliary index becomes too large it can be merged with the main index.

2.4 Retrieving Search Results

When a user makes a query he prefers to get a result document containing all query terms, so that the terms appear close to each other in the document. Consider an example of querying a

[3] https://en.wikipedia.org/wiki/MapReduce

phrase containing 4 terms. The part of the document that contains all terms is named a *window*. The size of the window is measured in number of words. For instance the smallest window for 4-term query will be 4. Intuitively, smaller windows represent better results for users. Such a window can become one of the parameters ranking a document in the search result. If there is no document containing all 4 terms, a 3-term phrase can be queried. Search systems hide the complexity querying from the user by introducing *free text query parsers* [18].

3
Related Work

3.1 Citations In Scientific Publications

Citations are the subject of many interesting scientific studies. Bradshaw *et al.* [5] showed that citations provide many different perspectives on the same article. They believe that citations provide means to measure the relative impact of articles in a collection of scientific literature. In their work the authors improved the relevance of documents in the search engine results with a method called Reference Directed Indexing (RDI). RDI is based on a comparison of the terms authors use in reference to documents.

Bertin and Atanassova [2] [3] and Bertin *et al.* [4] automatically extract citations and annotate them using a set of semantic categories. In [4] and [2] they used linguistic approach, which used the contextual exploration method, to annotate automatically the text. In [3] they proposed a hybrid method for the extraction and characterization of citations in scientific papers using machine learning combined with rule-based approaches.

There are several studies that used citations to evaluate science by introducing a map of science. A map of science graphically reflects the structure, evolution and main contributors of a given scientific field [9] [15] [17] [30].

Kessler [14] first used the concept of bibliographic coupling for document clustering. To build a cluster of similar documents Kessler used a similarity function based on the degree of bibliographic coupling. Bibliographic coupling is the number of bibliographic references two documents have in common. The idea was developed further by Small in co-citation analysis [29]. Later co-citation analysis and bibliographic coupling was used by Larson [16] for measuring the similarity of web pages.

Another approach is to use citations to build summaries of scientific publications. There are three categories of summaries proposed based on citations: an overview of a research area (multi-document summarization) [23], an impact summary (single document summary with citations from the scientific article itself) [19] and a citation summary (multi- and single document summarization, in which citations from other papers are considered) [27]. In work by Nakov *et al.* citations have been used to support automatic paraphrasing [22].

An expert literature survey on citation analysis was made by Smith [31], she reviewed hundred of scientific articles on this topic.

3.2 Popular Academic Search Engines

3.2.1 CiteSeer^x

CiteSeer^x is built on the concept of a citation index. The concept of citation index was first introduced by Eugene Garfield [10]. According to Eugene Garfield citations are bibliographic references linking scientific documents. In his work Eugene Garfield proposed an approach where citations between documents were manually cataloged and maintained so that a researcher can search through listings of citations traversing citation links either back through supporting literature or forward through the work of later researchers [6].

Lawrence *et al.* automated this process in CiteSeer^x [1] [12], a Web-based information system that permits users to browse the bibliographic references between documents as hyperlinks. CiteSeer^x automatically parses and indexes publicly available scientific articles found on the World Wide Web.

CiteSeer^x is built on top of the the open source infrastructure SeerSuite[2] and uses Apache Solr[3] search platform for indexing documents. It can extract meta information from papers such as the title, authors, the abstract and bibliographic references. The extraction methods are based on machine learning approaches such as ParseCit [8]. CiteSeer^x currently has over 4 million documents with nearly 4 million unique authors and 80 million citations.

CiteSeer^x indexes bibliographic references while in CiteWise we intend to index not only bibliographic references but also cited text in a body of a document. If by indexing bibliographic

[1] CiteSeer, `http://citeseerx.ist.psu.edu/`
[2] SeerSuite, `http://citeseerx.sourceforge.net/`
[3] Apache Solr. `http://lucene.apache.org/solr/`

references CiteSeer^{x} mainly aims to simplify navigation between linked documents, in CiteWise we focus on simplifying retrieval of documents containing a text of interest.

3.2.2 Google Scholar

Google Scholar is a freely accessible web search engine that makes full-text and metadata indexing of scientific literature [4]. Besides the simple search, Google Scholar proposes a unique ranking algorithm that ranks documents *"the way researchers do, weighing the full text of each document, where it was published, who it was written by, as well as how often and how recently it has been cited in other scholarly literature"* [5]. The "Cited by" feature allows one to view abstracts of articles citing the given article. The "Related articles" feature shows the list of closely related articles. It is also possible to filter articles by author name or publication date. Google Scholar contains roughly 160 million documents by May 2014 [24].

Google Scholar is based on keyword search, and thus does not work well for the case of finding relevant citations. This happens due to the fact that it does not take document structure into account, i.e, that some sentences in a document are more likely to contain claims than others. Therefore, Google Scholar might return a large number of irrelevant results for statement queries.

---

[4]Google Scholar, `http://scholar.google.ch/`

[5]`https://scholar.google.com/scholar/about.html`

4
CiteWise

4.1 System Overview

The components of CiteWise are shown in Figure 4.1. CiteWise allows one to perform the three following main operations: parsing PDF files, indexing document collections and querying the resulting indexes. Correspondingly, there are three major components responsible for carrying out these operations: *Parser*, *Indexer* and *Search Web App*. The system has two more components for storing data: *Indexes Storage* and *Meta Data Storage*. We use *Indexes Storage* for storing indexes built on citations. This storage is very simple and was not designed to represent any relations in data structures. Moreover, it does not allow one to perform any sophisticated operations over the stored data. Therefore, we use *Meta Data Storage* to represent complex data structures and perform sophisticated queries, like aggregating citations referring to the same article.

The workflow of the system is shown in Figure 4.2. The first operation performed by the system is parsing. The *Parser* converts a PDF file into text. Then it extracts meta information, like citations and references, from the textual representation of the file. Next it packages extracted information into data units corresponding to formats acceptable by *Indexer* and *Meta data storage*. A data unit publishable to the *Indexer* consists of a citation that should be indexed and additional information related to this citation (citation context, a file URI, bibliographic references) that should be stored. A data unit publishable to the *Meta data storage* consists of a citation, a source

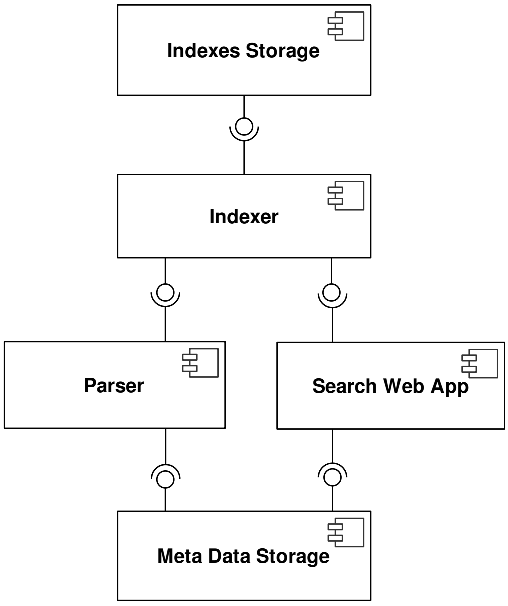


Figure 4.1: Component diagram of CiteWise.

paper identifier and bibliographic references. We use *Meta data storage* for aggregating citations referring to the same source. Once the *Parser* has processed the PDF file it can proceed to the next paper if there are any left. When all papers are processed, the user can make queries with the *Search Web App*.

The next sections of this chapter describe the implementation of each component in detail and show the reasons behind choosing a particular solution.

4.2 Parser

It is practical to divide the work of the *Parser* into two phases: *PDF processing* and *Document publishing*, as in Figure 4.3. The output of the *PDF processing* phase is the input to the *Documents publishing* phase.

4.2.1 PDF Processing

The main role of the *PDF processing* phase is to parse scientific articles into text and extract citations and bibliographic references to create documents for publishing. Parsing PDF files from different sources is a very challenging task due to the large variation in the structuring of article content. Thereby, building a universal parser is very hard in practice. In our case, we try to

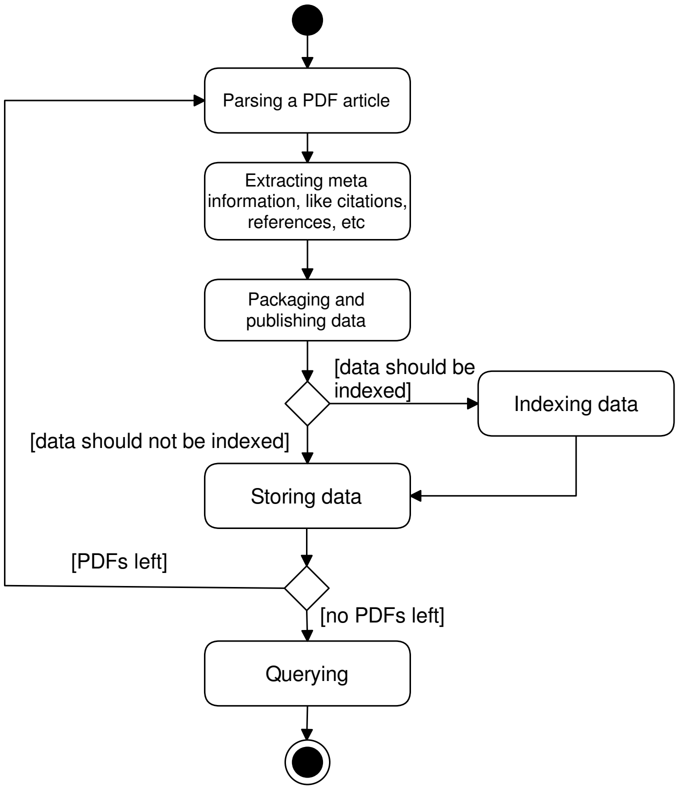


Figure 4.2: Activity diagram of CiteWise.

identify common patterns covering the structure of majority of the scientific articles or at least the articles found in our dataset.

The *PDF processing* phase starts with recursively walking though the directory tree of the collection of PDF documents. While walking through the directory, the *Parser* filters non-pdf files and parses and processes each PDF file separately. We use Apache PDFBox library[1]. The library extracts full text from PDF files, but without any hints to the initial structure of the article. To find citations and bibliographic references in text, we search them in different parts of the article. Therefore, we implemented an algorithm to break the PDF text into sections.

Generally, we are interested in identifying the body of a document where we can find citations and the references section where we can find bibliographic references. One way of finding these sections can be using keywords that might signify the beginning or the end of some sections. Based on those keywords, one can extract different sections of a document. Figure 4.4 shows a sample text of a parsed PDF document with keywords.

One can notice the following characteristics of scientific articles:

[1]Apache PDFBox, `https://pdfbox.apache.org/`

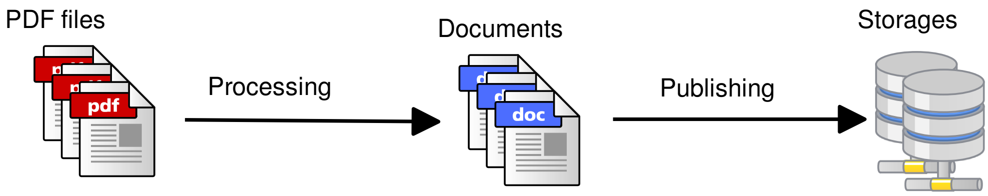


Figure 4.3: Parser workflow

- The body of a document comes before the references section.
- The appendix or author's biography sections can come after the references section.
- Each document contains the "Abstract" and the "References" words and might contain the "Appendix" word. We call these words keywords.

The keywords can be written in different formats, like using upper or lower cases. Table 4.1 illustrates variations of the keywords.

body	references	appendix
Abstract ABSTRACT	References References: REFERENCES	Appendix APPENDIX

Table 4.1: Keywords identifying different sections in a document

After breaking a document down into sections as shown in Figure 4.4, the text is presented in one-column format. There are two aspects regarding this format. First, sentences can be split by new line symbols at the end of a line. Second, words can be split by dash symbol at the end of a line. We introduce a normalization step where new lines are substituted by white spaces and dashes are removed in the end of a line to obtain continuous text.

As a result of the normalization step, we have a document divided into body and references sections. Before searching citations in the body of a document, we break the body into sentences. In general, breaking text into sentences is not an easy task. Consider a simple example with a period. A period not only indicates the end of a sentence but also can be encountered inside the sentence itself, like an item of a numbered list or a name of a scientist. Besides, not all sentences end with a period, like the title of a section or an item of a list. We use the Stanford CoreNLP library which employs natural language and machine learning techniques to extract sentences [13].

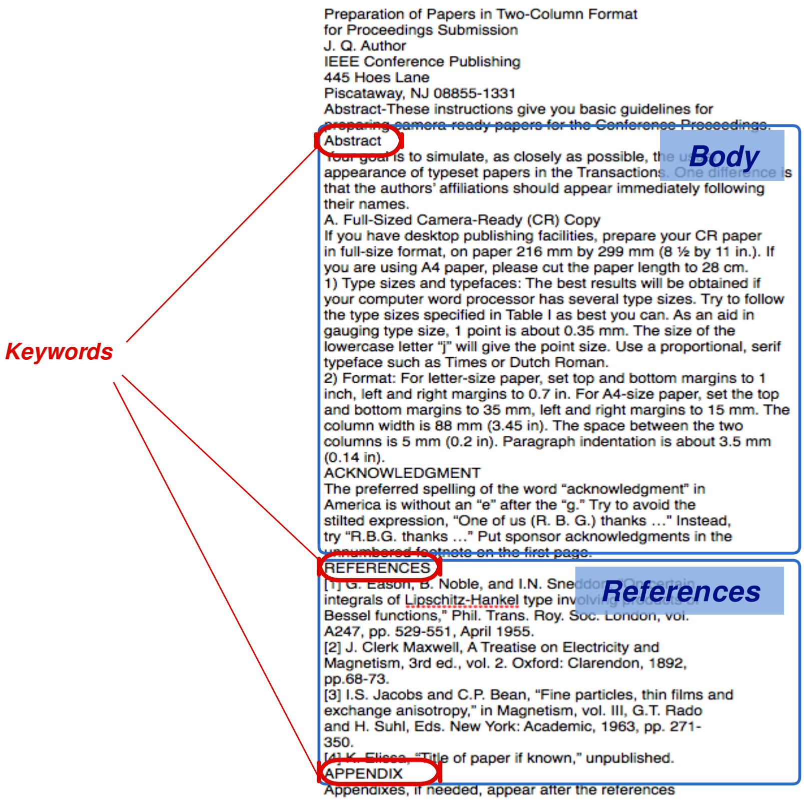


Figure 4.4: Sample text of the parsed scientific article. Keywords help to break the document into sections.

Next, we search for the citations in the body and for the bibliography references in the references section. When an author cites a work she puts a link to a bibliographic reference in the sentence. It is common to use square brackets ([ ]) to link to a bibliographic reference from the sentence. Thus, we can identify citations by detecting square brackets in the text. After analyzing some set of articles we found multiple patterns in using square brackets for citations, as shown in Table 4.2.

We also need to extract bibliographic references from the references section. For that we studied most common variants of composing the references sections. Table 4.3 summarizes these findings. To extract bibliographic references we used a list of regular expressions matching the patterns listed in Table 4.3. By parsing identifiers (e.g. numbers in square brackets) from bibliographic references we can match citations with bibliographic references using bibliographic

Patterns of using [ ]	Example in text
[21]	Our conclusion is that, contrary to prior pessimism **[21]**, [22], data mining static code attributes to learn defect predictors is useful.
[20, 3, 11, 17]	In the nineties, researchers focused on specialized multivariate models, i.e., models based on sets of metrics selected for specific application areas and particular development environments **[20, 3, 11, 17]**.
[24, Sections 6.3 and 6.4]	Details on the life-cycle of a bug can be found in the BUGZILLA documentation **[24, Sections 6.3 and 6.4]**.
[PJe02]	In a lazy language like Haskell **[PJe02]** this is not an issue - which is one key reason Haskell is very good at defining domain specific languages.

Table 4.2: Frequent patterns in using square brackets ([and])for citing

links.

The pipeline of the PDF processing stage described above is shown in Figure 4.5. The last step in the PDF processing stage is extracting titles from bibliographic references. The objective point of extracting titles from bibliographic references is to collect citations referring to the the same source (scientific article). In general case, different formats of bibliographic references can identify the same source or scientific article. For example, an article may have different editions, published in different journals in different years or simply different authors may use different style formatting. What we consider to be identical for all bibliographic references citing the same paper is the paper's title.

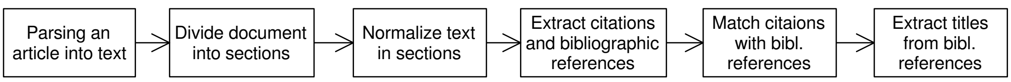


Figure 4.5: Pipeline of the PDF processing stage

References section templates
[1] J. Bach. Useful features of a test automation system (partiii) . . . **[2]** B. Beizer. Black-Box Testing. John Wiley and Sons, ...
1. J. R. Hobbs, Granularity, Ninth International Joint Conference . . . **2.** W. Woods, What's in a Link: Foundations for Semantic Networks, ...
[1]. Arnold, R.S., Software Reengineering, ed. . . . **[2].** Larman, C., Applying UML and Patterns. 1998, ...
[ASU86] A. V. Aho, R. Sethi, and J. D. Ullman. Compilers: . . . **[AU73]** A.V. Aho and J.D. Ullman. The theory of parsing, translation . . . . . .

Table 4.3: Frequent patterns of writing bibliographic references in a references section

Processing bibliographic references We try to recognize common patterns covering the majority of bibliographic references. Table 4.4 shows some examples of bibliographic references. First, we noticed that if a bibliographic reference contains some sort of quotation marks, for example, double quotes (“”) or single quotes (‘’), then it is highly probable that a title is enclosed by these quotes. Then, we made some observations for bibliographic references without quotes. Very often, a bibliographic reference is structured as follows: it begins by listing the paper's authors, then the title, and then comes the rest of the reference (see Figure 4.6). We use Core NLP library to break a reference into parts according to our view. In most of cases it is enough to take the second part of the bibliographic reference to be a title.

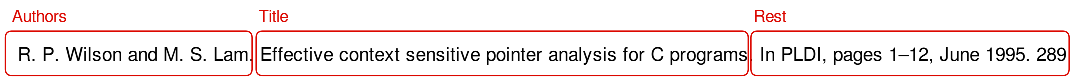


Figure 4.6: Common structure of a bibliographic reference

4.2.2 Document Publishing

There are two systems where documents are published to: Solr and MongoDB. Solr corresponds to the Indexer and Indexes Storage components and MongoDB corresponds to the Meta Data Storage component in Figure 4.1. We use Solr for indexing citations and storing indexes. We use MongoDB for aggregating citations referring to the same source paper.

Conradi, R., Dyba, T., Sjoberg, D.I.K., and Ulsund, T., “Lessons learned and recommendations from two large norwegian SPI programmes.” Lecture notes in computer science, 2003, pp. 32-45.”
P. Molin, L. Ohlsson, ‘Points & Deviations - A pattern language for fire alarm systems,’ to be published in Pattern Languages of Program Design 3, Addison-Wesley.
R. P. Wilson and M. S. Lam. Effective context sensitive pointer analysis for C programs. In PLDI, pages 112, June 1995. 289
Allen, Thomas B. Vanishing Wildlife of North America. Washington, D.C.: National Geographic Society, 1974.
I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Towards a Theoretical Model for Software Growth. In Proceedings of the 4th International Workshop on Mining Software Repositories, Minnesotta, USA, May 2007.

Table 4.4: Some examples of bibliographic references

The data stored in Solr is very ‘flat’, which means that Solr cannot store hierarchical data [32][28]. In our case, along with the references, we intend to store the title of the scientific article parsed from the reference string, so we can aggregate citations referring to the same scientific article. We are also interested in a solution that does not require reviewing all Solr documents to find citations referring to the same scientific article as it will be too slow and will decrease the quality of the user experience. Thus we use an external storage solution that can keep the titles of scientific articles and all the citations referring to a specific article. As there are few relations in our data and we would like to have a scalable solution we decided to use MongoDB as an external storage.

Publishing documents to Solr The common way to interact with Solr is using a REST API[2]. Solr provides client libraries for many programming languages to handle interactions with Solr's REST API. In our project we used the SolrJ[3] client library for Java language. The basic Solr storage unit is called document. For every detected citation we compose a document to publish. Figure 4.7 represents a structure of documents we publish to Solr.

Every document representing one citation consists of the following fields:

- id: document unique id, mandatory field for publishing to Solr
- text: text of the citation that we want to index
- context: citation with a text framing it, we take 1 sentence before and 1 after the citation
- path: URL of a document where citation was found

[2]`http://en.wikipedia.org/wiki/Representational_state_transfer`
[3]`https://cwiki.apache.org/confluence/display/solr/Using+SolrJ`

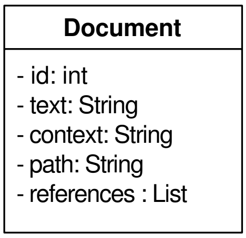


Figure 4.7: Document structure publishing to Solr

- references: list of bibliographic references from the references section matching this citation

Publishing documents to MongoDB MongoDB is a document-oriented NoSQL database that stores data in JSON-like documents with dynamic schema[4]. To connect to the database we used a Java driver provided by MongoDB. Although MongoDB is a ‘schemaless’ database we adhere to the JSON structure of the document shown in Listing 1. The JSON document consists of following fields:

- id: document id, field automatically assigned by MongoDB
- title: title of a scientific article
- citations: citations with its references of the scientific article identifying by title field

Every time we send a new citation with a paper title to MongoDB, we check if a document with the same title already exists. If so, we add a new citation to the document, otherwise we create a new document.

[4]MongoDB database, `http://www.mongodb.org/`

```
{
   "_id" : ObjectId("547ef1b219795f049d6a0ad0"),
   "title" : "Re-examining the Fault Density-Component Size  Connection",
   "citations" : [
   {
      "citation" : "Hatton, [19], claims that there is compelling empirical
                   evidence from disparate sources to suggest that in any
                   software system, larger components are proportionally more
                   reliable than smaller components.",
      "references" : [
      "[19] L. Hatton, Re-examining the Fault Density-Component Size ..."
      ]
   },
   {
      "citation" : "Hatton examined a number of data sets, [15], [18] and
                   concluded that  there was evidence of macroscopic behavior
                   common to  all data sets despite the massive internal
                   complexity of each  system studied, [19].",
      "references" : [
      "[15] K.H. Moeller and D. Paulish, An Empirical Investigation of ...",
      "[18] T. Keller, Measurements Role in Providing Error-Free Onboard ...",
      "[19] L. Hatton, Re-examining the Fault Density-Component Size  ..."
      ]
   }]
}
```

Listing 1: Sample document stored in MongoDB

4.3 Indexer

We use Solr for indexing citations. Solr is a software from Apache Software Foundation built on Apache Lucene. Apache Lucene is an open source, IR library that provides indexing and full text search capabilities[5]. While web search engines focus on searching content on the Web, Solr is designed to search content on corporate networks of any form. Some of the public services that use Solr as a server are Instagram (photo and video sharing social network), Netflix (movie hosting service) and StubHub.com (public entertainment events ticket reseller).

Figure 4.8 illustrates a high level architecture of Solr. Solr is distributed as a Java web application that runs in any servlet container, for example, Tomcat or Jetty. It provides REST-like web services so external applications can make queries to Solr or index documents. Once the data is uploaded, it goes through a text analysis pipeline. In this stage, different preprocessing

[5] Apache Lucene, `http://lucene.apache.org/core/`

phases can be applied to remove duplicates in the data or some document-level operations prior to indexing, or to create multiple documents from a single one. Solr comes with a variety of query parser implementations responsible for parsing the queries passed by the end user as search strings. For example, *TermQuery, BooleanQuery, PhraseQuery, PrefixQuery, RangeQuery, MultiTermQuery, FilteredQuery, SpanQuery* and others. Solr has xml configuration files (schema.xml and solrconfig.xml) to define the structure of the index and how fields will be represented and analyzed (see Appendix A.1 for Solr installation and configuration).

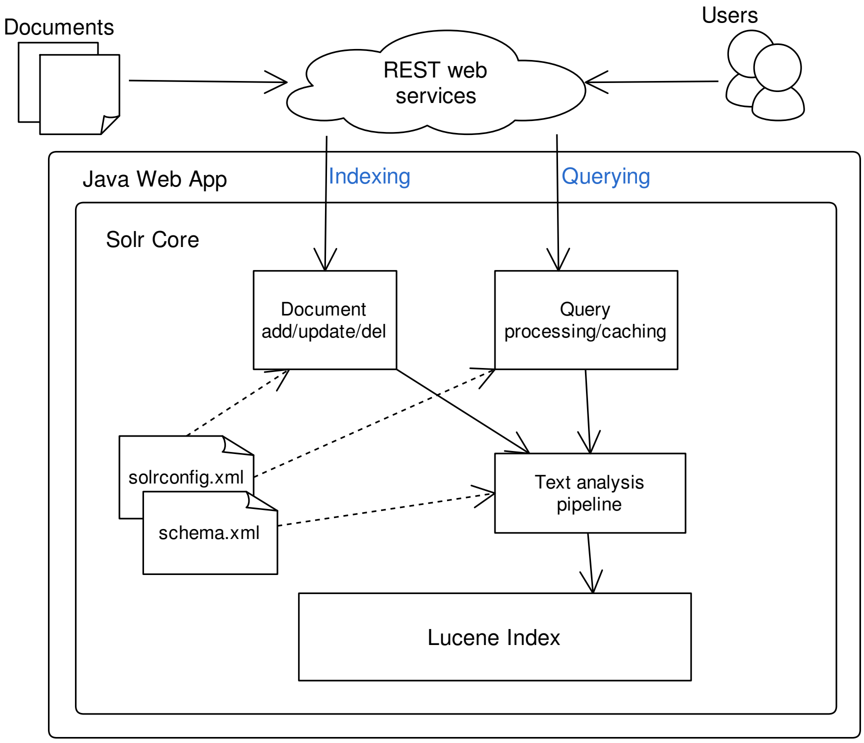


Figure 4.8: High level architecture of Solr

4.3.1 Solr's Ranking Model

Solr's ranking model is based on the Lucene scoring algorithm, also known as a TF-IDF model [18]. This model takes into consideration following factors:

- **tf** - term frequency, a frequency of the term in a document. The higher the term frequency, the higher a document score.

- **idf** - inverse document frequency, an inverse frequency of the term in all documents. The rarer the term occurs in all documents, the higher its contribution to the document's score.
- **coord** - coordination factor, takes into account the number of query terms in a document. The more query terms in a document, the higher score it has.

The exact scoring formula with the description of all factors can be found on the official web page of the Lucene documentation[6].

4.4 Web Search Interface

Web Search interface is a Java web application running in a servlet container. Figure 4.9 shows an architecture of a web search application. The application is based on the MVC (model-view-controller) architectural pattern implemented with Struts[7]. The application communicates with the Solr via Solr's REST API and with Mongo database via Java database connector.

4.4.1 CiteWise Main Page

The main page of CiteWise presents a simple search interface allowing the user to search for citations. Figure 4.10 shows a sample response to the user query "software testing is time-consuming". As a result the user sees a list of documents matching the query. Each document has a citation with a list of bibliographic links supporting this citation. A user can click to "Show context" link to see a text surrounding the citation in the original paper. If the source paper is available then user can open it using a link "See pdf on SCG resources".

If a reference has a title recognizable by CiteWise then a user can see all citations referring to the paper from this reference by clicking on the button next to the reference. Figure 4.11 demonstrates this feature. A user can see all citations of the paper "Software maintenance and evolution: a roadmap" in a popover dialog. Users can get more information about each citation by following a "View details" link.

A user can take advantage of using enhanced search query syntax. The query syntax is explained on the help page of the CiteWise interface and in the Appendix of this article.

4.4.2 Search by Bibliography Page

Another feature provided by CiteWise is the possibility to search by bibliography entries. For example, a user can search by authors, title or publication venue. An example of a search by

[6]Apache Lucene, scoring formula
[7]Struts framework, `https://struts.apache.org/`

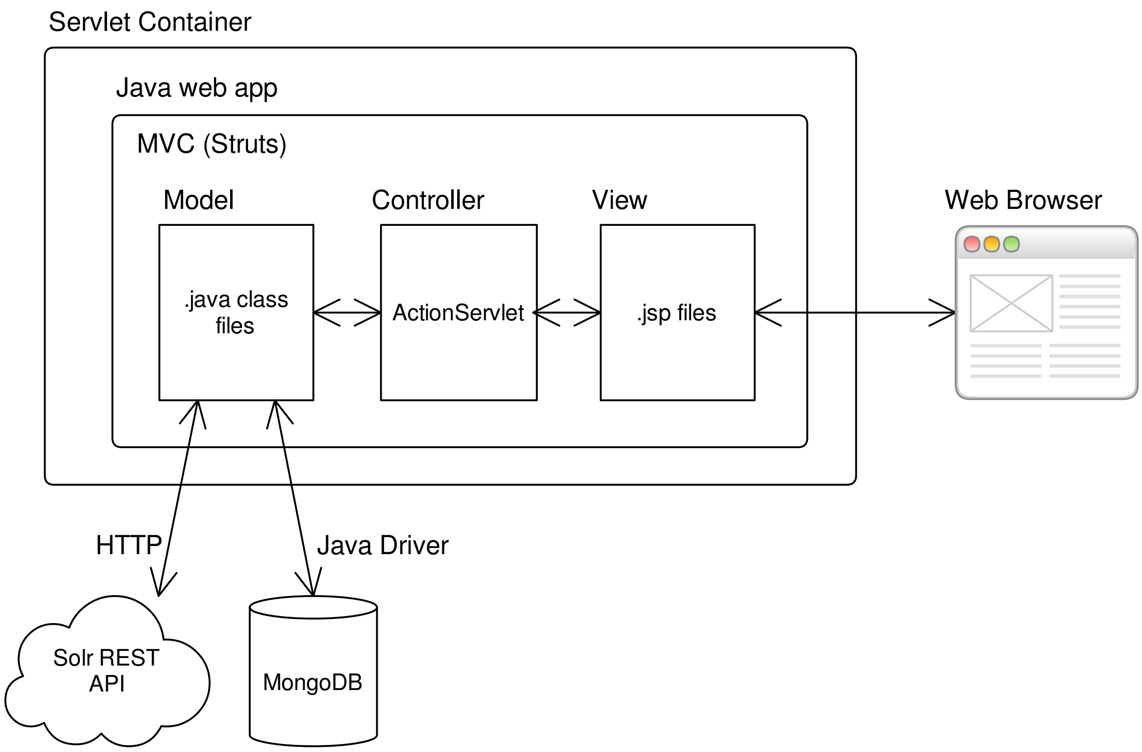


Figure 4.9: Architecture overview of a web search application

author is shown in Figure 4.12. A user sees a list of bibliographic entries with searched author name. If an entry has an extractable title then user can see citations from other papers referring to the entry.

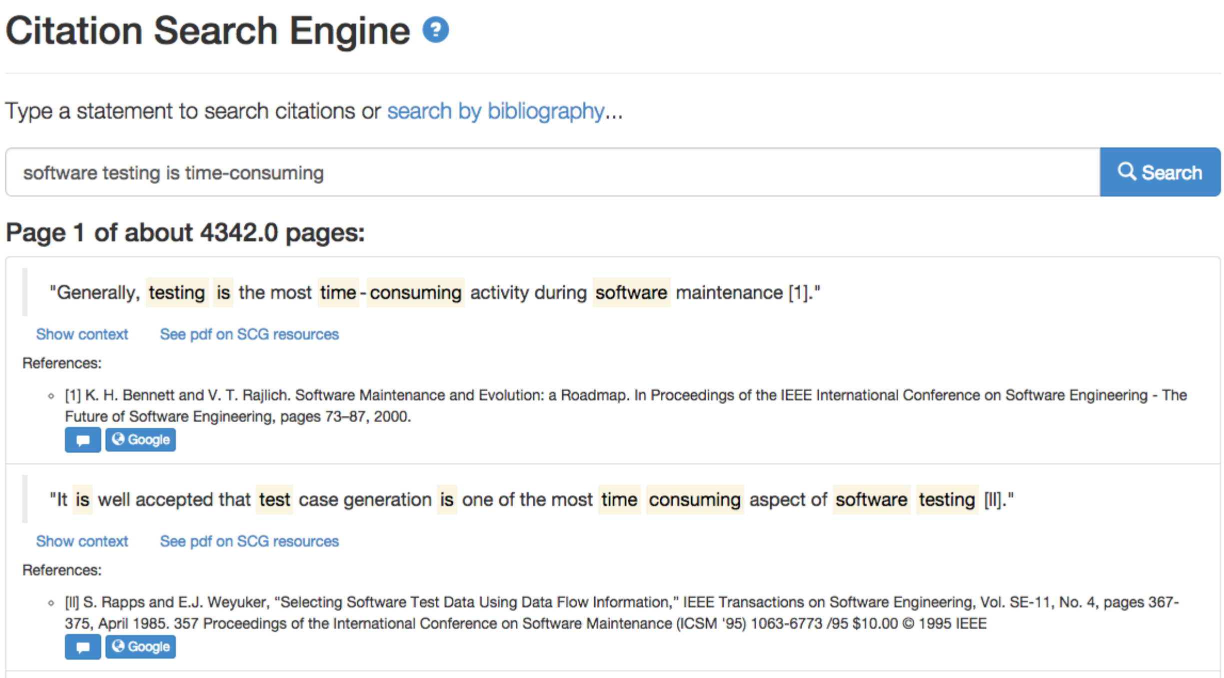


Figure 4.10: A screenshot of the main page of the CiteWise interface showing results for a statement query "software testing is time-consuming"

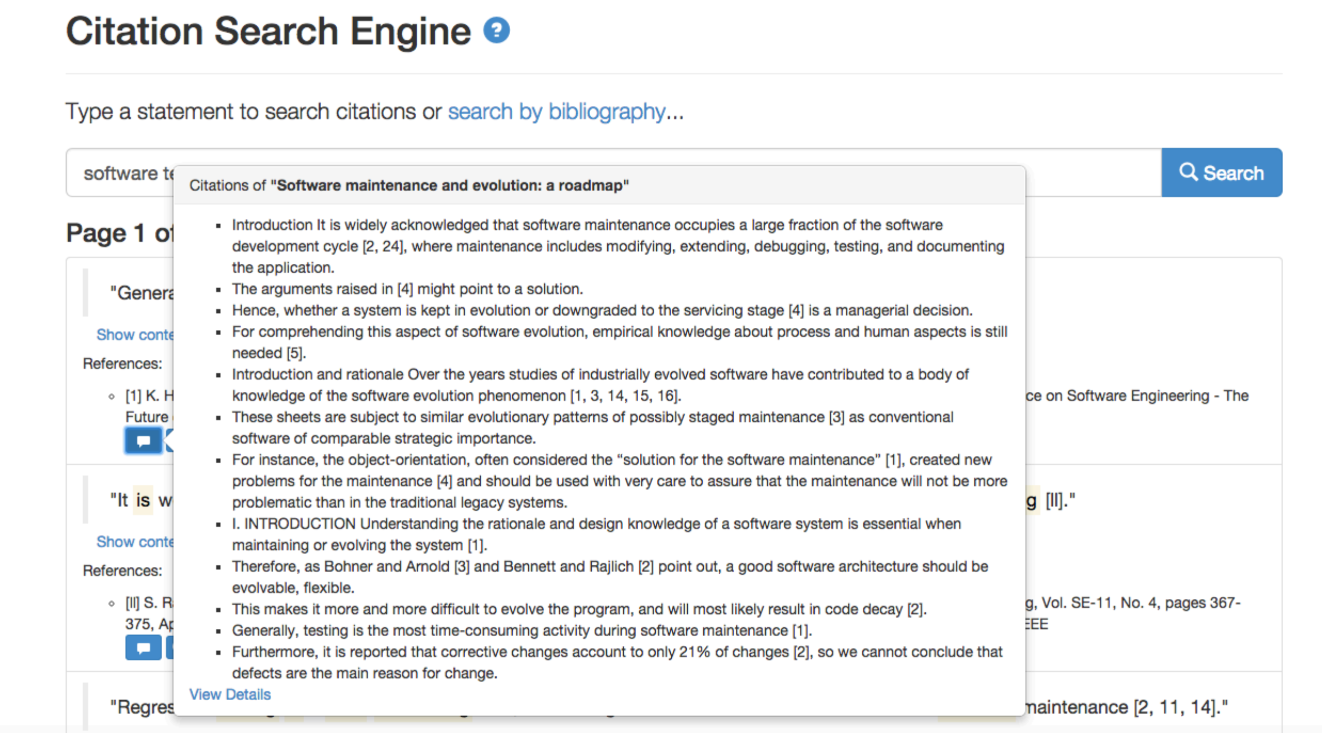


Figure 4.11: A screenshot of the main page of a CiteWise interface showing citations referring to the article with the recognizable title “Software maintenance and evolution: a roadmap.”

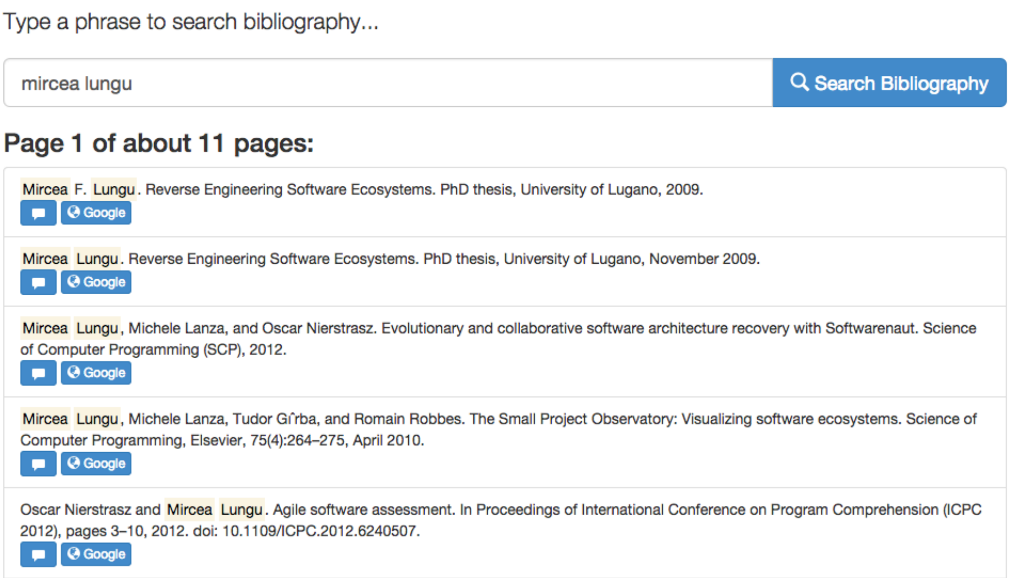


Figure 4.12: A screenshot of the 'search by bibliography' page of a CiteWise interface illustrating a search for citations based on a meta-information query. Here a user searches for citations of scientific articles authored by Mircea Lungu.

5 Evaluation

To measure the effectiveness of CiteWise we conducted evaluation experiments comparing it with other search engines. We had two main candidates to compare CiteWise with: CiteSeerX and Google Scholar. There are many aspects on how search engines might be compared. In our experiments we focused on comparing efficiency and usability of search engines. By efficiency we mean how quickly users can find documents and by usability we mean simplicity of search interfaces and personal impression. Preliminary tests showed that CiteSeerX is too slow in showing results. Moreover, users complained that resulting documents are not relevant. Too many results were from a different domain than Computer Science, like Biology or Physics. Thus, in the first part of our experiments we compared CiteWise with Google Scholar.

In the second part of our experiments, we used an aggregation feature of CiteWise to build summaries of scientific articles. We compared those summaries with summaries built using a TextRank [21] algorithm.

5.1 Experiment Setup

5.1.1 Data and Tools

For evaluation experiments we used a dataset of scientific articles collected by members of the Software Composition Group (SCG)[1] over decades. The collection contains about 16000 scientific articles and covers various topics in computer science. The Google Scholar dataset is much larger than the dataset used in our experiments, so we reduced the search space to the domain of Software Engineering and Programming Languages. During the experiment, all participants were provided with a laptop (MacBook Air, OS X version 10.10.3) and their actions were recorded with a screen casting application (QuickTime Player).

5.1.2 Participants

We intentionally looked for experts in the domain of Software Engineering and Programming Languages that can participate in experiments. Nine experts with different experiences (7 PhD candidates, 1 postdoctoral researcher, 1 professor) participated in the experiments (see Table 5.1).

ID	Position	Domains of Interest	Years of Experience
P1	Professor researcher	Software Engineering and Programming Languages	35
P2	PostDoc researcher	Software and Ecosystem Analysis	11
P3	PhD candidate	Software Quality	2
P4	PhD candidate	Ecosystem Analysis	2
P5	PhD candidate	Dynamic Analysis	2
P6	PhD candidate	Software Architecture	3
P7	PhD candidate	Development Tools	3
P8	PhD candidate	Parsing	3
P9	PhD candidate	Software Visualization	1

Table 5.1: The table describes experts which participated in the experiments, their domain of interests and academic experience in years.

5.1.3 Process

Participants were split into two groups. All experiments were conducted over two days and each day was dedicated to one group. Both groups were asked to perform the same tasks. However the second group was asked to do one more additional task (see Table 5.2). The idea of giving an

[1] http://scgresources.unibe.ch/Literature/

additional task to the second group came after conducting experiments with the first group on the first day. Time given to complete each task was limited to 5 minutes. All tasks are described in subsection 5.1.4.

Groups	Participants	Tasks to perform
Group1	P1, P3, P4, P5, P6	Task 1a, Task 2
Group2	P2, P7, P8, P9	Task 1a, Task 1b, Task 2

Table 5.2: Devision of participants by groups and tasks given to each group.

Each experiment was set up to last for approximately 45 minutes and each experiment involved only one participant. An experiment starts with a short training session, where we introduce the participant to: 1) user interfaces of both CiteWise and Google Scholar, 2) standard syntax query common to both search engines. Every task in the experiment was orally explained to the participant.

5.1.4 Tasks

Task 1a As a first task, a participant was asked to find a reference to a claim from one of his papers written in the past using CiteWise or Google Scholar. We specified the type of the search engine in the beginning of the task. A test subject can read the cited sentence as well as the context of this sentence but is not aware of the referred source paper. The task is to find a paper that proves the given claim. We use following procedure to conduct Task 1a:
Before the experiment.

1. We look for a paper published by the test subject.
2. We extract four citations from that paper.
3. We delete extracted citations from CiteWise so the test subject could not find an exact match using CiteWise.

During the experiment.

1. We let the test subject read one cited sentence as well as the context of this sentence.
2. We ask the test subject to find a referred paper that proves the given claim using the given search engine (CiteWise or Google Scholar).
3. We repeat steps for four citations every time changing the used search engine. In the first run of the Task 1a we asked the participant P1 to find a paper using CiteWise. Then, in the

second run we asked P1 to complete the same task using Google Scholar and so on. We asked the participant P2 to use Google Scholar for the first run, CiteWise for the second run and so on. Thus, we alternated a search engine type for the first run and then alternated search engine types for all remaining runs respectively.

During the execution of tasks, we observe the following:

- *Search time*, the time spent by participant to find a paper supporting the given cited text.
- *Number of queries*, the amount of queries made by participant to find a reference.
- *Number of words in each query*, numbers of words in each query made by the test subject while searching.
- *Participant comments*, any comments made by the test subject during the task execution.

Task 1b Task 1b was given to the second group as an extra task. By conducting this task, we would like to know which search engine would the test subject use if it is not specified in the task description. As in Task 1a a test subject was also given a citation to find a reference to, but this time a search engine was not specified and a citation was taken from a paper not authored by the test subject. Every participant received only one citation for this task. As for the previous task the citation was removed from CiteWise before the experiment. During Task 1b, we observed which search engine was used to find a reference.

Task 2 In Task 2 we asked participants to compare two summaries generated with CiteWise and TextRank algorithms. The TextRank algorithm is a graph-based ranking algorithm for Natural Language Processing (NLP) [20]. It extracts sentences from the text based on their importance. We use the following procedure to conduct Task 2:
Before the experiment.

1. We ask every participant in advance to provide a paper that she thinks is important in her research field.

2. We verify that a provided paper was cited at least by ten other papers in the CiteWise dataset.

3. We build a first summary using the TextRank algorithm. We use a Python implementation of this algorithm that can be found on GitHub [2]. We extract the text of a paper and feed it to TextRank. We limit the size of summaries to the size of an abstract in a paper, that is approximately 9-10 sentences.

[2] `https://github.com/adamfabish/Reduction`

4. We build a second summary using citations to the paper collected by CiteWise. CiteWise might collect more than ten citations of a paper. In this case we pick ten sentences randomly. Again we limit the size of summaries to the size of an abstract in a paper.

During the experiment.

1. We let the test subject read two summaries.
2. We ask test subjects to assess the quality of summaries by giving a score from 0 to 10.

5.2 Questionnaires

5.2.1 Pre-experiment Questionnaire

Before the beginning of the experiment we ask the test subjects to provide preliminary information by filling in a pre-experiment questionnaire. The goal of the pre-experiment questionnaire is to gather general statistics about the participants' experience in using various search engines. We ask the participants to fill in a form with questions shown in Figure 5.1.

5.2.2 Debriefing interview

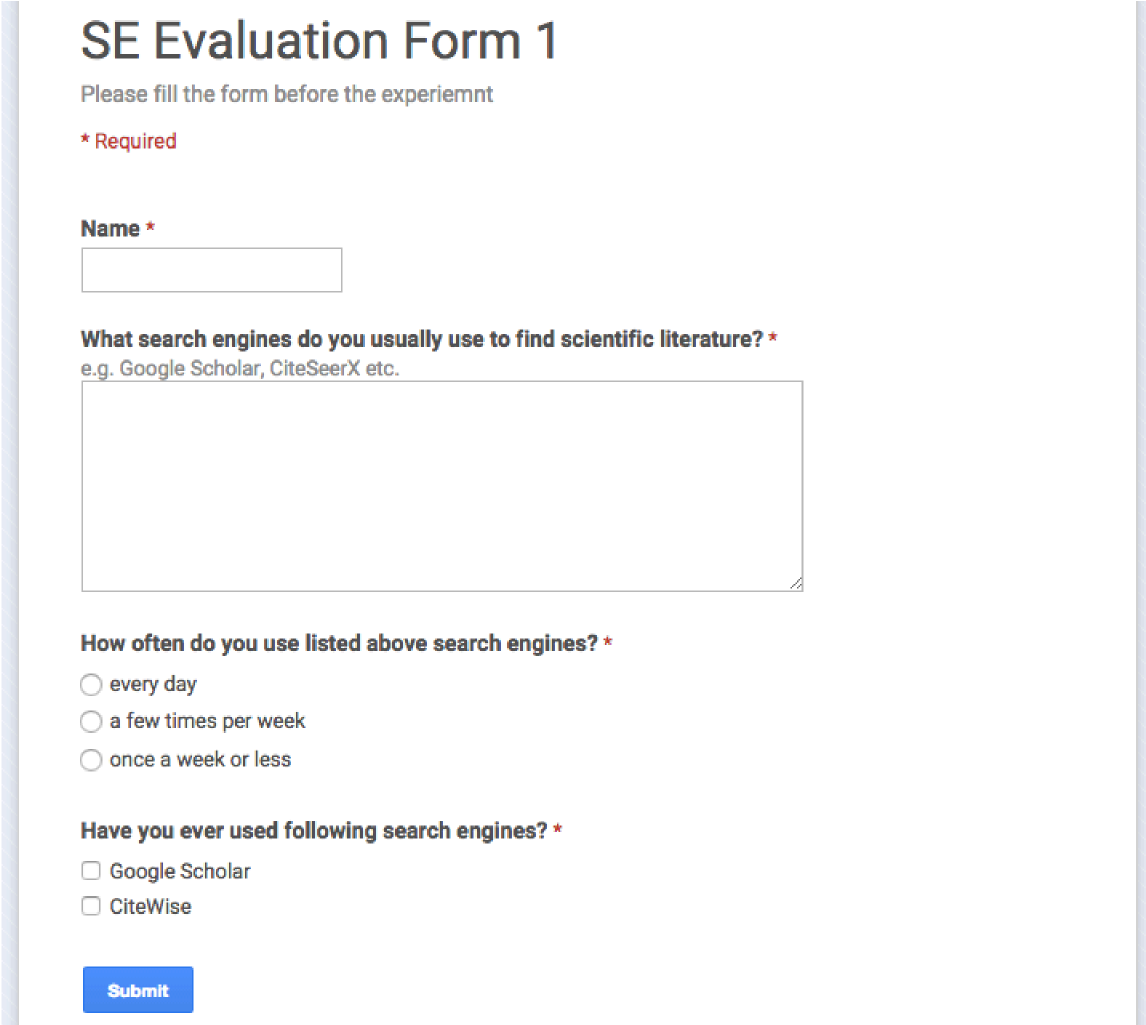
After completing Task 1a and Task 1b we conduct a semi-structured interview with participants, that lasts approximately 5 minutes. The main goal of the debriefing interview is to get an immediate feedback on using Google Scholar and CiteWise. During the interview the participants have the chance to share their impression on using both search engines. Sample questions asked during the interview: 1) What did you like/dislike about using each search engine? and 2) What difficulties did you have?

5.2.3 Post-experiment Questionnaire

Right after the experiment we ask the participant to fill in a post-experiment questionnaire. The main goal of the post-experiment questionnaire is to gather further feedback on using CiteWise and Google Scholar. We ask the participants to fill in a form with following questions: 1) Has the experiments changed your opinion on the two search engines? and 2) Would you consider using one of these search engines?

5.3 Evaluation Results

The pre-experiment questionnaire showed that almost all participants (8 experts) use Google Scholar to find scientific literature. Some participants mentioned that they use IEEE Xplore,



SE Evaluation Form 1

Please fill the form before the experiemnt

* Required

Name *

What search engines do you usually use to find scientific literature? *
e.g. Google Scholar, CiteSeerX etc.

How often do you use listed above search engines? *

- every day
- a few times per week
- once a week or less

Have you ever used following search engines? *

- Google Scholar
- CiteWise

Submit

Figure 5.1: Pre-experiment questionnaire

ACM digital library and DBLP as well (see Table 5.3). Half of the respondents (4 participants) use search engines daily, 2 respondents use search engines a few times per week (see Table 5.4).

Four respondents answered positively on the question if they have ever used CiteWise. However all of them mentioned that they used CiteWise only a few times.

5.3.1 Results for Task 1a

In Task 1a we measured search time. Each participant performed Task 1a 4 times: 2 times with CiteWise and 2 times with Google Scholar. In overall, we made 18 measurements for CiteWise and 18 measurements for Google Scholar. Figure 5.2 illustrates results for search time in Task 1a

Participants	Google Scholar	DBLP	IEEE Xplore	ACM Library
P1	✓	✓		
P2	✓	✓		
P3	✓		✓	✓
P4	✓			
P5	✓			
P6	✓			
P7			✓	✓
P8	✓			
P9	✓			
Total	8	2	2	2

Table 5.3: Pre-Experiment Questionnaire. The table shows total number of participants using particular search engine to find scientific literature.

using boxplots [3]. Table 5.5 shows the mean and standard deviation of search times for both search engines. From Table 5.5 we observed that the average time to find a reference for a given citation is approximately 2.5 minutes. Participants were slightly faster with finding results using CiteWise. However, there is no statistically significant difference between search times of CiteWise and Google Scholar according to t-test with significance level p = 5%.

Figure 5.3 illustrates results for number of queries in Task 1a using boxplots. It shows median and mean values for number of queries for both search engines. From Figure 5.3 we conclude that in 50% of cases for CiteWise in Task 1a, participants found a supporting paper in less than 2 queries.

Table 5.6 shows average values and standard deviations for number of queries made by participants to find references and average number of words in queries. We did not see any significant differences in a number of queries and average number of words in a query between two search engines. From Table 5.6 we concluded that on average participants made 2-3 queries before finding a referred paper and that the average number of words in a query was 4.

During the experiments we noticed that participants were more familiar with Google Scholar's search interface so participants spent some time exploring the CiteWise interface. This could affect search time for CiteWise making it longer.

We also noted that the way search engines present results is an important factor of the search engine usability. For example, most of the participants admitted that they like that CiteWise shows the exact place from the article where match was found. In contrast, Google Scholar shows a title of the article and a beginning of the abstract, so it is not clear where the match was found. In this case participants had to open the article and make a manual search over the text.

[3] `http://en.wikipedia.org/wiki/Box_plot`

Participants	Every day	A few times per week	Once a week or less
P1	✓		
P2		✓	
P3	✓		
P4			✓
P5	✓		
P6			✓
P7		✓	
P8			✓
P9	✓		
Total	4	2	3

Table 5.4: Pre-Experiment Questionnaire. The table shows how often participants use search engines to find scientific literature.

	Mean (sec)	Std (sec)
CiteWise	150	97
Google Scholar	160	78

Table 5.5: The mean and standard deviation of search times for CiteWise and Google Scholar in Task 1a.

5.3.2 Results for Task 1b

Task 1b was given to four participants. Table 5.7 shows in what search engine a supporting paper was found and the order in which a participant used search engines. For example, the participant P6 first searched for a paper in CiteWise, then he switched to Google Scholar and finally he switched to CiteWise where he found the result paper. From Table 5.7 we concluded that all participants found a supporting paper using CiteWise. Meanwhile three of participants used both search engines and one participant did not use Google Scholar at all.

5.3.3 Results for Task 2

Results for Task 2 are shown in Figure 5.4. It illustrates scores from 0 to 10 given by participants to summaries generated with TextRank and citation from CiteWise. All participants except one (6 participants) gave better scores to the summary composed by citations from CiteWise. We could not generate summaries for participants P4 and P6 since they did not provide us with papers. According to t-test with a significance level p = 5% there is a significant difference between scores given to summaries generated with TextRank and CiteWise. Participants noted that a summary generated with TextRank consists of sentences either too general or not important for understand-

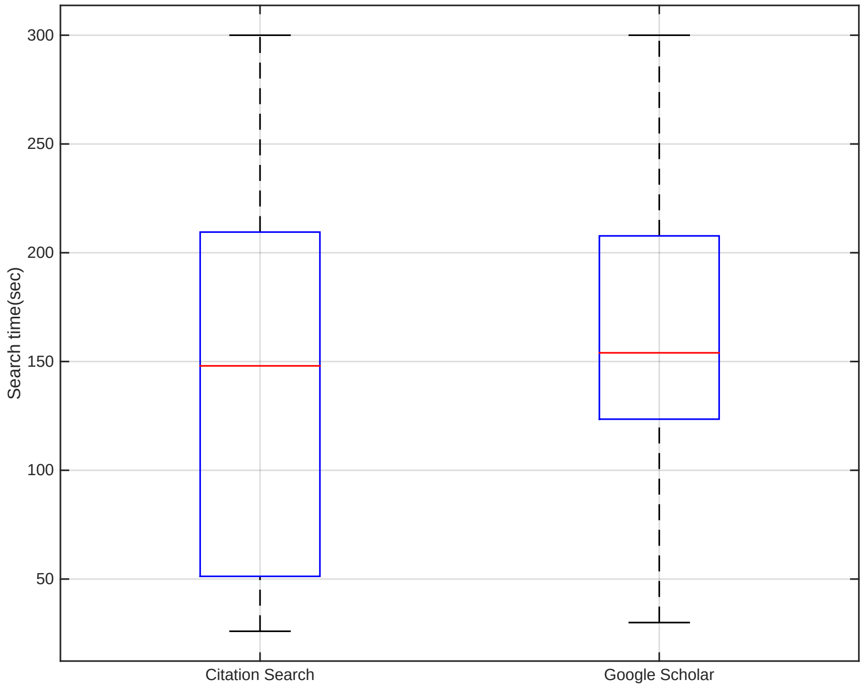


Figure 5.2: Boxplot of search times in Task 1a for both search engines. The red band inside a box is a second quartile (the median). The bottom and top of the box are the first and third quartiles. The end of whiskers represent the minimum and maximum values for search times.

ing the paper. According to participants' opinions, the summary composed with citations tends to contain sentences more relevant for understanding the summarized paper. However, sometimes cited sentences rephrase each other expressing the same idea. Also, compared to TextRank there is no natural flow in the summary from citations. In other words sentences in a summary are not ordered to make a story.

5.3.4 Final Questionnaire

The final questionnaire shows that all participants answered positively when they were asked whether they are willing to continue to use CiteWise. Some participants specified that they will use Google Scholar and CiteWise for different purposes. According to the opinion of two participants CiteWise is more appropriate to search for related work on the given topic. Others (5 participants) think that CiteWise is good to prove claims while writing a scientific paper. One participant opinion states that CiteWise is useful for discovering new works in the given domain.

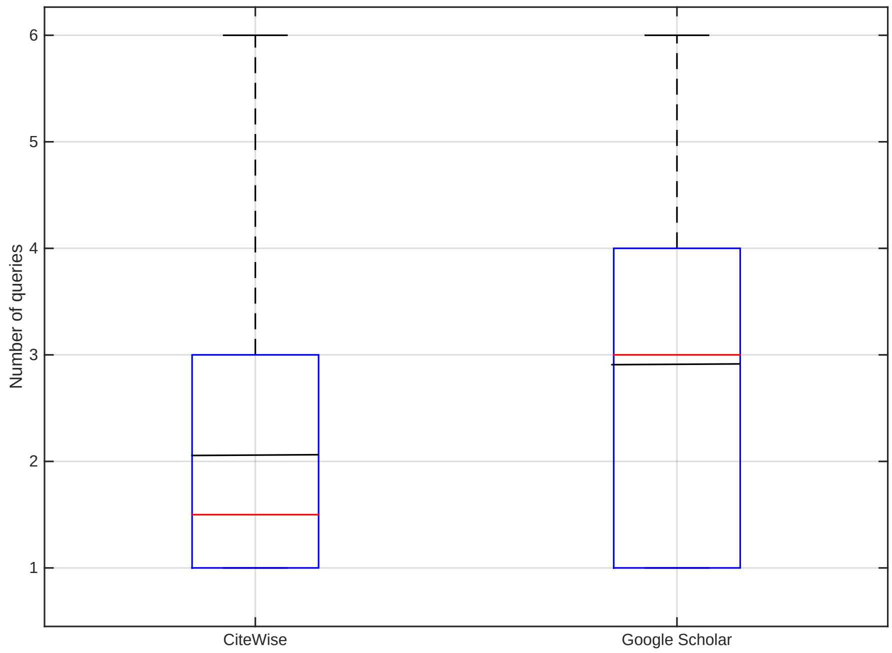


Figure 5.3: Boxplot of number of queries in Task 1a for both search engines. The red band inside a box is a second quartile (the median). The black band inside a box is a mean value. The bottom and top of the box are the first and third quartiles. The end of whiskers represent the minimum and maximum values for number of queries.

Participants appreciated the possibility to see citations with a context and the possibility to search by bibliographic entries.

5.3.5 Results Summary

In our evaluation experiments we compared CiteWise with Google Scholar. During the experiments we collected statistics on search time, number of queries and average number of words in queries. The results show that CiteWise performs slightly better for mean value of search time, but there is no statistically significant difference among search engines. We noticed that in 50% of cases participants found a supporting paper in CiteWise using only one query. Overall, given that Google Scholar is one of the most popular academic search engines, CiteWise might complement Google Scholar. Indeed, when participants have a possibility to choose between two search engines, all participants succeeded in the task accomplishment using CiteWise.

The results for the comparison show that summaries generated with citations give a better

	Mean	**Std**
Number of queries		
CiteWise	2.1	1.5
Google Scholar	2.9	1.7
Average number of words in a query		
CiteWise	4.3	1.7
Google Scholar	4.2	1.2

Table 5.6: The mean and standard deviation values for a number of queries and avearge number of words in a query in Task 1.

Participants	**How search engines were used**
P6	CiteWise, Google Scholar, CiteWise
P7	Google Scholar, CiteWise
P8	Google Scholar, CiteWise
P9	CiteWise

Table 5.7: The table illustrates results for Task 1b. It shows the order in which participants used search engines during the task accomplishment. The search engine where a supporting paper was retrieved is marked red.

description of a paper. The automatic citation aggregation feature of CiteWise could be used to generate summaries or even judge the importance of a paper, for example, by counting the number of citations.

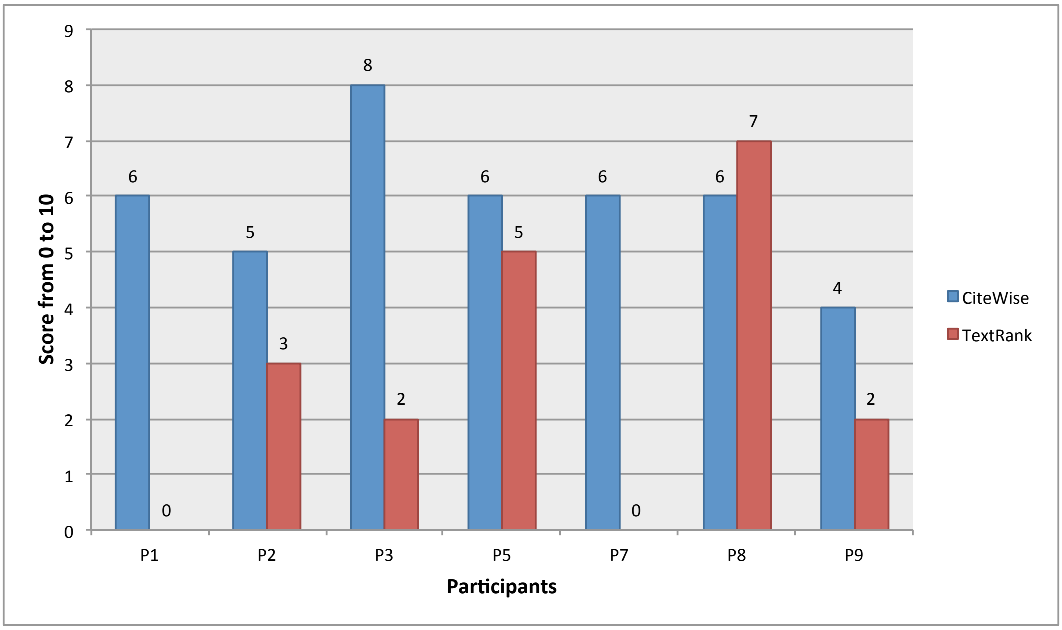


Figure 5.4: Scores from 0 to 10 given by participants in Task 2 to the summaries generated with TextRank and citations from CiteWise.

6
Conclusion

In this work we address the problem of IR for scientific articles. We believe that considering meta-information helps us to build enhanced search systems. We particularly focused on citations considering them as important text blocks. We designed and implemented CiteWise which automatically extracts and indexes citations from scientific articles in PDF format. Moreover we studied the structure of citations and built an algorithm that aggregates citations referring to the same source. We used this feature of CiteWise to generate automatic summaries of papers.

We evaluated our system by conducting user evaluation experiments. In the first part of our experiments, we compared our system with the popular academic search engine, Google Scholar. We observed how fast users are in finding results using both search engines. Our results showed that CiteWise performs equal to or better than Google Scholar. In the second part of our experiments, we used an aggregation feature of CiteWise to build summaries of scientific articles. We compared those summaries with summaries built using a TextRank algorithm. Our results showed that CiteWise gives a better description of scientific articles according to participant's opinion.

7
Future Work

One of the ways the CiteWise parser can be improved is finding cited sentences that do not contain any specific identifiers. Indeed the task is straightforward when a sentence contains square brackets, for example, '[34]' or '[Ali86]'. However, sometimes a link to bibliography can be composed only from the authors' names. In this case it becomes difficult to distinguish a citation from any other sentence (see Figure 7.1).

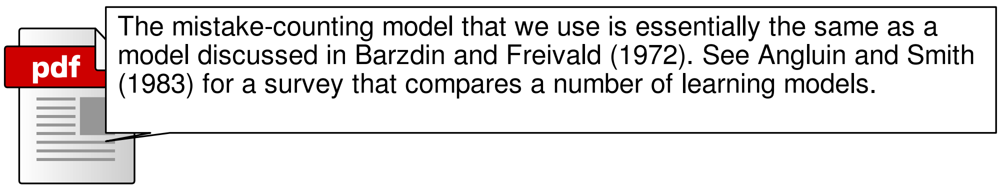


Figure 7.1: An example of citations from the article "Learning Abound: Quickly When Irrelevant Attributes A New Linear-threshold Algorithm" authored by Nick Littlestone. In both sentences a link to a bibliographic reference composed from authors' names makes it hard to distinguish a citation from any other sentences.

Another issue in using square brackets as citation identifiers arises when square brackets are used not as links to bibliography. For example, a parser might mix up an array in a code snippet with a link to bibliography (see Figure 7.2). Usage of code snippets are common in computer science literature so it would be nice to have a method to distinguish a snippet of source code from natural text.

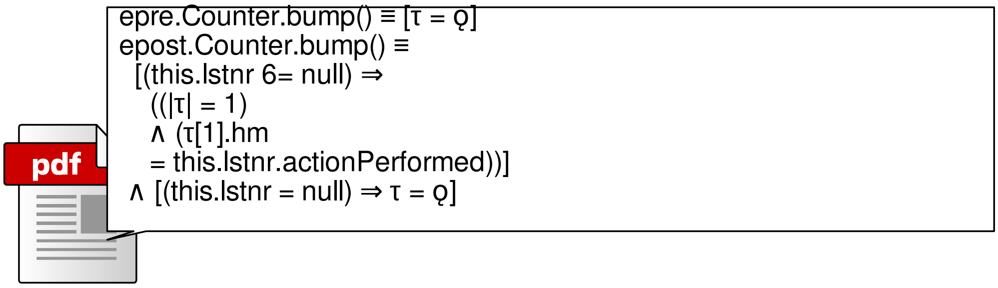


Figure 7.2: An example of the code snippet from the article "Modular Verification of Higher-Order Methods with Mandatory Calls Specified by Model Programs" authored by Steve M. Shaner *et al.* The code snippet is wrongly considered as a citation and is matched to the first bibliographic entry.

Bibliography

[1] K. Aberer, A. Boyarsky, Philippe Cudré-Mauroux, Gianluca Demartini, and O. Ruchayskiy. Sciencewise: A web-based interactive semantic platform for scientific collaboration. In *10th International Semantic Web Conference (ISWC 2011-Demo), Bonn, Germany*, 2011.

[2] Marc Bertin and Iana Atanassova. Semantic enrichment of scientific publications and metadata : Citation analysis through contextual and cognitive analysis. *D-Lib Magazine*, 18:8, 2012.

[3] Marc Bertin and Iana Atanassova. Extraction and characterization of citations in scientific papers. In Valentina Presutti, Milan Stankovic, Erik Cambria, Ivn Cantador, Angelo Di Iorio, Tommaso Di Noia, Christoph Lange, Diego Reforgiato Recupero, and Anna Tordai, editors, *Semantic Web Evaluation Challenge*, volume 475 of *Communications in Computer and Information Science*, pages 120–126. Springer International Publishing, 2014.

[4] Marc Bertin, Jean-Pierre Desclés, Brahim Djioua, and Yordan Krushkov. Automatic annotation in text for bibliometrics use. In *FLAIRS Conference*, pages 313–318, 2006.

[5] Shannon Bradshaw. Reference directed indexing: Redeeming relevance for subject search in citation indexes. In Traugott Koch and IngeborgTorvik Slvberg, editors, *Research and Advanced Technology for Digital Libraries*, volume 2769 of *Lecture Notes in Computer Science*, pages 499–510. Springer Berlin Heidelberg, 2003.

[6] Shannon Glenn Bradshaw. *Reference directed indexing: Indexing scientific literature in the context of its use*. Northwestern University, 2002.

[7] Soumen Chakrabarti, Martin Van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11):1623–1640, 1999.

[8] Isaac G. Councill, C. Lee Giles, and Min yen Kan. Parscit: An open-source CRF reference string parsing package. In *International Language Resources and Evaluation*. European Language Resources Association, 2008.

[9] Gaizka Garechana, Rosa Rio, Ernesto Cilleruelo, and Javier Gavilanes. Visualizing the scientific landscape using maps of science. In Suresh P. Sethi, Marija Bogataj, and Lorenzo Ros-McDonnell, editors, *Industrial Engineering: Innovative Networks*, pages 103–112. Springer London, 2012.

[10] Eugene Garfield et al. Science citation index — a new dimension in indexing. *Science*, 144(3619):649–654, 1964.

[11] Roya Ghafele, Benjamin Gibert, and Paul DiGiammarino. Driving innovation through patent application review: The power of crowdsourcing prior art search. *Journal of Intellectual Property Rights*, 16:303–308, 2011.

[12] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. CiteSeer: An Automatic Citation Indexing System. In *Proceedings of the Third ACM Conference on Digital Libraries*, DL '98, pages 89–98, New York, NY, USA, 1998. ACM.

[13] The Stanford Natural Language Processing Group. Stanford CoreNLP API. http://nlp.stanford.edu/software/corenlp.shtml.

[14] M. M. Kessler. Bibliographic coupling between scientific papers. *American Documentation*, 14(1):10–25, 1963.

[15] Richard Klavans and Kevin W. Boyack. Toward a consensus map of science. *J. Am. Soc. Inf. Sci. Technol.*, 60(3):455–476, March 2009.

[16] Ray R Larson. Bibliometrics of the world wide web: An exploratory analysis of the intellectual structure of cyberspace. In *PROCEEDINGS OF THE ANNUAL MEETING-AMERICAN SOCIETY FOR INFORMATION SCIENCE*, volume 33, pages 71–78, 1996.

[17] Loet Leydesdorff, Stephen Carley, and Ismael Rafols. Global maps of science based on the new web-of-science categories. *CoRR*, abs/1202.1914, 2012.

[18] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[19] Qiaozhu Mei and ChengXiang Zhai. Generating impact-based summaries for scientific literature. In *Proceedings of ACL-08: HLT*, pages 816–824, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[20] Rada Mihalcea. Graph-based ranking algorithms for sentence extraction, applied to text summarization. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 20. Association for Computational Linguistics, 2004.

[21] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. Association for Computational Linguistics, 2004.

[22] Preslav I. Nakov, Ariel S. Schwartz, and Marti A. Hearst. Citances: Citation sentences for semantic analysis of bioscience text. In *In Proceedings of the SIGIR04 workshop on Search and Discovery in Bioinformatics*, 2004.

[23] Hidetsugu Nanba and Manabu Okumura. Towards multi-paper summarization reference information. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 926–931, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[24] Enrique Orduña Malea, Juan M. Ayllón, Alberto Martín-Martín, and Emilio Delgado López-Cózar. About the size of google scholar: playing the numbers, July 2014.

[25] M. F. Porter. *Readings in Information Retrieval*, chapter An Algorithm for Suffix Stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[26] Roman Prokofyev, Alexey Boyarsky, Oleg Ruchayskiy, Karl Aberer, Gianluca Demartini, and Philippe Cudré-Mauroux. Tag recommendation for large-scale ontology-based information systems. In *Proceedings of the 11th international conference on The Semantic Web-Volume Part II*, pages 325–336. Springer-Verlag, 2012.

[27] Vahed Qazvinian and Dragomir R. Radev. Scientific paper summarization using citation summary networks. In *Proceedings of the 22Nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 689–696, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[28] Jonathan Rochkind. Thinking like Solr its not an rdbms. https://bibwild.wordpress.com/2011/01/24/thinking-like-solr-its-not-an-rdbms/.

[29] Henry Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24(4):265–269, 1973.

[30] Henry Small. Visualizing science by citation mapping. *J. Am. Soc. Inf. Sci.*, 50(9):799–813, July 1999.

[31] Linda C. Smith. Citation analysis. `https://www.ideals.illinois.edu/bitstream/handle/2142/7190/librarytrendsv30i1i_%20opt.pdf?sequence=1`, 1981.

[32] Solr Wiki. Why use Solr? http://wiki.apache.org/solr/WhyUseSolr.

A
User Guide for CiteWise Deployment

A.1 Solr Installation

Solr installation requires JDK and any servlet container to be installed on the server machine. Here we describe the configuration of Solr for Apache Tomcat container. We need to download the Solr distribution that can be found on the official Solr home page [1]. Solr is distributed as an archive. After unzipping the archive, the extracts have following directories:

- **contrib/** - directory containing extra libraries to Solr, such as Data Import Handler, MapReduce, Apache UIMA, Velocity Template, and so on.
- **dist/** - directory providing distributions of Solr and some useful libraries such as SolrJ.
- **docs/** - directory with documentation for Solr.
- **example/** - Jetty based web application that can be used directly.
- **Licenses/** - directory containing all the licenses of the underlying libraries used by Solr.

Copy the dist/solr.war file from the unzipped folder to $CATALINA_HOME/webapps/solr.war. Then point out to Solr location of home directory describing a collection:

[1] Apache Solr, `http://lucene.apache.org/solr/`

- **Java options:** one can use following command so that the container picks up Solr collection information from the appropriate location:

```
$export JAVA_OPTS="$JAVA_OPTS -Dsolr.solr.home=/opt/solr/example"
```

By a collection in Apache Solr one indicates a collection of Solr documents that represents one complete index.

The Solr home directory contains configuration files and index-related data. It should consist of three directories:

- **conf/** - directory containing configuration files, such as solrconfig.xml and schema.xml
- **data/** - default location for storing data related to index generated by Solr
- **lib/** - optional directory for additional libraries, used by Solr to resolve any plugins

A.1.1 Solr Configuration

Configuring Solr instance requires defining a Solr schema and configuring Solr parameters.

Defining Solr schema A Solr schema is defined in the schema.xml file placed in the conf/ directory of the Solr home directory. The Solr distribution comes with a sample schema file that can be changed for the needs of the project. The schema file defines the structure of the index, including fields and field types. The basic overall structure of the schema file is:

```
<schema>
  <types>
  <fields>
  <uniqueKey>
  <copyField>
</schema>
```

The basic unit of data in Solr is document. Each document in Solr consists of fields that are described in the schema.xml file. By describing data in the schema.xml, Solr understands the structure of the data and what actions should be performed to handle this data. Here is an example of a field in the schema file:

```
<field name="id" type="integer" indexed="true" stored="true" required="true"/>
```

Table A.1 lists and explains major attributes of field element.

Here is a fragment of schema file defining fields of a document in CiteWise collection:

Name	Description
default	default value if it is not read while importing a document
indexed	true if field should be indexed
stored	when true a field is stored in index store and is accessible while displaying results
compressed	when true a field will be zipped, applicable for text-type fields
multiValued	if true, field can contain multiple values in the same document.

Table A.1: Major attributes of field element in a schema.xml file

```
<fields>
 <field name="_version_" type="long" indexed="true" stored="true"
     multiValued="false"/>
 <field name="id" type="string" multiValued="false"/>
 <field name="text" type="text_en" indexed="true" multiValued="false"/>
 <field name="context" type="string" indexed="false" multiValued="false"/>
 <field name="path" type="string" indexed="false" multiValued="false"/>
 <field name="reference" type="string" indexed="false" stored="true"
     multiValued="true" />
</fields>
```

Every document represents a citation with matching bibliographic references. In the schema file we indicate that we want to index a text field which is the citation text. We store an id of a citation, that is a generated value, calculated from the hash of the citation string. Specifying the id is particularly useful for updating documents. We also store a context for a citation and a path to the scientific article where the citation was found. As a citation can refer to multiple sources, we make the reference field multivalued.

In the schema configuration file, one can define the field type, like string, date or integer and map them to Java classes. This can be handy when we define custom types. A field type includes the following information:

- Name
- Implementation class name
- If the field type is a TextField, it will include a description of the field analysis
- Field attributes

A sample field type description:

```
<fieldType name="text_ws" class="solr.TextField" positionIncrementGap="100">
   <analyzer>
   <tokenizer class="solr.WhitespaceTokenizerFactory"/>
   </analyzer>
</fieldType>
```

Other elements in the Solr schema file listed in Table A.2:

Name	Description
uniqueKey	specifies which field in documents is a unique identifier of a document, should be used if you ever update a document in the index
copyField	used to copy a field value from one field to another

Table A.2: Description of some elements in schema.xml

Configuring Solr Parameters To configure a Solr instance we need to describe the solrconfig.xml and solr.xml files.

solr.xml The solr.xml configuration is located in solr home directory and used for configuration of logging and advanced options to run Solr in a cloud mode.

solrconfig.xml The solrconfig.xml configuration file primarily provides you with an access to index-management settings, RequestHandlers, listeners, and request dispatchers. The file has a number of complex sections and mainly is changed when a specific need is encountered.

A.1.2 Enhanced Solr Search Features

Solr provides a number of additional features that can enhance the search system. One of the features we use is synonyms. To use this feature you need to specify synonyms.txt file with listed synonyms. This file is used by synonym filter to replace words with their synonyms. For example, a search for ”DVD” may expand to ”DVD”, ”DVDs”, ”Digital Versatile Disk” depending on the mapping in this file. This file can be also used for spelling corrections. Here is an example of synonyms.txt file:

```
GB, gib, gigabyte, gigabytes
MB, mib, megabyte, megabytes
Television, Televisions, TV, TVs
Incident_error, error
```

Additionally, there are other configuration files that appear in the configuration directory. We are listing them in Table A.3 with the description of each configuration:

Name	Description
protwords.txt	file where you can specify protected words that you do not wish to get stemmed. So, for example, a stemmer might stem the word ”catfish” to ”cat” or ”fish”.
spellings.txt	file where you can provide spelling suggestions to the end user.
elevate.txt	file where you can change the search results by making your own results among the top-ranked results. This overrides standard ranking scheme, taking into account elevations from this file.
stopwords.txt	Stopwords are those that will not be indexed and used by Solr in the applications. This is particularly helpful when you really wish to get rid of certain words. For example, in the string, ”Jamie and joseph,” the word ”and” can be marked as a stopword.

Table A.3: Additional configuration files in Solr

A.2 MongoDB Installation

MongoDB is a NoSQL document-oriented database. Data in MongoDB is stored in JSON-like documents with a dynamic schema. The format of stored data is called BSON, which stands for Binary JSON. BSON is an open standard developed for human readable data exchange[2]. MongoDB requires a little amount of configuration to start to work with.

To install MonogoDB follow instruction on the official web site `http://docs.mongodb.org/manual/installation/`.

A.2.1 MongoDB configuration

Once the MonogDB distribution the is downloaded, it is very easy to set up a database server. All we need to start the MongoDB server is to type *mongod* command. In our case we would like to specify database location with –dbpath parameter and default listening port:

```
> mongod --dbpath /home/aliya/mongodb2 --port 27272
```

MongoDB provides REST API. To enable REST API use parameter –rest:

```
> mongod --dbpath /home/aliya/mongodb2 --port 27272 --rest true
```

[2]BSON specification, `http://bsonspec.org/`

The simple way to communicate with the MongoDB server is to use the MongoDB shell, in our case we specify –port parameter to connect to our instance of MongoDB:

```
> mongo --port 27272
```

Compared to relational databases MongoDB operates with a collection, which is equivalent to a table, and a document, which is equivalent to a record in relational databases. MongoDB does not require creating databases and collections explicitly. Databases and collections can be created while starting to use MongoDB. To see list of databases or collections, type show dbs in mongo shell:

```
> show dbs
```

MongoDB shell allows one to make queries, updates, deletes on collections, get various statistics on data and server usage, and manipulate with data with map-reduce interface, full documentation can be found on the official web site[3].

A.3 Running the parser

Before running the parser the Solr web application should be deployed on the Tomcat web server and the MongoDB instance should be run. One should use Java version 7 or above to run the parser. Get the parser distribution:

```
> git clone git@scg.unibe.ch:citation-search-engine
```

The cloned directory consists of three modules:

- **solr** - Solr related configuration files,
- **citation_search** - a parser of scientific articles, that extracts meta-information and publish documents to Solr and MongoDB,
- **citation_search_web** - a web application for searching citations.

All files related to the parser are located in the *citation_search* directory. The *citation_search* directory has a standard Maven project layout[4]. Go to the resources *parser.properties* according to your development environment. Table A.4 describes properties of a *parser.properties* file with sample values.

One can change default logging properties for the Log4j library in a *logj4.properties* file. Once property files are configured, build a jar file executing following command from the directory containing a *pom.xml* file:

[3] MongoDB database, http://www.mongodb.org/
[4] Apache Maven, https://maven.apache.org

Property	Description	Sample value
solr.url.citations	Endpoint for publishing citations.	http://localhost:8088/solr/collection1/
solr.url.bibliography	Endpoint for publishing bibliographic references.	http://localhost:8088/solr/collection2/
db.host	MongoDB host server IP address.	127.0.0.1
db.port	MongoDB listening port.	27272
db.name	MongoDB database name.	CS
db.collection	MongoDB database collection name.	papers
pdfs.path	Location of pdf files.	/home/aliya/Library

Table A.4: Explanation of properties of a *parser.properties* file.

```
> mvn assembly:assembly -DdescriptorId=jar-with-dependencies DskipTests
```

Maven will generate a jar file *citation_search-1.0-jar-with-dependencies.jar* in a *target* folder. To execute the jar file run following command:

```
> java -jar citation_search-1.0-jar-with-dependencies.jar
```

A.4 Search Interface Deployment

All web application related web files are located in a *citation_search_web* directory. The directory has a standard Maven project layout [5]. Change a *search.properties* file in a *resources* folder. Table A.5 describes properties of a *search.properties* file with sample values .

Property	Description	Sample value
solr.url.citations	Endpoint for querying citations.	http://localhost:8088/solr/collection1/
solr.url.bibliography	Endpoint for querying bibliographic references.	http://localhost:8088/solr/collection2/

Table A.5: Explanation of properties of a *search.properties* file.

One can change default logging properties for Log4j [6] library in a *logj4.properties* file. Once

[5] Apache Maven, https://maven.apache.org
[6] Apache Log4j, `http://logging.apache.org/log4j/2.x/`

property files are configured, build a war file executing following command from the directory containing a *pom.xml* file:

```
> mvn package -DskipTests
```

Maven will generate a war file in a *target* folder. Deploy to the Tomcat web server by putting a warfile in a Tomcat *webapp* directory or use a Tomcat web interface to deploy through Tomcat's manager.