

Icecite: A fully web-based research paper manager.

Hannah Bast, Claudius Korzen
Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{bast, korzen}@informatik.uni-freiburg.de

ABSTRACT

Some impressive abstract. Process of identifying metadata and references is splitted into extraction and matching.

1. INTRODUCTION

Literature research is one of the most essential task to accomplish in a regular scientific research process. However, it's usually a complex and time-consuming task. That's why we have implemented *Icecite*, a fully web-based research manager aimed to automatize and to facilitate the necessary steps of a *digital* literature research process. *Digital* means that Icecite concentrates on managing digital literary materials, namely scientific research papers provided as PDF files.

One of the core features of Icecite is the automatic identification of both, the metadata (title, author(s), year, journal, etc.) and the bibliographic references in given research papers. Our approach splits the identification process into 2 steps. In the first step, we extract significant data from given PDF files (e.g. the titles and the entries of bibliographies). In the second step, we match each extract to the referred record of an underlying digital library (e.g. DBLP or Medline) to get the full and correct metadata fields. Our approach outperforms traditional machine learning approaches in terms of runtime and accuracy of identified data.

On the basis of the automatic identification of data, Icecite can provide an extensive set of further features, that exceeds the feature sets of existing (mainly desktop-based) software solutions. The full feature set of Icecite includes:

- (1) Managing research papers in a personal library.
- (2) Automatic identification of metadata of research papers.
- (3) Automatic identification of bibliographic references of research papers.
- (4) Instant search in DBLP and Medline.
- (5) Instant (fulltext-)search in personal collection of research papers.

- (6) Importing document from digital libraries into the library with a single click.
- (7) Automatic search for PDF files, providing fulltexts of research papers.
- (8) Reading & Annotating of PDF files in browser.
- (9) Tagging of research papers with keywords.
- (10) Offline usability.
- (11) Synchronization of the library data (metadata, references, PDF files, annotations).
- (12) Sharing of the library data with other member nearly in real-time.
- (13) Fully web-based and easy-to-use user interface

In the following, we will give an overview how the listed features act together to form the general structure of Icecite and how they support the following most essential tasks of a proper literature research process.

Organize collections of research papers.

Doing scientific literature research usually implies the need to manage a set of collected literary materials, i.e. digital research papers provided as PDF files as mentioned above.

The collection may consist of locally stored files or bookmarks to the locations of files in web, or both. Depending on the size of the collection, keeping a consistent structure on naming the files are recommended, in order to keep track of them. However, finding proper filenames and renaming the files are annoying and time-consuming tasks.

Icecite eliminates these steps entirely by holding all the collected research papers (*documents*) in a personal *library*. A library is placed in the *library view* of Icecite (shown in figure 1, consider the caption below the figure for a detailed explanation of each UI-element). On adding a document to the library, its metadata and its bibliographic references are identified automatically due to the features (1)+(2). Because the metadata are used to describe the document uniquely, there is no need anymore to name files manually. The identified references of a document are placed next to the PDF file in the *document view* (shown in figure 2), that is accessible via clicking a document in the library view. A more detailed discussion about our approach of automatic identification of metadata and bibliographic references can be found in section 3.

Although Icecite is web-based, all library data (metadata, PDF files, etc.) are stored locally on client's filesystem. That's why some features of Icecite can be used even in case of offline mode, e.g. browsing the library, annotating

Library		[icecite]		Logged in as: Anton Chigurh	
Upload PDF		Delete Documents		Search your documents as well as DBLP ×	
Title	Author(s)	Venue	Year		
<input type="checkbox"/> ● Accurate Information Extraction from Research Papers using Conditional Random Fields	Fuchung Peng, Andrew McCallum	HLT-NAACL	2004		
<input type="checkbox"/> ● Broccoli: Semantic Full-Text Search at your Fingertips	Hannah Bast, Florian Baurle, Björn Buchhold, Elmar Haussmann	CoRR	2012		
<input type="checkbox"/> ● Cruising with a Battery-Powered Vehicle and Not Getting Stranded	Sabine Storandt, Stefan Funke	AAAI	2012		
<input type="checkbox"/> ● DocuBrowse: faceted searching, browsing, and recommendations in an enterprise context.	Andreas Girgensohn, Frank M. Shipman, Francine Chen, Lynn Wilcox	IUI	2010		
<input type="checkbox"/> ● Notification for shared annotation of digital documents.	A. J. Bernheim Brush, David Barger, Jonathan Grudin, Anoop Gupta	CHI	2002		
<input type="checkbox"/> ● PaperSketch: a paper-digital collaborative remote sketching tool	Nadir Weibel, Beat Signer, Moira C. Norrie, Hermann Hofstetter, Hans-Christoph Pott, Harald Reiser	IUI	2011		

Figure 1: A screenshot of the *library view* in Icecite. Each entry of library is listed with the following metadata fields: title, author(s), conference and year of publication. The checkbox besides each entry exists, to select the entry for deletion (the deletion is actually done by clicking the button *Delete Documents* in the upper left). The small bullet to the left of the title of each entry displays the status of entry (a *green* bullet means, that the full metadata were found for entry). On clicking the button *Upload PDF*, you can add a locally stored PDF file to library. The needed metadata will be computed automatically then. You can type in a search query into the search field (located to the right of the *Delete Document* button), to search in own library as well as in the digital library *DBLP*. On clicking an entry of search result, it will be added to library. In contrast, on clicking an entry of library, user is forwarded to the *document view* (cf. Figure 2).

PDF files or load PDF files into the library (certainly without automatic identification of metadata and references). It can be achieved by using features of the html5 standard, as described in section 6.3.

Additionally, the library data are synchronized periodically with the server to (1) access them from any location and (2) share them with other users for collaboration. The mechanism of data synchronization is described in section 6.3.

Searching for topic-related research papers.

In order to explore topic-related research papers and to extend the personal collection with them, generally the following 4 sources can be consulted:

(S1) Digital libraries, such as *DBLP* or *Medline*.

(S2) The web, such as publisher websites (like *ACM*, *IEEE*, *Springerlink*, etc.)

(S3) Bibliographies from already known research papers.

(S4) Personal recommendations.

The listed sources differ from each other in the amount of provided data. While the sources (S1) and (S2) may provide (links to) PDF files of research papers, the sources (S3) and (S4) usually don't (but only parts of metadata fields). Hence, in case of sources (S3) and (S4), extra effort is needed to retrieve the PDF files (e.g. by consulting the source (S1) or (S2) in addition). However, the invested time and effort may be in vain, if a file isn't fetchable, because it isn't available in general or the access to it is restricted and only available to licensees.

Accurate Information Extraction from Research Papers using Conditional Random Fields

Fuchun Peng
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
fuchun@cs.umass.edu

Andrew McCallum
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
mccallum@cs.umass.edu

Abstract

With the increasing use of research paper search engines, such as CiteSeer, for both literature search and hiring decisions, the accuracy of such systems is of paramount importance. This paper employs Conditional Random Fields (CRFs) for the task of extracting various common fields from the headers and citation of research papers. The basic theory of CRFs is becoming well-understood, but best-practices for applying them to real-world data requires additional exploration. This paper makes an empirical exploration of several factors, including variations on Gaussian, exponential and hyperbolic- L_1 priors for improved regularization, and several classes of features and Markov order. On a standard benchmark data set, we achieve new state-of-the-art performance, reducing error in average F1 by 36%, and word error rate by 78% in comparison with the previous best SVM results. Accuracy compares even more favorably against HMMs.

1 Introduction

Research paper search engines, such as CiteSeer (Lawrence et al., 1999) and Cora (McCallum et al., 2000), give researchers tremendous power and convenience in their research. They are also becoming increasingly used for recruiting and hiring decisions. Thus the information quality of such systems is of significant importance. This quality critically depends on an information extraction component that extracts meta-data, such as title, author, institution, etc., from paper headers and references, because these meta-data are further used in many component applications such as field-based search, author analysis, and citation analysis.

Metadata

Title: Accurate Information Extraction from Research Papers using Conditional Random Fields

Author(s): Fuchun Peng, Andrew McCallum

Venue: HLT-NAACL

Year: 2004

Title	Author(s)	Venue	Year
<input type="checkbox"/> <input checked="" type="radio"/> Accurate Information Extraction from Research Papers using Conditional Random Fields	Fuchun Peng, Andrew McCallum	HLT-NAACL	2004

Figure 2: A screenshot of the *document view* in Icecite. It shows both, the pdf file of research paper and the metadata of the paper and its references. Here you can read, annotate and comment the pdf file (you can turn the pdf into fullscreen-mode to hide the metadata to the right) and study the metadata of references. The entries of references list are of the same design as the entries in the *library view* (cf. figure 1). Again, on clicking an entry of references, you can add it to library, to read, annotate and share it and to study its references.

Due to the automatic identification of bibliographic references (2), the instant search in DBLP and Medline (3) and the automatic search for PDF files (5), Icecite speeds up the whole process of fetching topic-related research papers substantially. **[TODO: Eliminate this listing, and reuse it in infrastructure section]** Generally, there are three ways to add a new document to the library:

(ADD 1) Upload a PDF file to the library.

(ADD 2) Click an entry in the reference list of a selected document in the document view.

(ADD 3) Type a search query into the search field of the library view to browse for any entries in a digital library (DBLP or Medline). Click a record of the search result.

Variant (ADD 1) is followed directly by the automatic identification of the metadata and the bibliographic references of the research paper given by the uploaded PDF file.

Once all the metadata and references are retrieved, the document is added to the library.

In contrast, variants (ADD 2) and (ADD 3) are firstly followed by the automatic search for a PDF file, that contains the fulltext of the clicked record in the references list (resp. the search result). The PDF file is located in the web via the metadata, which were retrieved on the references identification (resp. via the metadata provided by the digital library). Once the PDF file was found, the further processing is similar to that for variant (ADD 1) as described above. See section 6.2 for a more detailed discussion of the search in digital libraries and the automatic search for PDF files.

Reading research papers.

While reading the PDF file of a research paper, it's quite common, to highlight some text passages and to make some notes to it. With Icecite, PDF files can be read online, if there is a proper browser plugin is installed to display the

PDF files in the browser and can be even annotated, if the plugin of Adobe Acrobat is used. On implementing this feature, we have paid attention to use *native*¹ annotations. This ensures, that the annotations are identified as such not only within Icecite, but also in various external PDF viewers. See the detailed description of this feature in section 6.4, why this intention isn't trivial to achieve.

Collaborate in groups.

On collaborating in (research-)groups to review some research papers together, many individual annotations are produced. Icecite is able to combine the individual annotations made to a PDF file to get a general conclusion. The data of library can be shared with other users nearly in real-time. So, once the set of documents and annotations in shared libraries was changed by a member of group, the changes are displayed to the other members instantly. See section 6.3 for a more detailed discussion of this feature.

In section 2, we will introduce some applications, that are related to Icecite. There are both, desktop-based and web-based applications, although existing desktop-based applications typically provide larger feature sets than existing web-based applications. We will compare the features of both types with those of Icecite. However, desktop-applications have some drawbacks...

[TODO: list drawbacks of desktop applications.]

[TODO: Present results of experiments].

[TODO: Present results of user study].

At the end of this introduction, we want to clarify, that the contribution of this paper is the overall design of Icecite and the basic ideas for each of the individual components of the system. Consider, that these components are complex problems each on their own and can not be discussed in all details in this paper. However, in section 8.1 we will give you an idea, how the various components can be optimized on future research.

2. RELATED WORK

We split the related work section mainly in three parts. In the first part, we introduce existing *desktop-based* and *web-based applications* similar to Icecite. In the second part, we focus on existing *extraction techniques* to retrieve informations about the metadata and/or the bibliographic references from research papers. Because we use a metadata knowledge base in background and match an extract to the referred record in the knowledge base to get full and correct metadata, we finally focus in the third part on *record matching techniques*.

2.1 Related Applications

A lot of applications were already developed, aimed to support researchers during a literature research process. In this section, we introduce some of them and compare their feature sets. Table 1 summarizes the individual feature sets in an overview.

2.1.1 Desktop-based Applications

The upper part of Table 1 breaks down the provided features of *desktop-based* applications. All of them can be seen

¹With native annotations we mean those annotations, which conform to the official PDF specifications

as *research managers* and are quite similar regarding to their basic structure: research paper are manageable in individual libraries, where they are listed with their (usually automatically identified) metadata. However further features differ from application to application. Because we can't describe all applications in detail here, we pick a single representative one, namely *Mendeley* [9]. We have chosen Mendeley, because it considers itself to be "the world's largest social reference management system" [24]. Moreover we believe that its feature set comes closest to the features of a state-of-the-art research manager. In the following, we will examine Mendeley's feature set in detail and compare it with those of the other applications.

Mendeley is basically a desktop-based application. Though Mendeley comes with a web-interface to browse and manage the library data in an ordinary web browser, which were synchronized with Mendeley's server. However the number of features of the web interface is restricted and that's why there is a '(✓)' in first column of Table 1 for Mendeley (a similar argument is valid for EverNote and Qiqqa). Further, Zotero comes with an extension for the Firefox browser.

In Mendeley, the metadata of research papers are extracted automatically from the PDF file. In contrast, the extraction of bibliographic references from PDF files isn't provided, neither in Mendeley² nor in other research managers. Consider, that ReadCube provides a feature called *enhancing PDF files*. For example, this feature enhance particular PDF files with prepared informations about bibliographic references. However, ReadCube doesn't extract these data directly from PDF files, but tries to resolve the DOI (digital object identifier) of a document and gathers the data from any web pages afterwards. Hence prepared references are only available, when the document contains such a DOI and when the data are retrievable from the web.

Further features of Mendeley are the built-in PDF viewer to read and to annotate PDF files within the application and the chance to tag documents with specific keywords. All data including the fulltexts of documents can be searched to find particular entries in the library. However external sources (like digital libraries) aren't browseable within Mendeley, as it is possible within Citavi, Qiqqa and ReadCube.

Furthermore, a bookmarklet (the so called *Web importer*) can be installed to the ordinary web browser to import the metadata of research papers from several external sources into the personal library, along with the corresponding PDF file, if it's provided by the chosen source. But the weaknesses of this feature are (1) the need to switch between Mendeley's interface and the browser and (2) the lack of an automatic search for PDF files in case that the external source doesn't provide PDF files. In contrast, ReadCube searches for PDF files on browsing the external sources to import them into the library with a single click. If a PDF could be *enhanced* (see above), this feature is also used to import a reference along with its PDF file into the library easily. However the approach to import documents by references is very simplistic, because the retrieved reference string can be just sent to an external source as query, in the hope that the correct

²Consider, that the extraction of references was supported in a previous version of Mendeley, but was removed again, "because it was consuming a fair amount of resources (on client and server side) without providing enough value" [TODO: How to cite the referred blog post?]

<i>Name of application</i>	<i>(CLOUD)</i>	<i>(OFFLINE)</i>	<i>(SHARED)</i>	<i>(EX-M)</i>	<i>(EX-R)</i>	<i>(AUTO-DL)</i>	<i>(ANNOT)</i>	<i>(SEARCH)</i>
Citavi [3]	-	✓	(✓)	(✓)	-	(✓)	-	(✓)
EndNote [6]	?	?	?	?	?	?	?	?
EverNote [8]	✓	✓	✓	-	-	-	-	(✓)
Mendeley [9]	✓	✓	✓	✓	-	-	✓	(✓)
Papers [10]	?	?	?	?	?	?	?	?
Qiqqa [11]	✓	✓	✓	✓	-	-	✓	✓
ReadCube [12]	-	✓	-	✓	(✓)	✓	✓	✓
Zotero [15]	✓	✓	✓	✓	-	-	-	(✓)
BibSonomy [2]	✓	-	✓	-	-	-	-	-
CiteULike [4]	✓	-	✓	-	-	-	(✓)	✓
EndNote Web [7]	✓	-	✓	-	-	-	-	(✓)
RefWorks [13]	✓	-	✓	-	-	-	-	(✓)
A.nnotate [1]	✓	-	✓	-	-	-	✓	(✓)
Crocodoc Personal [5]	✓	-	✓	-	-	-	✓	-
WebNotes [14]	✓	-	✓	-	-	-	✓	(✓)
Icecite	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Overview of the provided features by applications related to Icecite. If a feature is fully provided by an application, it’s labeled with an checkmark. If a feature is only partially provided, it’s labeled with an circle. The upper part of the table denotes desktop-based, the lower part web-badsed applications. Each acronym in the header of the table denotes an specific feature. The meanings of them are as follows: (*WEB*): application is web-based; (*EX-M*): extraction of metadata from PDF files; (*EX-R*): extraction of references from PDF file; (*PDF*): management of PDF files; (*ANN*): annotating PDF files; (*TAG*): tagging documents in the personal library; (*S-PL*): searching the personal library; (*S-FT*): searching the fulltexts; (*S-ES*): searching in external sources; (*IMP-ES*): Importing documents from external sources into personal library; (*IMP-R*): Importing referenced documents into the personal library; (*RES-PDF*): resolving PDF files for external documents; (*SY-M*): Synchronization of metadata; (*SY-P*): Synchronization of PDF files; (*SY-A*): Synchronization of annotations; (*SH*): Sharing documents of personal library with other users.

record will be found in the external source.

As mentioned above, Mendeley can synchronize the library data with its server to access them via the provided web interface or from several local installations of Mendeley on various devices. Moreover, the data synchronization offers the chance to share them with other members of individual groups to collaborate.

[TODO: Explain some special features? For example the feature of ReadCube: list all citing papers]

2.1.2 Web-based Applications

The lower part of Table 1 breaks down the provided features of *web-based* applications. Compared to desktop-based applications, their amount is usually much smaller. Most of the web-based applications focus only on specific subareas of literature research, like *pure metadata managing* or *PDF file annotating*.

Pure metadata managing is for example supported by *BibSonomy*, *CiteULike*, *EndNote Web* and *RefWorks* [13]. Basically, each application allows to collect bibliographic metadata systematically in a personal library and to generate bibliographies for own publications from them. A record can be either created manually or imported from external sources. All metadata records are generally fully searchable. Furthermore, libraries are typically shareable with other users to get a combined set of metadata. Two of the listed applications (namely *BibSonomy* and *CiteULike*) allow to attach

PDF files to each record, whereas the automatic extraction of metadata and references is generally not supported. Further, annotating PDF files is possible seldomly because only *CiteULike* provides this feature in a premium version for paying users. Though, annotating PDF files in the browser is the aim of the following type of web-based applications.

Annotating PDF file in the browser is for example supported by *A.nnotate*, *Crocodoc* or *WebNotes*. All applications allow to read, annotate and comment PDF files collaboratively. The features are usable via an ordinary web browser due to techniques of HTML5 or Flash. Basically, the scopes of supported files are not limited to PDF files, but also include arbitrary text files, images and even websites. Hence, such applications are not primarily intended to support a literature research process. That’s why features like *automatic metadata extraction from PDF files* or *automatic search for fulltexts* are usually missing in annotation applications

2.1.3 Comparison of web-based and desktop-based applications

[TODO: Wrote something about our aim, e.g. providing same scope of features, that are currently provided by desktop-based applications].

2.2 Metadata Extraction Techniques

In this section, we focus on related techniques to extract the metadata fields (title, author(s), affiliation(s), journal, etc.) of research papers and their bibliographic references. At first we will introduce some machine learning techniques, namely *Hidden Markov Models*, *Support Vector Machines* and *Conditional Random Fields* and analyze their extraction accuracy. Afterwards, we will do the same with rule-based techniques

2.2.1 Hidden Markov Models (HMMs)

Hidden Markov Models have been applied to identify the metadata fields of both, research papers and their references. Seymore et al [28] describe an approach, where a manually-constructed model are used to identify the metadata fields in the headers of research papers. The constructed model contains multiple states per field. They achieve an extraction accuracy of 92.9% over all metadata fields.

Borkar et al [18] use a nested HMM to identify the metadata fields of references. The constructed HMM contains one state per metadata field. Each state itself consists of another HMM, representing the internal structure of the field. With this method, Borkar et al achieve an average precision and recall ratio of 0.87 over all fields.

2.2.2 Support Vector Machines (SVMs)

Support Vector Machines are used by Han et al [22] to identify the metadata fields in the headers of research papers. Their experiments based on 500 training headers and 435 test headers resulted in an overall accuracy of 92.9%.

The metadata extraction of Mendeley relies on a two-stage SVM and is aimed to extract the title and the authors from research papers (Granitzer et al, [20]). In a first step, it classifies each line of the header text into title, author and other classes using text and formatting features. In a second step, the classification is improved by using contextual information such as the predicted class labels of the neighboring lines. The experiments of Granitzer et al. based on a training set of 1000 research papers. They resulted in an average precision rate of 0.81 and an average recall rate of 0.62 for the author extraction and in an average precision rate of 0.94 and an average recall rate of 0.91 for the title extraction.

2.2.3 Conditional Random Fields (CRFs)

Peng et al. [27] utilizes Conditional Random Fields to extract the metadata from both, the headers and the references of research papers. Experiments to test the field extraction from headers were performed on 500 training headers and 435 test headers and resulted in an overall accuracy of 73.3%. Experiments to test the field extraction from references were performed on the *Cora* dataset, containing 500 references. They resulted in an overall extraction accuracy of 77.3%. **[TODO: The claimed numbers are instance accuracies: the percentage of instances in which every word is correctly labeled. Word accuracies is quite higher]**

Machine learning techniques are generally considered to be robust and adaptable [22]. Moreover, the above announced extraction accuracies show, that utilizing machine learning techniques is a promising way to extract metadata from research papers and their references.

However as claimed by Guo and Jin [21], generating accurate labeled datasets needed to train the models is time-

consuming and costly. Alternatives to machine-learning approaches are rule-based approaches, which are usually faster, but less accurate.

2.2.4 Rule-based approaches

Rule-based approaches base on a set of rules, describing how to identify the metadata to extract in research papers. The rules are deduced from human observations regarding the basic structures of research papers.

Jöran Beel et al. [17] have identified the titles of 693 research papers by using a rule-based approach, i.e. by utilizing their layout informations, i.e. the font sizes and their positions within the documents. Their experiments yield to 77,9% correctly identified titles.

Cortez et al [30] propose a unsupervised knowledge-based approach to recognize the components of reference strings given in any format. The authors split each reference string into blocks and label each block with the name of a metadata field. Therefore a knowledge is used, that was constructed automatically from an existing set of sample metadata records. Experiments were performed on 300 reference strings and have produced an overall extraction accuracy of 82%. **[TODO: Verify this accuracy].**

All approaches seen so far try to extract any metadata fields of research papers and/or their references without utilizing any metadata knowledge bases to guide the extraction process. Guo and Jin introduce this approach in [21]. They extract metadata using a rule-based approach and check if there is any matched document in a knowledge base, which is mainly developed from DBLP. If matched, they use the metadata stored in the knowledge base to make sure the extracted reference metadata are correct. The experiments on 97 research papers and 2157 references to be extracted resulted in an average extraction accuracy of 89,1% over all metadata fields.

Basically, our approach follows up the approach of Guo and Jin. In a first step, the title and each reference-string is extracted from a research paper, profiting from the speed advantage of a simple rule-based approach compared to a machine learning approach. In a second step, each extract is matched to a record of DBLP or Medline, to get full and correct metadata. That's why we inspect existing record matching techniques in the following.

2.3 Record Matching Techniques

As described above, we match the extracted data to a record of a metadata knowledge base to get full and correct metadata. In this section, we inspect the most common record matching techniques.

For our purposes, record matching can be defined in the following way: Given an extract (i.e. the title or a reference of a research paper), that refer to a record in the metadata knowledge base. Identify this record and associate it with the given extract. The task may be affected by noisy factors (like extraction failures, different spellings, abbreviations, etc.), whereby the typographic design of a metadata field may differ from the specifications in the referred record.

Most of the existing record matching techniques base on string comparison approaches, which generally determines the similarity of two strings on the base of character-based, token-based or phonetic-based similarity measures. We only focus on character-based measures here. For a detailed discussion of all the measures, see the surveys in [19] and [23].

Character-based similarity measures analyzes the similarity of two given strings s and t by comparing the strings character by character. For example, the *Levenshtein distance* [25] between s and t is defined as the minimal number of needed edit operations (*add/replace/delete* a character) to transform s into t . The Levenshtein distance is quite reasonable to match an extracted title to its referred record, because the expected similarity between the extracted title and the record's title is quite high. Apart from that, the Levenshtein distance isn't reasonable to match an extracted reference to its referred entity, because of the multiple metadata fields and their unknown order in the reference string. Hence, the expected similarity between an arbitrary combination of the metadata fields of a record and the extracted reference is usually low and thus mostly insignificant.

Another example of a character-based measure is the *Smith-Waterman distance* [29], an extension of the Levenshtein distance, which was introduced to find the best matching substrings between s and t . For example, the Smith-Waterman distance is quite reasonable to decide, if specific metadata fields (e.g. the title) of a record is included in a given reference string. So, the referred record may be identified by the number of the record's fields included in the given reference string.

Because the complexity of both algorithm is $O(|s| \cdot |t|)$, brute-force pairwise comparisons over all records to identify the referred record by a given extract are usually not feasible. Instead it's quite common to segment the set of records into *blocks* in a pre-processing step [23]. This segmentation usually relies on some weak and non-costly criteria. The costly string similarity measures are then only executed on single blocks of entity candidates.

However choosing a reasonable blocking strategy, i.e. selecting attributes for blocking is a challenging task. Michelson and Knoblock suggest a machine learning approach, that learns blocking schemes automatically, see [26].

3. METADATA IDENTIFICATION

In this section, we describe our approach to identify the metadata of research papers in detail. For a discussion of the bibliographic references identification, see section 4. As mentioned above, we use a simple rule-based algorithm to extract the title from the PDF file of a research paper. After the extraction, the title is matched to a record of the metadata knowledge base (which is given by DBLP and Medline) to get complete and correct metadata. The algorithm to extract the titles from research papers is described in section 3.1, whereas the matching process is described in 3.2.

3.1 The extraction of titles from PDF files

Basically, we use the open source Java library *PDFBox*³ to extract data from PDF files. With *PDFBox*, we get basically the location, the width, the height and the font) of each character. Let these attributes describe the *type* of an character. With these types, the types of all words and all text lines are reconstructed. Furthermore, all text lines are segmented into several regions such that each region contains all consecutive lines with the same type. Figure 3 shows an example how such a division might look like. It results a natural text hierarchy with the following layers:

$characters \rightarrow words \rightarrow textlines \rightarrow regions$

³<http://pdfbox.apache.org/>

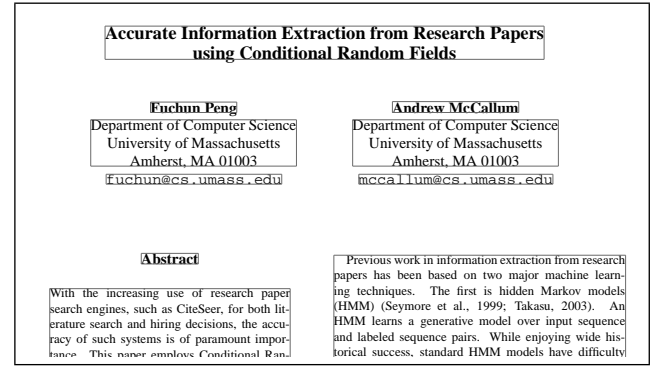


Figure 3: Example for a division of a PDF file into regions.

Each layer offers its own type, whereas the type of a region follows by the types of the contained text lines; the type of each text line follows by the type of each contained word, and so on. Based on the types, we define a natural order on the regions, which is given by their font sizes followed by their font styles. On sorting by font styles, the following order is applied:

bold > *italic* > normal.

Finally, all types with the same font sizes and the same font styles are ordered by their locations.

On identifying a title, we assume that all lines belonging to the title are placed immediately after each other on the front page. Further we assume, that these lines possess the same type and that this type is the *largest* one (according to their defined order) compared to the types of the remaining text lines. It follows from the assumptions, that there is exactly one region placed on the front page, that contains the title exclusively. It possesses the largest type and so, it can be identified by sorting the regions by their types.

3.2 The matching of extracted titles

Once the title of a research paper is extracted, it is matched to a record of the metadata knowledge base (KB) to get full and correct metadata. The basics of the matching process is quite simple and base on computing the Levenshtein distances between the extracted title and the titles of KB records. However, as mentioned in section 2.3, brute-force pairwise comparisons over all KB records are not feasible. That's why the matching process is preceded by a weak filtering mechanism to get a small-sized set of candidates C from the KB . The costly computations of the Levenshtein distances are then only executed on the set of candidates.

The filtering mechanism base on an ordinary *inverted index I*. The index is filled with all words, which results from normalizing the title, author names and years of each record in the KB (author names and years are included for the references matching process, which is described in section 4.2). Normalizing a string means:

- (1) Split the string into words,
- (2) Transform each character into lowercase,
- (3) Remove all punctuations, special characters and stop words.

With the inverted index, each word w is mapped to an inverted list I_w containing all the KB records, whose metadata contain the word.

The set of candidates are queried from the index in a typical manner: First, the extracted title is also normalized (as described above) and for each resulting word the associated inverted list of records are fetched. In a second step, all the lists are merged into a single list L . Its elements are sorted by their occurrences in the fetched lists (and hence by the number of shared words between the record and the extracted title). C is then given by the top- k elements of L , where $k > 0$ is an arbitrary, but fixed integer value. The subsequent computations of the Levenshtein distances are restricted to C and finally, the extracted title is matched to the record, for which the Levenshtein distance between its title and the extracted title is minimal.

4. REFERENCES IDENTIFICATION

In this section, we describe our approach to identify the bibliographic references of research papers in detail. Again, a rule-based algorithm is used to extract the references from PDF files and each extract is matched to a record of the metadata knowledge base. The algorithm to extract the references is described in section 3.1 and the matching process is described in 3.2.

4.1 The extraction of references from PDF files

Compared to the extraction of the title, the extraction of references is much more challenging. That's because there isn't a universal convention on designing a reference or on arranging its metadata fields. So we have to consider various styles for the bibliography entries.

The text lines of a PDF file are extracted on the same way as described for the title extraction. To identify the position of the bibliography in the research paper, all the lines are checked for the bibliography header, e.g. by analyzing their types and by searching for the words *References*, *Literature*, *Bibliography*, etc. On the references identification, only text lines are considered, which follows the bibliography header and which are members of the so called *content box*. The *content box* of a page contains all the *relevant* text lines in a page, that means all text lines except the page headers and the page footers.

These text lines are traversed sequentially. On visiting a text line, also the previous and the next line are examined to mark the with one of the following labels:

(R_H) line is a reference header (the first line of a reference)

(R_B) line is a member of the reference body (all further lines of a reference)

(E) line is the last line in the bibliography.

Consider, that each reference consists of exactly one line, which is marked with (R_H) and of an arbitrary number of lines, which are marked with (R_B) . The extraction process is finished, if a line was marked with (E) . Hence, to extract the references, all text lines have to be marked correctly.

Depending on the layout of a bibliography and its references, marking a line with the correct label may be a non-trivial task. However identifying the end of a bibliography is comparatively simple: a line is marked with (E) , if the type of the next line is larger than the type of the current line or if the line pitch between the current line and the next line is

"too" large. But especially the distinction between reference headers and reference bodies is challenging. The basic idea of our approach is to identify characteristic attributes, on which a label of a line can be reasoned. Examples for such attributes are discussed in the following.

The existence of reference anchors.

[TODO: Assume, that the attributes are consistent]
[TODO: Example] *Reference Anchors* are unique and usually short identifiers, which are prepended to reference headers, like [1], [Miller2012], etc. If the reference headers of a bibliography contains reference anchors, marking the text lines is simple: If a line is prepended with an anchor, it is labeled with (R_H) and otherwise with (R_B) .

The indentation of reference bodies.

[TODO: Example] **[TODO: Pseudocode?]** Another characteristic attribute is the indentation of the reference bodies compared to the reference headers. In this case, marking the lines is a bit more tricky: Initially, the first line (after the header) of the bibliography is labeled with (R_H) . All further lines are marked as follows: If a line is intended to either the previous or the next line, it is marked as (R_B) . If the previous or the next line is intended to the current line, it is marked as (R_H) . Otherwise, it is marked with the label of the previous line.

Special care is needed, if the text lines are arranged in multiple columns and the column of the previous or the next line differs from the column of the current line. Therefore, it must be taken into account on checking for indentations, that one of the lines is shifted horizontally by a specific amount.

The line pitches between references.

If no attributes introduced so far are used in a bibliography, marking the lines is most difficult. Another but weak characteristic attribute are the line pitches between each individual reference, which are usually larger than the line pitches between two common text lines. Again, the first line of the bibliography is marked with (R_H) in this case. Further lines are also marked with (R_H) if its pitch to the previous line is larger than the common one. Consequently, a line is marked with (R_B) if its pitch correlates with the common line pitch. Obviously, marking a line with (R_H) or (R_B) is challenging if the previous line is located in another column or another page.

4.2 The matching of extracted references

The process of matching an extracted reference string is quite similar to that of matching an extracted title. Thus, a set of candidates are determined for an extracted reference in the same way. However the processes differ in the evaluation of these candidates. Because a reference doesn't contain only the title, but also further metadata of the belonging research paper (e.g. the name of authors, the year of publication, the name of conference, etc.) in unknown order, a different scoring scheme on evaluating the candidates is needed.

We can't discuss the applied scoring scheme in all details here. In principle, it consists of three different similarity scores resulting from the following alignments:

(A1) Alignment of the candidates' title against the extracted reference string.

(A2) Alignment of the candidates’ author names against the extracted reference string.

(A3) Alignment of the candidates’ year against the extracted reference string.

For example, (A1) bases on the Smith-Waterman distance, introduced in section 2.3, to find the longest common substring between the extracted reference and the title of a candidate. Finally, all scores are combined into a single score and the extracted reference is matched to the record, for which this score is maximal.

5. EXPERIMENTS

In this section, we present the result of our experiments, which we have executed to test the quality and the performance of our extraction and matching algorithms. [TODO: Tests for automatic download for reference/search entry?]

5.1 Input data

Our test collection consists of 1200 randomly selected research papers (available as PDF files), whose metadata are stored in *DBLP* (700) or *Medline* (500).

DBLP is a digital library, providing various metadata of more than *2.1 million* computer science research papers.

Medline is a digital library, providing various metadata of more than *21 million* research papers, mainly focused on topics of medicine.

Based on this test collection, we have created various ground truths manually, storing different data for each research paper: [TODO: Emphasize, that both libraries are of different size]

(GT1) The title and the unique key of matching record in DBLP.

(GT2) The title and the unique key of matching record in Medline.

(GT3) Each reference and the unique key of accordant record in DBLP for each reference [TODO: no match]

(GT4) Each reference and the unique key of accordant record in Medline for each reference [TODO: no match]

On evaluating the quality of our algorithms, we have run them on the collected PDF files in various experiments. At the same time, we have measured the execution times for computing the results to evaluate the performance. The individual results of the experiments are discussed in the sections 5.3 - 5.5.

5.2 Computing environment

The code for the extraction of the title and the references from PDF files is written in Java. Apart from that, the code for the matching of extracts to records of DBLP is written in C++.

All the tests were run on a single machine with 4 Intel Xeon 2.8 GHz processors and 35GB of main memory, running Ubuntu 9.10 64-bit.

5.3 Extraction times and quality

[TODO: tables in which detail? Alternative:]

Runtime Evaluation	DBLP	Medline
Search/download PDF	?	?
Title Extraction	43.46ms	42.65ms
Title Matching	4.93ms	26.31ms
References Extraction	141.85ms	170.75ms
References Matching	(43.2ms)	(269.6ms)
Total time per PDF	233.44 ms	509.31ms

Table 2: Overview of average runtime to compute the whole metadata for a PDF of the DBLP- respectively the Medline test collection. [TODO: runtime for references extraction & -matching: overall or per reference?; times are "netto", e.g. the time for connecting to matching server isn't considered yet.]

Quality Evaluation	DBLP	Medline
Search/download PDF	?	?
Title Extraction	94.7%	89.8%
Title Matching	98.1%	84.2%
References Extraction	81.6%	91.1%
References Matching	90%	83.1%
Prob. of correct metadata	68.2%	57.2%

Table 3: Overview of percentages by correct results in each subtask. [TODO: false-positives (extracted string is not a reference) aren't considered]

5.3.1 Title extraction

On comparing the extracted titles with the expected title, we have distinguished the results into various categories:

(TE1) The extracted title and the expected title are equal.

(TE2) The extracted title contains the expected title only partially.

(TE3) The extracted title contains the expected title amongst other words.

(TE4) The extracted title and the expected title have no words in common.

[TODO: Define TE1-TE4 in more precision]

	#	max.	TE1	TE2	TE3	TE4
DBLP	700	98.8%	94.7%	0.4%	0%	3.7%
Medline	500	98.8%	89.8%	3.4%	0.2%	5.4%

Table 4: Breakdown of quality results by category for title extraction.

Table 3 provides the percentages of the quality results in each of these categories. Due to encoding issues, the extraction of titles was impossible for a few PDF files. The third column in table 3 shows the maximum obtainable percentage.

Table 5 provides the average runtimes for both test collections (2nd column) as well as a breakdown of them into the individual tasks to solve to extract a title from a PDF file:

(XX1) Load the PDF file into memory.

(XX2) Extract the characters from the PDF file.

	time/title	XX1	XX2	XX3	XX4
DBLP	43.46ms	34.7%	51.2%	13.9%	0.2%
Medline	42.65ms	33.6%	52.3%	13.9%	0.2%

Table 5: The average runtimes to extract a title from a PDF file of DBLP and of Medline. Columns 3-6 provide a breakdown of runtimes in tasks to solve.

(XX3) Assemble the characters to text lines.

(XX4) Identify the title amongst the text lines.

The percentages of the runtimes for each task are shown in the columns 3-6.

5.3.2 References extraction

Apart from the title extraction, we didn’t distinguish the results of comparing the extracted references with the expected references into various categories. Instead, we define only this single category [TODO: Why?]:

(YY1) The extracted and the expected reference are equal [TODO: bis auf eine geringe edit-Distanz].

	# files	# refs.	max.	YY1
DBLP	700	9753	98.5%	81.6%
Medline	329	10135	99.2%	91.1%

Table 6: Breakdown of quality results by category for reference extraction.

Table 6 provides the percentage of the category YY1 in 5th column. Furthermore, the 3rd column provides the total number of references, contained in the research papers of the particular test collections. As for the title extraction, a few references weren’t extractable due to encoding issues. The maximum percentage of extractable references are shown in the 4th column.

	time/PDF	XX1	XX2	XX3	XX4
DBLP	141.85ms	10.5%	65%	20.6%	3.9%
Medline	170.75ms	7.9%	64.2%	24.1%	3.8%

Table 7: The average runtimes to extract the references from a PDF file. Columns 3-6 provide a breakdown of runtimes by sub-tasks.

The average runtimes to extract the references from research papers of the particular test collection are shown in table 7.

5.4 Matching times and quality

Table 8 provides the evaluation results of both, title matching and references matching. The percentages of correct title matchings are shown in 3rd column and of correct references matchings in the 5th column.

Tables 9 and 10 provide the average runtimes to match a title, respectively a reference to a record of the particular digital library. Furthermore, it provides a breakdown of runtimes into the following tasks to solve on extraction:

(ZZ1) Find candidates.

(ZZ2) Score the top- k candidates [TODO: explain the k].

(ZZ3) Identify the best matching candidate.

	# title	TM1	# refs	RM1
DBLP	700	98.1%	500	90%
Medline	500	84.2%	496	83.1%

Table 8: The percentages of correct title and references matchings.

The percentages of the runtimes for each task are shown in the columns 3-5.

	time/title	(ZZ1)	(ZZ2)	(ZZ3)
DBLP	4.93ms	59.2%	40.8%	<0.1%
Medline	26.31ms	85.3%	14.7%	<0.1%

Table 9: The average runtimes to match a title to a record of DBLP (respectively Medline). Columns 3-5 provide a breakdown of runtimes in tasks to solve.

	time/reference	(ZZ1)	(ZZ2)	(ZZ3)
DBLP	4.32ms	85.4%	14.4%	0.2%
Medline	26.96ms	90.8%	9.2%	0.1%

Table 10: The average runtimes to match a reference to a record of DBLP (respectively Medline). Columns 3-5 provide a breakdown of runtimes in tasks to solve.

5.5 Automatic PDF search times and quality

[TODO: Something to test?]

6. SYSTEM INFRASTRUCTURE

In the sections 3 and 4 we have already introduced two of the main components of Icecite, namely the automatic identification of metadata and references from research papers. In this section, we will give you a general overview of the implementation details of all further components. Moreover, we will discuss, how the components act together and how they are embedded into the typical workflows of Icecite.

6.1 The User Interface

The user interface of Icecite is fully web-based and was developed with GWT (*Google Web Toolkit*). It is mainly splitted into the library view (shown in figure 1) and into the document view (shown in figure 2). In the library view, each document of the personal library is displayed with its full metadata. In the document view, the PDF file is placed next to the identified references of a selected document. To add a document to the library, there are generally three ways:

(ADD 1) Upload a PDF file to the library.

(ADD 2) Click an entry in a reference list in the document view.

(ADD 3) Type a search query into the search field of the library view to browse for any entries in a digital library (DBLP or Medline). Click a record of the search result.

Variant (ADD 1) is followed directly by the automatic identification of the metadata and the bibliographic references of the research paper given by the uploaded PDF file. Once all the metadata and references are retrieved, the document is added to the library.

In contrast, variants (*ADD 2*) and (*ADD 3*) are firstly followed by the automatic search for a PDF file, that contains the fulltext of the clicked record in the references list (resp. the search result). Once the PDF file was found, the further processing is similar to that for variant (*ADD 1*) as described above. The automatic search for PDF files is discussed in detail in section 6.2.

In the document view, PDF files can be read if there is a proper browser plugin is installed to display the PDF files in the browser. It can be even annotated if there is the plugin of Adobe Acrobat is installed. On implementing this feature, attention was paid to use *native*⁴ annotations. This ensures, that the annotations are identified as such not only within Icecite, but also in various external PDF viewers. See the detailed description of this feature in section 6.4, why this intention isn't trivial to achieve.

All library data (metadata, PDF files, etc.) are stored locally on client's filesystem via the HTML5 *FileSystem API*, such that the basic features of Icecite can be even used offline. To keep the data consistent with the server and to share the data with other users, the data are synchronized periodically with the server. See section 6.3 for a detailed discussion of the data synchronization.

6.2 The search functionality

Obviously, a PDF file is needed to identify the metadata and the bibliographic references for a research paper. However, there are no PDF files given on adding documents to the library via the methods (*ADD 2*) or (*ADD 3*). That's why Icecite is able to find PDF files automatically in web. In this section, we describe both, the basic search functionality to browse the own library and external digital libraries and the functionality to search for PDF files automatically.

The core of Icecite's search functionality is built with *CompleteSearch*, an interactive and efficient search engine, implemented by Bast and Weber [16]. On typing a search query in search field of the library view, all records in DBLP and Medline are searched on the one hand. On the other hand, also the metadata, the fulltext and the annotations (i.e. comments and tags, see section 6.3) of each document in the personal library are searched. Hence, there are two instances of *CompleteSearch*, each with different underlying indexes: one containing the data of DBLP and Medline (I_{DM}) and one containing the data of personal library (I_L). Consider, that the data of library are quite volatile and that modifications on it are very common. So to keep the index I_L up to date, the indexing is triggered automatically on every modification in library. Apart from that, the data in DBLP and Medline are more settled, so that the update of index I_{DM} is necessary less frequently and is triggered manually from time to time.

The search results from both indexes are combined and displayed in library view, where the resulting entries of I_L are listed before the entries of I_{DM} . The entries of I_L are enriched with fulltext excerpts to highlight the matching parts in fulltext. Clicking an entry of I_L triggers a forwarding to the *document view* and clicking an entry of I_{DM} induces the adding of the entry to the library and hence the automatic search for an accordant PDF file.

The process of the searching a PDF file automatically is multistage. At first stage, Icecite follows up an url, that is

⁴With native annotations we mean those annotations, which conform to the official PDF specifications

provided by DBLP [TODO: Add Medline]. Ideally, this url (called "ee" in DBLP, ee = electronic edition), refers to the location of the PDF file. In this case, the searched PDF file can be downloaded from the ee. However, the url usually doesn't refer to the PDF file directly, but to any document-specific page of the publisher, where the download of the PDF file is usually restricted by the publisher and only available for licensees.

Icecite checks the availability of the PDF file and downloads it, if applicable. If there is no such PDF file or access to it is restricted, Icecite switches to the second stage, where it searches for the file by querying Google with the title and the author names of the record. [TODO: Is it appropriated to mention this here? (because its not legal)] Nevertheless, Icecite isn't always able to identify the correct PDF file among Google's search result definitely. Thus, a set of file candidates are shown to the user, where he can choose the correct one.

6.3 Synchronizing and sharing data

In Icecite, all data of the personal library (metadata of documents, PDF files, annotations of PDF files, etc.) are stored locally on the client utilizing the HTML5 *FileSystem API*. This API allows a web application to read and write data to a sandboxed section of the local file system. Once the library data are requested from the server for the first time, they will be stored locally where they can be browsed without requesting them from the server again and again. In combination with another HTML5 feature called *ApplicationCache*, some features of Icecite are even accessible in offline scenarios, for example: Browsing the library, uploading PDF files into the library, reading and annotating PDF files in browser.

To keep the locally stored data consistent with the server, the data are synchronized periodically. Due to the synchronization, documents of the personal library are shareable with other users. A major benefit is to be able to see all annotations made by various users in a shared PDF file nearly in real time. The background of the synchronization and sharing process is built by a pure *Subversion* system. It is utilized to manage the individual data sets of users on the server and to combine the individual data sets of shared libraries. Due to Subversion, all changes on the datasets are fully reproducible and any conflicts (e.g. on annotations) are generically resolvable.

6.4 Annotating PDF files in browser

During the implementation of a feature to be able to annotate PDF files in browser, we paid special attention to utilize *native* annotations in PDF files. Hence, annotations are identified as such not only within Icecite, but also in the most conventional external PDF viewers. Moreover, annotations aren't embedded immutably into PDF files in this way – instead, they are modifiable even in external PDF viewers. [TODO: Mention here, that other applications don't use native annotations].

All annotations data are stored separately from the PDF files, such that Icecite is able to synchronize the annotations independently from the PDF files. So, the annotations must not always be transferred to server on synchronizing changes on annotations. Because the annotations aren't stored within a PDF file, they are injected instantly when opening the file.

The mentioned characteristics were implemented against the widespread Adobe Acrobat Standard plugin. This plugin are aimed to display PDF files in a web browser. However, PDF files cannot be annotated in a browser out of the box, even if it's feasible in the desktop application of Adobe Acrobat. In the following we present a method, how PDF files can be annotated in the browser nonetheless.

Generally, a PDF file can include several pieces of javascript code, that is triggered on certain events, for example on opening the PDF file. Accordingly, a small javascript snippet is injected into a PDF file when adding it to the library. This snippet unlocks the standard annotation tools of the Adobe Acrobat plugin which are locked per default **[TODO: why?]**. Furthermore, it establishes a message handler, that allows a two-way communication between a PDF file (P) and the surrounding web application (A). This message handler is used in the direction $A \rightarrow P$ (1) to send the annotations to inject on opening a PDF file and (2) send the annotations to add/edit/delete after a synchronization process. In contrast, the direction $P \rightarrow A$ is used to inform the application about user actions regarding the annotations (e.g. add, edit, delete an annotation).

7. USER STUDY

#pages: 1

8. CONCLUSION

#pages: 1 (combined with references)

8.1 Future Work

- eliminate limitation to Google chrome.
- eliminate limitation to Adobe Professional.

9. REFERENCES

- [1] A.nnotate.com: Upload, Annotate, Share. Online document review and collaboration - PDF, Word and HTML. <http://www.a.nnotate.com/>. Jan. 2013.
- [2] BibSonomy: The blue social bookmark and publication sharing system. <http://www.bibsonomy.org/>. Jan. 2013.
- [3] Citavi: Reference Management and Knowledge Organization. <http://www.citavi.com/>. Jan. 2013.
- [4] CiteULike. <http://www.citeulike.org/>. Jan. 2013.
- [5] Crocodoc Personal: View & Comment on Any Document. <http://personal.crocodoc.com/>. Jan. 2013.
- [6] EndNote. <http://www.endnote.com/>. Jan. 2013.
- [7] EndNote Web. <http://www.myendnoteweb.com/>. Jan. 2013.
- [8] Evernote: Remember everything with Evernote. <http://www.evernote.com/>. Jan. 2013.
- [9] Mendeley: Free reference manager and PDF organizer. <http://www.mendeley.com/>. Jan. 2013.
- [10] Papers: Your Personal Library of Research. <http://www.mekentosj.com/papers/>. Jan. 2013.
- [11] Qiqqa: Free reference manager and research manager. <http://www.qiqqa.com/>. Jan. 2013.
- [12] ReadCube: Free Reference Manager - Academic Software For Research. <http://www.readcube.com/>. Jan. 2013.
- [13] RefWorks: Your online research management, writing and collaboration tool. <http://www.refworks.com/>. Jan. 2013.
- [14] WebNotes. <http://www.webnotes.net/>. Jan. 2013.
- [15] Zotero. <http://www.zotero.org/>. Jan. 2013.
- [16] H. Bast and I. Weber. The CompleteSearch Engine: Interactive, Efficient, and Towards IR& DB Integration. In *CIDR*, pages 88–95, 2007.
- [17] J. Beel, B. Gipp, A. Shaker, and N. Friedrich. SciPlore Xtract: Extracting Titles from Scientific PDF Documents by Analyzing Style Information (Font Size). In *ECDL*, pages 413–416, 2010.
- [18] V. R. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *SIGMOD Conference*, pages 175–186, 2001.
- [19] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.
- [20] M. Granitzer, M. Hristakeva, K. Jack, and R. Knight. A comparison of metadata extraction techniques for crowdsourced bibliographic metadata management. In *SAC*, pages 962–964, 2012.
- [21] Z. Guo and H. Jin. Reference metadata extraction from scientific papers. In *PDCAT*, pages 45–49, 2011.
- [22] H. Han, C. L. Giles, E. Manavoglu, H. Zha, Z. Zhang, and E. A. Fox. Automatic Document Metadata Extraction Using Support Vector Machines. In *JCDL*, pages 37–48, 2003.
- [23] M.-Y. Kan and Y. F. Tan. Record matching in digital library metadata. *Commun. ACM*, 51(2):91–94, 2008.
- [24] P. Kraker, C. Körner, K. Jack, and M. Granitzer. Harnessing user library statistics for research evaluation and knowledge domain visualization. In *WWW (Companion Volume)*, pages 1017–1024, 2012.
- [25] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.
- [26] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*, pages 440–445, 2006.
- [27] F. Peng and A. McCallum. Accurate information extraction from research papers using conditional random fields. In *HLT-NAACL*, pages 329–336, 2004.
- [28] K. Seymore, A. McCallum, and R. Rosenfeld. Learning Hidden Markov Model Structure for Information Extraction. In *AAAI 99 Workshop on Machine Learning for Information Extraction*, pages 37–42, 1999.
- [29] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
- [30] E. C. C. Vilarinho, A. S. da Silva, M. A. Gonçalves, F. de Sá Mesquita, and E. S. de Moura. Flux-cim: flexible unsupervised extraction of citation metadata. In *JCDL*, pages 215–224, 2007.