High-Pass Text Filtering for Citation Matching

Yannis Foufoulas[1(✉)], Lefteris Stamatogiannakis[1], Harry Dimitropoulos[1], and Yannis Ioannidis[1,2]

[1] Department of Informatics and Telecom, University of Athens, Athens, Greece
{johnfouf,estama,harryd,yannis}@di.uoa.gr
[2] "Athena" Research Center, Maroussi, Greece

Abstract. Open publications are increasing at such a rapid pace that it is almost impossible for researchers to keep up with them. Even in terms of computational complexity, the data are becoming bigger and bigger, so there is a great need to provide new and faster algorithms for mining scientific articles. One such important mining task is finding citation links between the literature, which can assist researchers looking into the literature, finding dependencies between publications, and so on. In this paper, we introduce a greedy citation matching algorithm, that works with plain unstructured text and mines citations from papers regardless of the format in which the citations are presented. This research is supported by the European Commission under projects OpenAIRE2020 (643410) and Human Brain Project (720270).

Keywords: Citation matching · High-pass filtering · Reference extraction

1 Introduction

Scholarly communication is currently at a new phase where researcher's published results are more optimally shared, discovered, validated and re-used when they are exposed in their full context. This means that they are best accompanied by all the relative information that provides an insight and capacity to translate the research process and activities that have taken place. Such information may be citations. If we are able to provide links between the literature, then this can be used for multiple purposes including literature search, finding similar publications, analysis of research trends, etc.

Most of the time, citation extraction and parsing is not enough; there is also a need to match the citations to metadata databases in order to enrich them with more useful information, such as the complete author list, scientific areas, journal information and in some cases abstracts or even fulltexts. Such info is not always included in the citation text, while abstracts and fulltexts are never included. Even when some of this exists, there is no algorithm that assures 100% recall rate in the parsing and extraction phase.

Due to these reasons, the matching of the citations with a metadata database is an important task. Currently, a user is able to download publications' metadata

© Springer International Publishing AG 2017
J. Kamps et al. (Eds.): TPDL 2017, LNCS 10450, pp. 355–366, 2017.
DOI: 10.1007/978-3-319-67008-9_28

from various sources including repositories and other systems which offer APIs (e.g. PubMed API[1], ArXiv API[2], CrossRef Search API[3], OpenAIRE[4] API)

Figure 1 presents the most common workaround to extract citations links. The first step is citation extraction and parsing. Citation extraction regards the extraction of a citation and its metadata (title, author names, journals, dates).

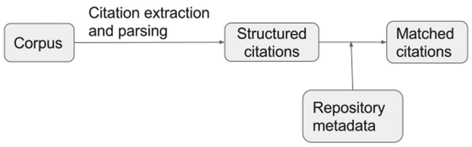


Fig. 1. Citation extraction, parsing and matching workflow

Titles, author names and other metadata are typed in many different ways and orders. An example of two citations that refer to the same paper, yet are cited quite differently, follows:

Friston, K. J., Holmes, A. P., Worsley, K. J., Poline, J. P., Frith, C. D., and Frackowiak, R. S. J. (1995). Statistical parametric maps in functional imaging: a general linear approach. Human Brain Mapping 2:189–210.

[Friston et al. 94] Statistical parametric maps in functional imaging: a general linear approach, Karl J Friston, Andrew P Holmes, Keith J Worsley, J-P Poline, Chris D Frith, Richard SJ Frackowiak. Human Brain Mapping Vol. 2(4), pp. 189–210.

For this reason, citation parsing is a difficult task and has already been addressed many times by the community. Mainstream citation extraction approaches use heuristics [5], machine learning techniques [6,7], knowledge-based approaches [2,4], and other methods to overcome this issue. However, due to the different ways that a document is formatted - and the different languages - this process may be time consuming. In the Related Work section, we will present more thoroughly the existing techniques.

The second step is citation matching. This phase regards the enrichment of the extracted citation. The title, the authors and the other extracted fields are matched against the repository of interest. Since the metadata of the repository are also structured, this matching seems like a simple string match. However, there are also several problems to tackle, like the different ways that author names (or other metadata) are written, title misspellings, publications with the same title and other issues.

In this work, we match the publications' plain text with the repository metadata and produce directly the enriched matched citations. We are eliminating

1 https://europepmc.org/.
2 http://arxiv.org/.
3 http://www.crossref.org/.
4 https://www.openaire.eu/.

the citation parsing step, replacing it with a fast text filtering step whose purpose it to keep only the sections in the text which contain references. Figure 2 presents the workaround of the proposed method.

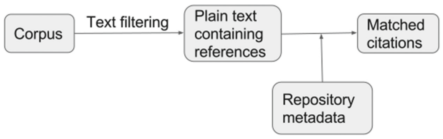


Fig. 2. Citation matching workflow

The first step uses heuristics and high pass text filtering to extract the whole reference section from the text and any other section that may contain references.

The next step is the final citation matching step. The structured repository metadata are matched against the references sections from the text using database and pattern matching techniques.

Our technique is able to extract references from anywhere in the text, including footnotes, and not only from the references section. Moreover, since the full-text is not parsed to produce a structured citations list, the algorithm does not depend on the references or the publication's format. Finally, as the experiments have shown, the presented method is able to provide citation links between a corpus consisting of publications' plain text and a specific repository up to more than an order of magnitude faster than the techniques that parse the citations before the matching.

The rest of the paper is organized as follows. In Sect. 2, we present the related work. In Sect. 3, we lay out our reference section extraction algorithm and in Sect. 4, we introduce our citation matching algorithm, and the implementation details. In Sect. 6, some experimental results are shown.

2 Related Work

While dealing with the citation matching problem, the first issue we need to address is that of data representation. The data can be either structured (e.g. XML) or unstructured (e.g. plain text). E.g., when a publication is in XML format it often has structured references, titles and authors, which can be easily matched against an existing/given database of publications. On the other hand, when provided with unstructured plain text it is important to consider that citations are presented in different formats according to each repository/publisher, making it difficult to produce an algorithm that is able to extract references from all possible repository formats.

To solve this problem: CITESEER [5] identifies the reference section and then uses heuristics and machine learning techniques to find title, author, year of publication, page numbers, and citation tags; CERMINE [12] uses some geometrical, lexical values, the format and some heuristics like the uppercase etc., to extract the references section and machine learning techniques to find reference strings; GROBID [8] and ParsCit [3] use Conditional Random Fields [9]. GROBID and CERMINE work with PDF files, whereas ParsCit works also with plain text.

All these methods target the problem of extracting and parsing citations from a paper. Having completed this step, if someone needs to match the citations against a metadata database, he has to match the extracted titles, authors and the other metadata. While this task seems simpler, it is also difficult since it includes title and metadata matching. Title matching is difficult due to possible typos. Moreover, when a title matches with another, it is a good hint that we are talking about the same paper, but this is not always the case. Matching metadata including authors, journal information etc. is very difficult due to the different ways this information is presented. The existing tools target mainly the extraction of structured citation lists, without matching them to metadata databases.

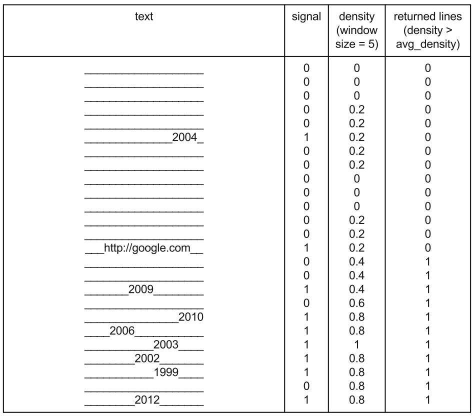
3 Reference Sections Extraction Algorithm

As already mentioned, the main challenge is the different ways that citations are presented. Thus in order to be able to locate them in a publication it is crucial to use global heuristics. Ideally, we should find a globally common characteristic. Such a characteristic is the appearance of years or URLs in the text: in all publications, the references sections are dense in dates and URLs. Using this feature, we split each publication text on its newlines, remove blank and small lines that consist of less than ten characters, and we mark all lines where at least one year (4 digit numbers, between 1900 and 20xx) or URL appears in their context, producing a list of marked or unmarked items. By considering this list as a signal in time, we can then run a high-pass filter process, keeping regions with higher than average density of dates or URLs. In this way, we are able to extract not only the references section but also text that contains citations anywhere in the body of the publication.

An illustration of how the reference extraction algorithm works can be seen in Fig. 3. In the first step, the lines containing URLs or dates are marked. The next step calculates the density of such patterns per window. In this example, a window consists of 5 lines, so the lines from 4th to 8th constitute a window. The density for the lines of this window is 0.2, since a year appears in one line (1/5). In the final step we mark the lines with density higher than average. The unmarked lines are filtered out, and the sections from the text which may contain references are extracted[5].

This step's main goal is to reduce, as fast as possible, the amount of text that will be processed without missing any valid citations. We do not care about false positives as these will be eliminated during the next steps.

[5] Note that this is an unreal example where the references section covers almost half of the complete text. When the main body is larger, the average density of dates and URLs is lower and possible references in footnotes are not filtered out.



text	signal	density (window size = 5)	returned lines (density > avg_density)
________________	0	0	0
________________	0	0	0
________________	0	0	0
________________	0	0.2	0
________________	0	0.2	0
____________2004_	1	0.2	0
________________	0	0.2	0
________________	0	0.2	0
________________	0	0	0
________________	0	0	0
________________	0	0	0
________________	0	0.2	0
________________	0	0.2	0
___http://google.com__	1	0.2	0
________________	0	0.4	1
________________	0	0.4	1
______2009_______	1	0.4	1
________________	0	0.6	1
______________2010	1	0.8	1
____2006__________	1	0.8	1
__________2003____	1	1	1
________2002______	1	0.8	1
__________1999____	1	0.8	1
________________	0	0.8	1
________2012______	1	0.8	1

Fig. 3. Reference extraction algorithm with window size = 5

4 Title and Metadata Matching

Citation matching is the next step of the presented algorithm. There are several problems that we have to address. First, matching titles with plain text can be very time-consuming, because title lengths vary. Moreover, the same title does not always refer to the same paper. We also need to match with metadata. Citations may be written in different formats, the order of publication metadata varies, author names can be written in different ways (John Smith or J. Smith). Our algorithm solves the title and metadata matching problem in the following steps:

- Preprocessing phase (possibly offline)
  - Normalization of structured metadata
  - Creation of characteristic inverted index
- Matching phase
  - Title matching
  - Validation of results using metadata matching

4.1 Preprocessing

Normalization of Metadata. At first, we normalize the titles and other publication metadata by:

- Reducing spaces between words
- Replacing punctuation marks with underscores
- Converting text to lower case

For example if we have the title:
"The PageRank citation ranking: Bringing order to the web"

we transform it as follows:
"the pagerank citation ranking_ bringing order to the web"

The preprocessing phase addresses misspelling issues related to number of spaces, punctuation and case sensitivity. Exactly the same preprocessing procedure has to be applied to the publications' fulltext before the final matching.

Characteristic Inverted Index. As mentioned before, title matching is a very demanding and time-consuming task. We produce an inverted index based on all trigrams that appear in the titles. With the term trigram, we refer to any sequence of three words in the titles. We execute a JOIN operator between the text and title trigrams. When a trigram from the text matches with a trigram from a title, we examine if the whole title matches in the text. If yes, we have a matched title, if not we have a title miss. Using trigrams instead of bigrams or single words, we reduce possible title misses. On the other hand, we do not use larger N-grams, because many titles consist of just three terms. A typical trigram-based inverted index for the above example title is shown in Table 1.

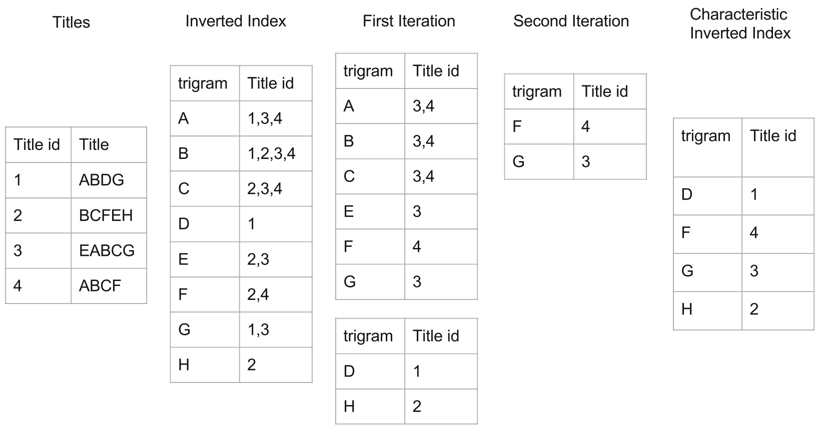
Table 1. Trigram-based inverted index

Trigram	Title id
The pagerank citation	1
Pagerank citation ranking_	1
Citation ranking_ bringing	1
Ranking_ bringing order	1
Bringing order to	1
Order to the	1
To the web	1

Obviously, since this index contains all trigrams appearing in all titles, it is both memory and computationally expensive to be used in a relational JOIN operation. Moreover, common trigrams that may co-exist in many titles could lead to a huge number of matching trigrams, thus to a huge number of title misses.

A way to reduce the size of the index and the title misses is to only use *identifying trigrams* in the index. An identifying trigram, is a trigram that appears in only one title. So, an ideal inverted index would only contain one identifying trigram per title. Because the ideal inverted index is unattainable most of the time, we try to approximate it using a simple heuristic iterative method. First we build the full trigram-based inverted index that contains all trigrams appearing in all titles. From this index we "pick" the trigrams and titles that only appear once, and remove them from the full inverted index. We repeat this procedure iteratively, increasing the threshold of trigram appearance count, until we have fully covered the set of titles of the full index. This produces a characteristic inverted index containing, for each title, a trigram that appears in as few other titles as possible.

The characteristic inverted index is used in a relational EQUI-JOIN between the trigrams appearing in the text and the index trigrams. Here follows an example of the described algorithm. Let A,B,C,D,E,F,G,H be trigrams and consider the example shown in Fig. 4.



Titles

Title id	Title
1	ABDG
2	BCFEH
3	EABCG
4	ABCF

Inverted Index

trigram	Title id
A	1,3,4
B	1,2,3,4
C	2,3,4
D	1
E	2,3
F	2,4
G	1,3
H	2

First Iteration

trigram	Title id
A	3,4
B	3,4
C	3,4
E	3
F	4
G	3

trigram	Title id
D	1
H	2

Second Iteration

trigram	Title id
F	4
G	3

Characteristic Inverted Index

trigram	Title id
D	1
F	4
G	3
H	2

Fig. 4. Characteristic inverted index creation

The goal is to select the minimum subset of trigrams that covers the full set of titles, such that selected trigrams have minimum number of assigned titles. Figure 4 presents the steps to produce the characteristic inverted index. In the **first iteration**, we select D and H which appear uniquely in titles 1 and 2, and we remove these trigrams and titles from the index. In the **second iteration**, we are able to identify titles 3 and 4 with trigrams E,G and F. We may use E or G to identify title 3. In this situation, we select the longer trigram according to the number of characters it contains. After removing these trigrams (F,G) and titles (3,4) from the index, it turns out that no titles remain, so our characteristic inverted index is ready.

When the inverted index is complete, a query runs which scrolls a window over the publication's fulltext, extracts all the trigrams and matches them to our inverted index. If a match happens then the full title from the inverted index is matched with the context of the trigram in the publication's text. This way we ensure achieving a high recall rate since all trigrams from the text are joined.

4.2 Matching

Title Matching. A query extracts the references sections from the text, scrolls a window over the extracted text, extracts all the trigrams and matches them to the characteristic inverted index. Experimentally, the size of the window is set by default to 60 whitespace separated strings, in order to include at least the title and the metadata. If a match happens then the full title from the inverted index is matched with the corresponding window of text.

Validation of the Results. After matching a title we have to deal with the difficult problem of disambiguating and filtering out false matches. We use the following techniques:

- We create a bag of words that contains author names, publication dates, publishers, journal names for each metadata record. We pattern match this bag of words to our window of text. Each category weighs differently than the others. So, the author matches weigh more than the other metadata, the author surnames weigh more than their first names, the journal names and the publishers more than the publication dates.
- Each match also weighs differently according to its distance (in number of words with more than two characters) from the title. If a word matches, the confidence value is increased by its weight and inversely proportional to its distance from the title in words.
- The length of the title (number of words) is also considered, since a larger matched title is more possibly a true match.

The following equation shows how the confidence value is calculated:

$$Conf = \frac{\frac{10*F(AS)+3*F(AF)+3*F(PY)+5*F(JN)}{MAXVAL} + \frac{L(t)}{L(C)}}{2}, \quad (1)$$

AS, AF, PY, JN are author surnames, first names, publication years and journal names respectively. Function F sums the distance weights for all the occurrences of the input pattern. If a pattern is matched 3 words away from the title, its distance weight is 1/3. L(t) is the length of the matched title, while L(C) is the length of the window. MAXVAL is the maximum value, if all the words in the window are matched with author surnames. It is used so that the final confidence value is between 0 and 1. After experimenting with various datasets, we have defined 0.1 as an appropriate threshold. If this value is above the threshold, then the citation is marked as true positive, else as false.

The fallback of this technique is that the context used for calculating confidence value has stable length for speed purposes, so it may contain strings from previous or next citations. Nevertheless, manual curation of experimental results indicates that less than 1% of matches are false positives because of metadata that match with an adjacent reference.

4.3 Implementation Details

Our algorithm is implemented on top of madIS [11], a powerful extension of a relational DBMS with user-defined data processing functionality. MadIS is built on top of the SQLite API[6].

[6] https://www.sqlite.org/.

MadIS allows the creation of user-defined functions (UDFs) in Python and it uses them in the same way as its native SQL functions. Both Python and SQLite are executed in the same process, greatly reducing the communication cost between them. This is a critical architectural characteristic and has a positive impact on joint performance.

MadIS is highly scalable, easily handling 10 s of Gigabytes of data on a single machine. This benefit transparently carries over to distributed systems (e.g., Hadoop [10], Exareme [1]) which can use madIS in each node.

In madIS, queries are expressed in madQL: an SQL-based declarative language extended with additional syntax and user-defined functions (UDFs). One of the goals of madIS is to eliminate the effort of creating and using UDFs by making them first-class citizens in the query language itself.

The expressiveness and the performance of madIS along with its scalability features were compelling reasons for choosing it to implement our algorithm. Our citation matching software is open source and hosted by Github.[7]

5 Experiments

We ran three experiments to evaluate the proposed method. In the first two experiments, we test some important features of our method, whereas in the third experiment we compare our method to GROBID and ParsCit. Our experiments ran on an Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz processor with a 500 GB SSD disk and 16 GB RAM, running Ubuntu Server 14.04 LTS.

The aim of our **first experiment** was to prove the benefits of the reference extraction algorithm. We ran the citation matching algorithm to the full publications' text (without using the reference extraction algorithm) and compared the results to those that are produced if the references are extracted. For this experiment, we used 100 publications from both arXiv and PubMed repositories (50/50), matching citations with 9.6 millions publications from OpenAIRE. ArXiv and PubMed were selected, as their deposited publications do not share similar reference formats and cover many scientific areas including medicine, physics, computer science and others. The results are shown in Table 2. The precision and the recall rates are based on manual validation of the results.

In case 1, running citation matching on the fulltext, we find a total of 289 citations. The recall rate (97.40%) is high and notably the 5 citation misses are all cases that the algorithm was able to match but are filtered out due to low confidence value. In case 2, we only process 12.3% of the total text lines, so the processing is about 8 times faster. Case 2, misses 12 more correct references that were found by case 1. By increasing the window size from 5 to 7 in case 3, we only increase by 0.3% the number of processed lines but we also find all the references missed by case 2. Finally, it turns out that the use of the reference extraction algorithm is very advantageous in terms of precision, since in case 1 the precision is very low (67.10%), whereas in cases 2 and 3 there are not any

[7] https://github.com/madgik/recital.

Table 2. Evaluation of matched citations

Case	1. full pub text	2. ref extraction window size = 5	3. ref extraction window size = 7
High confidence matches	289	182	194
True citation misses	5	17	5
High confidence precision	67.10%	100%	100%
Recall	97.40%	91.50%	97.40%
% of total text lines	100%	12.30%	12.60%

false positives. However, while inspecting the false positives of case 1, it turned out that the titles of the publications themselves that are placed on the top of the PDF had matched. We can avoid this if we simply exclude the first lines of the plaintexts in the pre-processing phase. Thus, the impact of the reference extraction algorithm on the processing time is very important since the main body of the algorithm is only applied on a small percentage of total text lines.

Our **second experiment** concerns the citation matching algorithm. We ran our algorithm on about 450 K publications' fulltexts retrieved from the ArXiv repository. The purpose of our experiment was to find citation links to OpenAIRE publications. The first step for doing this is building the characteristic inverted index. OpenAIRE publication index, during the experiments, consisted of 9,598,093 publications in which there were 8,168,090 distinct titles. While creating the characteristic inverted index, we managed to extract identifying trigrams for 7,430,948 titles whereas for the rest we increased the threshold of trigram appearance count, until we covered the full set of titles. Then, we ran the algorithm using the text references extraction algorithm with window size set to 7. The results of our experiment are shown in Table 3.

Table 3. Citation matching in OpenAIRE

Publication count	OpenAIRE publication count	High confidence Citation matches	High confidence Precision
450.4K	9.6M	968,880	99%

In this experiment, after validating a sample of 200 citations, we found two false positives. These two cases show one main disadvantage of our method. Rarely, there are titles which are substrings of other titles. This is not a problem when the other metadata differ, because the confidence value remains low. But when the authors are also the same, then we may end up with false positives. Consider the citations below:

Caire, Giuseppe, and Daniela Tuninetti. "The throughput of hybrid-ARQ protocols for the Gaussian collision channel." IEEE Transactions on Information Theory (2001).

Caire, G., and D. Tuninetti. "ARQ protocols for the Gaussian collision channel." IEEE INTERNATIONAL SYMPOSIUM ON INFORMATION THEORY. 2000.

These refer to two different publications with the same authors, where the second publications' title is a substring of the first.

Our **last experiment**, compares our method to GROBID and ParsCit. Both these tools extract and parse citations from publications. GROBID processes PDF files while ParsCit processes plain text. These tools extract structured citations and the matching to a given list of publications' metadata can be done as a next step. Our algorithm (and ParsCit) works with plain text, so we had first to convert the PDFs. We did it using the Unix util *pdftotext* without any parameters. We compare the methods on a dataset consisting of 15K PDFs, and a metadata database consisting of about 375 K publications from ACM[8]. GROBID ran in multi-threaded mode (8 threads), while ParsCit and our method ran in single-thread. Table 4 shows the execution times, and the matched citations.

Table 4. Comparison with GROBID and ParsCit

Method	PDF convertion time (minutes)	Preprocessing (minutes)	Extraction (minutes)	Matching (minutes)	Matched citations
GROBID	-	-	101	0.9	32595
ParsCit	11	-	720	0.9	31989
Proposed	11	1.9	0.5	2.2	39027

As can be seen, the proposed method is much more efficient in terms of speed. The preprocessing time regards the normalization of the metadata and the creation of the characteristic inverted index. The proposed method does not extract stuctured citations, but only sections from the text that may contain references. In this step, GROBID extracts 129252 structured citations with titles while ParsCit extracts 125328. In the next step, the titles are matched against the metadata database. Grobid produces 33206 title matches while ParsCit produces 32686. The presented numbers regard the citation matches where not only the titles, but also metadata like publication year and author names have matched.

Our method matches the metadata database with the plain text and produces 39027 high confidence matched citations. All three methods extract in common 31899 matched citations. Our method matches 7089 citations more than GROBID and 7105 citations more than ParsCit. The validation of a sample of 200 citations showed that 97% of them were valid and the rest were false alarms. GROBID also extracts 657 citations that are not extracted using the proposed method. These citations were missed mainly due to the PDF conversion. Comparing our method with ParsCit, where the same PDF conversion tool was used, it seems that our method misses just 67 citations. This supports our claim that our method achieves high recall rates when processing plain text.

[8] http://dl.acm.org.

6 Conclusions

Given a known database of publications' metadata, we propose a fast and accurate citation matching method from a corpus of publications to the metadata database. Our method does not use citation extraction but citation matching. This means that we do not extract external citations but we target extracting citations within a given dataset. That is why we can avoid using time-consuming machine learning techniques to extract full citation metadata from the publications' fulltext. Hence, the algorithm achieves the same accuracies and faster processing times regardless of the format of the publication's fulltext and in most cases even its language. Moreover, by matching every trigram in the fulltext, we ensure that our method achieves higher recall rates than methods that extract citations' metadata from fulltext before applying the match.

References

1. Chronis, Y., et al.: A relational approach to complex dataflows (MEDAL 2016) (2016)
2. Cortez, E., da Silva, A.S., et al.: Flux-CiM: flexible unsupervised extraction of citation metadata. In: Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries, pp. 215–224. ACM (2007)
3. Councill, I.G., Giles, C.L., Kan, M.Y.: ParsCit: an open-source CRF reference string parsing package. In: LREC 2008 (2008)
4. Day, M.Y., Tsai, T.H., et al.: A knowledge-based approach to citation extraction. In: International Conference on Information Reuse and Integration. IEEE (2005)
5. Giles, C.L., et al.: Citeseer: An automatic citation indexing system. In: Proceedings of the third ACM Conference on Digital Libraries, pp. 89–98. ACM (1998)
6. Han, H., et al.: Automatic document metadata extraction using support vector machines. In: Joint Conference on Digital Libraries. IEEE (2003)
7. Lawrence, S., Giles, L.C., Bollacker, K.: Digital libraries and autonomous citation indexing. Computer, 67–71 (1999)
8. Lopez, P.: GROBID: combining automatic bibliographic data recognition and term extraction for scholarship publications. In: Agosti, M., Borbinha, J., Kapidakis, S., Papatheodorou, C., Tsakonas, G. (eds.) ECDL 2009. LNCS, vol. 5714, pp. 473–474. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04346-8_62
9. Peng, F., McCallum, A.: Information extraction from research papers using conditional random fields. Inf. Process. Manage. (2006)
10. Shvachko, K., et al.: The HADOOP distributed file system. In: 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10. IEEE (2010)
11. Stamatogiannakis, L., et al.: madIS - extensible relational DB based on SQLite, https://github.com/madgik/madis. Accessed 5 March 2017
12. Tkaczyk, D., et al.: CERMINE - automatic extraction of metadata and references from scientific literature. In: 11th IAPR International Workshop on Document Analysis Systems, pp. 217–221. IEEE (2014)