Structured-Content Extraction from the Web for Bibliographic Reference Generation

Ramon Xuriguera*, Marta Arias**

*rxuriguera@lsi.upc.edu
**marias@lsi.upc.edu
UPC Barcelona Tech

Abstract. In this paper we present a system that automatically creates bibliographic indexes from a collection of PDF files by using the file contents to search the Web and later extract the information from the resulting pages. We pay special attention to the techniques used for extracting this data as well as the automatic generation of extraction rules and their evaluation.

1 Introduction

Working on a research project surely implies spending vast amounts of time reading related publications and the corresponding files, mostly in PDF format. Once the research is done, researches have to generate bibliographic indexes from these articles, which can be a very tedious and time-consuming task, even when using existing tools. We believe that this task could be automatized to a large extent. In fact, the subject of this paper is a prototype application that achieves precisely this with minimal or no user intervention.

There already exist many services and applications that attempt to tackle this problem in one way or another. We could divide the existent software in the following categories:

- Digital libraries: all of the databases available online that contain information on publications, usually focusing on a certain field. Examples include ACM or SpringerLink [1].
- Publication search engines like Google Scholar or Microsoft Academic Search [2] that collect information on publications and compute things like how many times an article has been cited.
- Finally, there are reference managers such as Mendeley [3] or the like, useful to index read articles and easily import and export their references. While these applications most often include features to search for references in some digital libraries, to this day we have not found any of them that actually undertakes the task of finding bibliographic information by just looking at the content of the files.

---

1. http://portal.acm.org, http://www.springerlink.com
2. http://scholar.google.com, http://academic.research.microsoft.com
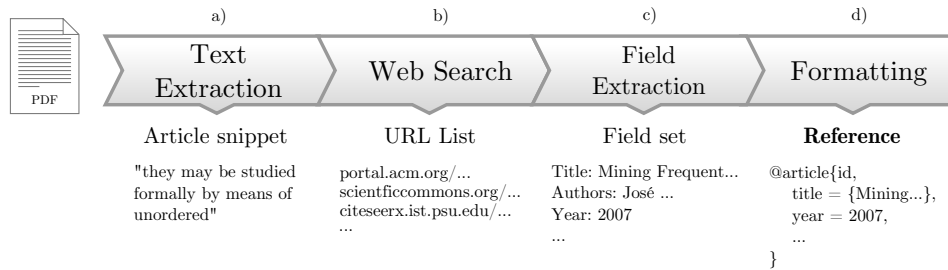3. http://www.mendeley.com

FIG. 1: Reference extraction process overview. a) Extract words from the text. b) Use search engines to find pages with bibliographic information. c) Extract reference fields. d) Format the results

In this paper we present a self-updating system that automatically creates bibliographic indexes from a collection of PDF documents and can regenerate extraction rules when they no longer work. Even though we focus on the references domain, the core techniques implemented here could be also used in other applications where information has to be extracted from multiple sources that may vary over time. This is part of a final degree project at UPC Barcelona Tech; the original tech report can be found at Xuriguera (2010). Source code is GPL-licensed and can be found in the git repository [4].

2   System Overview

One of the first ideas that might come to mind when trying to accomplish the task of generating bibliographic indexes from PDF articles is to extract the content of the paper and directly recognize and store the fragments to include in the bibliographic reference. This approach has some major drawbacks. After extracting the content from PDF files we would end up with free text that would have to be analysed using natural language processing techniques, something which is usually difficult and goes beyond the scope of this project. Besides, even if we were able to interpret the text we would still be missing valuable information that is not generally part of it, e.g. number of pages, ISSN, volume number.

The proposed solution avoids all these problems by using the Web as a source of semi-structured information. It consists of the four steps depicted in figure 1 and the basic idea is to use potentially searchable text snippets from the PDF file to retrieve known pages – generally digital libraries – that contain all the information needed to build the reference. Once this is done, formatting is straightforward. The system architecture is very similar to the one presented by Limanto et al. (2005) and Wang and Lochovsky (2003) with the main difference being that we do not have a crawler constantly scraping, but use search engines like in Etzioni et al. (2004) instead.

---

4. https://github.com/rxuriguera/bibtexIndexMaker

Document

> Columbus is a town in Polk County, North Carolina, United States. The population was 992 at the 2000 census.

Wrapper

Applying the wrapper

2000

FIG. 2: Intuitive wrapper description

By potentially searchable snippets we mean short sequences of words that are likely to yield relevant results when used as quoted queries in main search engines such as *Google* or *Bing*. It has proved to be a good practice to select sequences of about 6 to 10 words belonging to the abstract of the publications. Article abstracts usually appear next to other bibliographical information and are generally indexed by searchers. For the described procedure to work, the system must then use some sort of extraction rules on the pages returned by search engines. Fortunately, search results tend to be quite homogeneous for multiple publications on the same field, so a relatively low number of rules is enough to get acceptable results.

Our application's main strength lies in the automation of the rule generation and maintenance process, which results in very little or no user intervention.

3 Reference Extraction

The reference extraction process is done by using wrappers, which can be seen as filters that have some knowledge on the structure of the source pages (see figure 2). Internally, they are represented as sets of rules. We initially attempted to generate rules that extracted whole BibTeX references from the results. While it is quite straightforward to implement these wrappers manually, it generally required some degree of understanding of each site and, sometimes, even a bit of reverse engineering. This makes it unfeasible to generate them mechanically. Moreover, we have empirically tested that references are given in a wide range of formats depending on the digital libraries and the main topics of the publications included in them.

In order to avoid this, the wrappers implemented in the presented application are field-oriented and aim to pick single pieces of information from specific web pages. The extraction process is as follows:

- Given a set of search results for a publication, the system checks if it has any wrappers for them.
- If it does, then applies the best available wrappers (more on this in 4.1) for each field.
- If the extracted information is not valid, it tries with some other wrapper or search result.

Listing 1 shows this process in a bit more detail while still keeping it greatly simplified.

```
function ExtractReference(file):
  raw := RawTextFrom(file)
  # Get the strings to be used as queries
  queries := QueryStrings(raw)
  # Try wrappers for each result
  for result in SearchEngineTopResults(queries):
    # Extract and validate values for each wrapper
    for fieldWrapper in AvailableWrappersFor(result):
      value := ExtractFieldValue(fieldWrapper, result)
      valid := ValidateFieldValue(value, raw)
      if valid:
        UpVote(fieldWrapper)
      otherwise:
        DownVote(fieldWrapper)
      # The following would omit replacing valid values
      AddFieldToReference(reference, value, valid)
  Validate(reference)
  return reference
```

LST. 1: Pseudocode describing the main steps for extracting a reference from a file

Reference Validation Quality assessment tests on the extracted fields are not only useful to retry wrong extractions but also to inform the user of possible errors, deprecate rules that no longer work and to exclude incorrect references during the generation of new wrappers. A reference is considered to be valid if a specific set of fields have been successfully tested. These fields are weighted so more importance can be given to fields such as the title or authors than, for instance, the volume number.

Validation is done in different ways. When applicable, the system checks that field value appears in the original file. In other situations, e.g. for fields that are not usually part of the publication header, pre-defined regular expressions can be used.

4   Wrapper Generation

The automatic generation of new wrappers is necessary so rules can be easily maintained while reducing to a minimum the amount of manual work to be done by the user. The generation process requires examples, which in this context will consist of tuples with the value that has to be extracted and the URL of the web page containing it. For highly-structured pages the minimum number of examples needed to build good-enough rules is as low as one or two. On the other hand, if the desired information is mixed with other text, more noise has to be omitted and having a greater number of them is extremely useful. To ease the task of collecting examples, references that have been previously extracted are stored along with the source URL so they can also be used when generating new wrappers. As before, highly simplified pseudocode for wrapper generation is shown in listing 2. Basically, rules are built by following these steps:

1. Create rule for one example.

2. Merge it with the previous rules. If not possible, create new rule.
3. Repeat while there are more examples available.

Specific details vary for each of the implemented type of rules, which we describe next in more detail.

Path Rules define the location of an element by specifying its path in the HTML tree. They are represented by a list of triplets with the element name, attributes with their corresponding values and, finally, the sibling number. These rules are created bottom-up and stopping once the first element of the path is unambiguous. Example:

```
[('table',{'width':'100%'},7), ('tr',{},0), ('td',{},0)]
```

Two paths are only merged if they have the same length, element names and attributes, i.e. the only thing that varies is the sibling number. If two paths cannot be merged, different wrappers are then created. When applying this kind of rules, if more than one element is matched, all values are evaluated before choosing. As one may expect, we have found that CSS class names are especially useful for distinguishing among different fields.

Regex Rules are a bit more interesting. They define the location of a piece of information within some text, which is necessary when the same HTML element not only contains the desired value but some other information as well. For example, getting the volume number from a text that contains other fields

```
Vol. 27, No. 6, pp. 1204−1209.
```

would require an expression like [5]

```
Vol\.\ (?:.*)\,\ No\.\ (.*)\,\ pp\. (.*)\.
```

This is obviously not the most preferable regular expression that would allow us to get the volume number, but it is easy to generate it from a set of sufficiently different examples. The more different the examples, the more general the final rule will be. This kind of expressions are created by stripping the value to be extracted from the text and comparing it to other examples. The comparison is done with Python Difflib module, which internally uses, Ratcliff and Metzener (1988). If texts have a similarity greater than a given threshold, then they are merged by generalizing the substrings on which they disagree.

The first approach for creating these rules was similar to the one used in WHISK, Soderland (1999), that is, removing the value and checking the text on the sides. This was later abandoned because these characters differed considerably among pages from the same library. Looking at it in perspective, it could be interesting to combine this strategy with the current one.

---

5. In Python's regular expressions, `(?:<regex>)` keeps the string matched by `<regex>` while `(<regex>)` discards it.

```
function GenerateWrappers(url):
  for exampleSet in ExampleSets(url):
    # Different rules will be created depending on the
    # field
    if field is multi-value:
      rulers := [PathRuler, SeparatorsRuler, RegexRuler]
    otherwise:
      rulers := [PathRuler, RegexRuler]
    wrappers := []
    for ruler in rulers:
      rules := GenerateRules(ruler, exampleSet)
      # The following creates new wrappers if necessary
      UpdateWrappers(wrappers, rules)
    # Discards potentially incorrect wrappers
    Prune(wrappers)
    return wrappers
```

Lst. 2: Pseudocode describing the main steps for generating new wrappers

Those fields that may have more than one value are also taken into account. By using the same path rules described above and a generalization of regex rules that apply the regular expression to each value, we can extract multi-valued fields that are located in different HTML elements but that have a similar path (e.g. siblings or cousins). However, there is another situation that should be considered.

Separator Rules can be used when the multiple values of a field are located within the same HTML element. As their name suggests, these rules consist of a list of separator strings – usually ", " and " and ". Given an example, they are created by removing the values to be extracted and keeping the substrings in-between. Merging is done by joining lists.

For the specific problem of reference extraction the only field that allows multiple values is the author name. For them, an additional fixed rule to distinguish between name and last name is applied.

4.1 Wrapper Evaluation

By assigning a weight to each wrapper the system can decide which ones should be applied first. Weights are updated every time a wrapper is used by computing the ratio of correct extractions to the incorrect ones. As Evan Miller points out in Miller (2009), this presents some problems when comparing new wrappers with others that have been frequently chosen. Nevertheless, in this context we want to discard malfunctioning wrappers after as few iterations as possible, so this simple ratio seems appropriate. Not having enough working wrappers for a specific site will make the system notice that it is time to regenerate the rules. Theoretically, this situation should only happen when the format or design of the page has been altered.

5   Experimental Results

In order to test the system we have collected a set of about 30 publications uniformly distributed over three topics: Computer Science, Medicine and Economics. Although constrained by the lack of a large set of different publications, the results shown in this section should be enough to prove the feasibility of the proposed solution.

The chart on the right shows the percentage of publications in the set for which the extracted snippets yielded relevant results. The engine with the best results was *Google*. Furthermore, it seems that most engines peak around a value of 6 words, so our system uses these settings by default although they can be changed by the user.
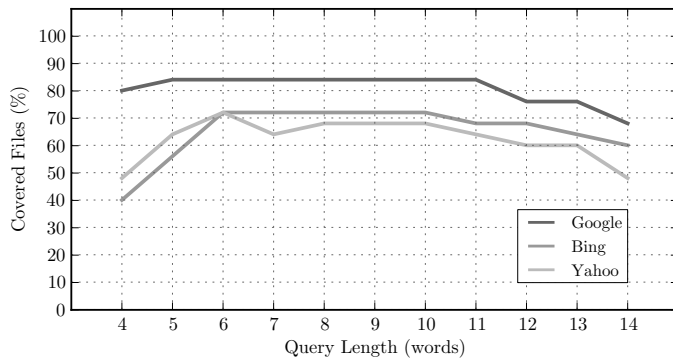
FIG. 3: Query coverage by search engine

We now generate wrappers for the digital libraries: ACM, SpringerLink, InformaWorld and Ideas by using 4 examples for each of them. Given a set of five articles indexed by these libraries and their correct references, we have computed the quality of the extractions. Results are shown in figure 4.
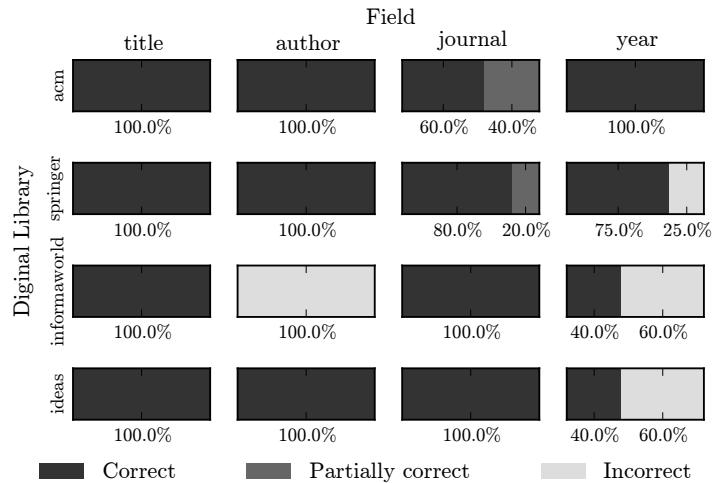
FIG. 4: Extracted fields correctness

Partially correct refers to those cases where the correct value of the field was returned with some additional text that should not be part of it. For example: pp. 102-120 instead of

`102-120`. As it can be seen on the chart grid, the most difficult field to extract is the one corresponding to the year of publication. This was quite surprising at first, but it turned out that this happens because years tend to appear in multiple places within the pages. For instance, if an article is very recent, its publication year might well appear in the copyright notice as well.

In case the generated wrappers do not work, it is still possible for the users to tune the rules as needed. In practice, with small modifications the correctness of the extractions is greatly improved. Additional tests and results can be found in the original report Xuriguera (2010).

6   Conclusions and Future Work

We have implemented a system for generating bibliographic references from PDF files by extracting information from the Web. Some of the techniques that we have used are based on assumptions that, within this specific domain, work well enough. It should be noted, though, that the way in which rules are created and applied can be easily generalized to work with any similar problems where information has to be extracted from a number of sources that change over time, while keeping the amount of manual maintenance as low as possible.

There are many areas of the process that could be refined. We have already pointed out some limitations on regular expression rules creation and given a hint on how to do it. In addition, user corrections on extracted references could be used as feedback on wrappers that are not working properly. From an application's point of view, there is still a lot of work to do too, especially for making it available to end-users. In this sense, it would be interesting to create plug-ins for reference managers like Mendeley, and make it possible to share and auto-update wrappers. Since this is an open source project, we actively encourage everyone to fork, improve and extend it.

References

Etzioni, O., M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates (2004). Web-scale information extraction in knowitall: (preliminary results). In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, New York, NY, USA, pp. 100–110. ACM.

Limanto, H. Y., N. N. Giang, V. T. Trung, J. Zhang, Q. He, and N. Q. Huy (2005). An information extraction engine for web discussion forums. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, New York, NY, USA, pp. 978–979. ACM.

Miller, E. (2009). How not to sort by average rating. `http://www.evanmiller.org/how-not-to-sort-by-average-rating.html`.

Python Difflib. Difflib - helpers for computing deltas. `http://docs.python.org/library/difflib.html`.

Ratcliff, W., J. and D. Metzener (1988). Pattern matching: The gestalt approach. *Dr. Dobb's Journal*, 46.

Soderland, S. (1999). Learning information extraction rules for semi-structured and free text. *Machine Learning 34*(1), 233–272.

Wang, J. and F. H. Lochovsky (2003). Data extraction and label assignment for web databases. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, New York, NY, USA, pp. 187–196. ACM.

Xuriguera, R. (2010). Bibtex bibliography index maker. `http://hdl.handle.net/2099.1/9455`. Written in Catalan.

Résumé

Cet article présente une application qui crée automatiquement un index bibliographique á partir de fichiers PDF. Pour obtenir le contenu de la fiche bibliographique, ce logiciel recherche le Web. Nous portons une attention particulière aux les techniques nécessaires pour extraire cette information.