Scraping the ACM Digital Library

Donna Bergmark (bergmark@cs.cornell.edu)
Cornell Digital Library Research Group

Paradee Phempoonpanich (pp73@cornell.edu) and Shumin Zhao (sz37@cornell.edu)
Cornell Computer Science Dept.
4128 Upson Hall, Ithaca NY 14853

Abstract

As part of a larger project to automatically reference link the online scholarly literature, an attempt to analyze PDF documents was undertaken. The ACM Digital Library was used as the corpus for these experiments. With the current PDF and HTML analysis tools, roughly 80% accuracy was obtained in the automatic extraction of reference linking information.

1 Introduction

The Association for Computing Machinery (ACM) has put a large portion of their literature online. This makes reference linking from and to the ACM literature possible. Reference linking [14] allows one to go directly from the "active paper" (which one is reading) to a referenced paper, assuming the referenced paper is also online.

Often, authors of newer "born digital" documents insert links to online references. However, this does not always happen, and it was not done with older literature. ACM has reference linked its new digital library to other items in digital library[15], but this still leaves many unlinked references.

For more aggressive reference linking[8], it is necessary to "scrape" the active document to extract the references and their bibliographic data; the bibliographic data is then used to look up one or more online locations for the referenced work; the references are turned into "actionable links" overlaid on the active document; finally if the user selects the link, a copy of the referenced work is retrieved and shown to the user.

This paper reports on experiences doing the first of the four tasks: scraping ACM papers for reference data. Other projects (CrossRef[13], OpCit[10, 1], ResearchIndex[5, 12, 6], SCI[2, 9]) provide alternative approaches to this task. In fact, much work of interest is going on in reference linking today [7] because of the huge growth in the amount of online scholarly literature. The principal need is for automatic techniques such as those used in this project.

2 Methodology

ACM was kind enough to provide us with a copy of their digital library to see what we could do in the way of scraping it for reference data. We already have software to do this for HTML documents [4], but papers from ACM journals, proceedings, and magazines are in PDF format.

Rather than manipulating the PDF directly, we decided to take the approach of converting the PDF to HTML and analyze it using our existing software. This paper reports on our experiences so far with this endeavor. We describe the results for some of the papers we looked at and point out where improvements are needed.

It should be noted up front that the produced HTML need not present a viewable version of the PDF paper; it needs only to have the right tags and content to allow it to be mined for reference linking information.

The first tool we considered was Access from Adobe Systems Inc. This acrobat plugin exports PDF files as HTML for use by sight-impaired persons wishing to use text-to-audio devices. However, we quickly ran into two unsolved problems: (1) Adobe explicitly forbids the use of their Access server merely for converting PDF to HTML, and (2) no command line interface to Access was known, nor was the source available.

We then discovered the PJ package, a very good but mostly undocumented PDF parser from Etymon Systems, Inc (`<http://www.etymon.com>`). We used this as our workhorse and wrote our code in Java (Version 1.4 beta of Java 2 Standard Edition from Sun). We used Java because that is the language in

which PJ is implemented, and we used version 1.4 because of its support for regular expressions.[1]

2.1 Deflation

Two Java programs are used to pre-condition the PDF source. The first one "deflates" the PDF source, and is the uncompression routine distributed with the PJ package. This does not work on all papers in the ACM digital library because a variety of compression schemes were used. However, it appears that none of the ACM papers is encrypted, which is a good thing. We have no hope of processing encrypted material.

The program `UncompressPdf` takes as input a PDF file amd deflates it, that is, deflates any flate-compressed stream object in the PDF input.

2.2 Normalization

If the decompression is successful, the document is next run through a PDF normalizer. This program, `PdfTrim`, straightens out some of the more complex PDF syntax, and prepares the document for further analysis. The main thing it does is to separate multiple PDF operators, along with their arguments, onto separate lines.

2.3 Parsing

The conditioned PDF document is then sent through our parser, `PdfParse`. The parser is a complex program first developed by Phempoonpanich and then greatly extended by Zhao. It makes two passes over the PDF object tree. During the first pass, the first page is parsed, to extract the title and authors. The title is the line drawn in the largest font. The authors are assumed to come next.

The second pass traverses the page tree to extract the content of the document, which is written out as an HTML file. The title is enclosed within `<h1>` tags, and the authors are denoted by `<center>` markup. Each section heading is written out as an `<h3>` element. The remaining content is written out as text. The reference section is split up into one `<p>` element per reference string. Embedded images are ignored, as they are not relevant to reference data analysis; we are interested only in textual reference data.

In addition, each text fragment that begins with `[`, `(`, or `{` is preceded by an HTML comment giving the page number and object number of that text fragment. This is to facilitate insertion of locators back into the original PDF file, to make links "active". (That part of this project remains to be implemented, but in principle it should work.)

The resulting HTML file is handed to our Reference Linking API [4], which then attempts to extract reference linking information. The remainder of this paper first outlines some specific techniques used in our reference data extraction software and then presents our data on how well the approach worked with ACM's source.

[1] Xpdf is a similar package, written in C and C++.

3 Title and Author Section

How well the reference data extraction software can work for the HTML generated from PDF depends on how well the PDF parser worked. It is important for the title and author section of a paper to be extracted, as this information is required to identify the work *citing* each one of the references.

A PDF string must satisfy two conditions to be recognized as the title of the paper being analyzed. First, it needs to have the largest font size on the first page. Second, if there are more than one such strings (separated by text in other font sizes), only the first one is taken as the title. This method works well for most PDF files. Emitting this string as an `<h1>` HTML element is sufficient for the reference data extraction software to pick it up.

After the title section is found, the PDF text right after the title is taken as the author section. Finding the author section is thus dependent on the success of finding the title. The author section is emitted as a series of `<center>` HTML elements, one per line. Disambiguating the contents of the author section is left up to the reference linking data extraction software.

Determining where the author section ends depends on finding a section title. This is a string on one line, with a rather large font, and often contains the word "Introduction". Also, the string "Abstract" in any font size ends the author section.

As part of determining where the author section ends, an attempt is made to determine the format of section titles. This is important in order to determine where the reference section starts. The reference section (1) begins with a section title, and (2) has a title such as "References", "Bibliography", etc. The PDF parser finds too many strings that look like section titles, but this is all right so long as one of them is the title for the reference section.

4 Contexts and References

Although many reference linking applications need only deal with the reference section of a paper, it is quite useful to be able to collect reference *contexts* as well. Contexts are sentences that contain one or more references. For example, a sentence like

> *"Other authors have presented alternative proofs of nonergodicity (Kaplan [11], Rosenkrantz and Towsley [12])."*

is a context that contains two references (also called *reference anchors*). Useful information associated with each reference includes the following:

- Data regarding what work this reference represents (title, authors, year of publication, etc.)
- Locations of online copies of this work
- Contexts in which this reference appeared

4.1 The Range Problem

Sometimes it happens that references such as "`[1]-[4]`" are encountered. The context containing this reference anchor belongs to references `1`, `2`, `3`, and `4`. In this case our existing reference linking software simply expands the range into the four separate references, and the context is associated with each one of them.

4.2 Reference Formats

The reference format is selected from one of six, depending on which format is first encountered. SQUARE_BRACKETS_AROUND_NUMERALS parses references like `[11]`. BRACKETS_AROUND_COMMAED_NAMES_AND_YEARS parses references such as [Bradford and Whaley, 1999; Ford, 1998]. The remaining four formats deal with other variations [3]. For finding the contexts, it is best to nail down the reference format as accurately as possible. Once set, it is not changed for the rest of the paper.

The way in which reference format is determined is as follows:

1. Starting at the top of the document (after title and authors), locate the next sentence (chunk of text from one full stop to the next).
2. Run six format parsers conceptually "in parallel"
3. If one succeeds in finding a reference, continue using that reference anchor parser for the rest of the paper, one sentence at a time.

4.3 Matching Reference Literals to Their Contexts

A *reference literal* is the actual reference string, with all of its details, as contained (typically) in the reference section at the end of the paper. More details can be found in [3], but briefly a reference literal is matched to its contexts as follows:

1. First, the tag is stripped from the literal.
2. A lookup is performed to see if this tag, e.g. `[11]` in canonicalized form, is contained in any of the contexts. If so, those contexts are added to this reference's context list.
3. Otherwise, if the tag matches none of the context anchors and the format is something like PARENTHESES_AROUND_ACRONYMS or BRACKETS_AROUND_COMMAED_NAMES_AND_YEARS, an approximate match is made between the context's reference anchors and the reference literal. For example, if a context contains the reference `(Braley and Hill, 1989)`, it would match up with a reference literal like `Braley, A. and Hill, J. ... 1989`, because every significant word in the anchor appears in the reference literal.

Of course, the success with which the references are parsed is totally dependent on the PDF parser having located the reference section in the first place, and then having successfully split each reference into its own paragraph (`<p>` HTML element) in the second place. The PDF page markup specifies positions of various textual elements. We assume that the first line of each reference literal is aligned to the left. Other lines of the reference might be indented to the right by a little amount of space. Properties such as intra- and inter- line spacing are helpful in separating the reference literals from each other. The main remaining problem is handling references that occur in two columns. The second column is usually intermixed with the first column, since they are rendered on the page "at the same time". This results in some reference literals in the HTML file to be combined.

4.4 Bibliographic Data

References' bibliographic data is extracted from reference literals by the `Deciter` tool written at U. of Southampton [11]. We extract the following information:

- title of the referenced work
- each author of the referenced work

- year of publication
- any urls that are present

The first author's last name, date, and part of the title are used to identify the work. It is possible also for the `Deciter` to extract the journal name, pages, etc. These additional data might be helpful in locating online copies of references.

4.5 Accuracy Metrics

As an objective measure of how well our algorithms work, we use metrics called *Item Accuracy* and *Reference Accuracy*. They are both computed as percentages: elements correctly extracted divided by total number of elements. Item accuracy is 100% if the following are correctly determined: title, each author, date of publication, each context, and each reference. The number of correct references is based on `number of reference literals in the paper` $\times$ `average reference accuracy for this paper`. The average reference accuracy is the number of elements extracted correctly from the reference section divided by the total number of elements therein.

5 Results – an Example Paper

First we present the results of scraping one paper in order to give a concrete feeling for how hard this project is.

A paper was randomly selected from the ACM Digital Library. After decompression and preconditioning, the paper was converted to HTML as described in Section 2. The paper was then subjected to analysis, with the goal of extracting all useful reference linking data from it.

The examined paper contained 29 contexts and 22 references. It appears that flate-compressed streams deal with bit-mapped images, and deflation attempts to convert bit-mapped characters into their ASCII equivalent. This translation did not always go right. Figure 1 lists the improperly rendered anchors. The first column has the reference anchors as they appeared in the published paper; the second column shows the characters into which they were converted during deflation. Errors are frequent, with some letters being mere approximations to the characters intended to be "drawn" on the printed page.

Figure 2 gives some of the consequences of the reference anchors being scanned incorrectly. We won't show the complete table, but it seems clear that the poor translation of square brackets and digits within them hampered the detection of contexts.

What the Reference Should Be	What the Reference was Scanned As
[1]-[4]	[l]-[4]
[8]	[s]
[10]	[lo]
[11]	[ll]
[12]	121
[13]	113
[6]	161
[13]	1131
[18]	[H]
[18]	[IS]
[19], [20], [21]	WI, POI, Pll
[20]	[ZO]
[5]	(51
[5]	[S]
[22]	$221
[5]	(51
[16]	1161
[5]	(51)
[7]	171

Figure 1: Table of mis-scanned references. Each reference (as it happens) is in a different context from each other. Thus 19 of the 29 contexts had an incorrect anchor.

Particularly devastating was that the very first reference anchor in the paper was misspelled (the letter "ell" rather than the digit "1"). Thus despite the references being in the form SQUARE_BRACKETS_AROUND_NUMERALS, the much looser format BRACKETS_AROUND_COMMAED-_NAMES_AND_YEARS was assumed instead. This format does not even require the references to be numeric.

This led to "s", "ll", etc. being treated as reference anchors. This led to some references acquiring many bogus contexts. In other words, since the anchor never matches the tag on the reference literal (which in fact *is* numeric), the fallback of approximate matching is used. In the case of the first anchor, for example, since `[1]` does not match the tag "`1.`" (or any of the other tags for that matter), it causes the reference to be searched for the letter "l", and if found, then this is added to the reference as one of its contexts.[2]

Fixing the brackets and digits by hand resulted in all 29 contexts being found and matched up with their

[2]This suggests that some programming changes should be made. First, reference anchors consisting of only 2 or fewer lower case letters should not be considered as anchors. Secondly, when searching for anchor words within the reference string, it should match a whole word, not just part of one.

Context Number	Reference Anchors	Consequences
1	[1]-[4]	References 1, 2, and 3 will not match up with this context.
3	171	This should have been [7]. This sentence is not recognized as a context, let alone be matched up with reference literal 7.
4	[s],[9]	Should have been [8], [9]. Picked up as a context but only for reference 9., not for 8..
5	[lo]	Should have been [10]. Was picked up as a context because we are not requiring numerics, but it will not match reference literal 10..
6	[ll], Rosenkrantz and Towsley (121).	Should have been [11], Rosenkrantz and Towsley [12]). Was picked up as a context (because of [ll]) but no reference literal will match with these non-digits.
7	113)	Should be [13]; was not recognized as a context.

Figure 2: Analyzing contexts from PDF source which has been uncompressed and trimmed – typical problems with finding references in uncompressed text.

corresponding reference literals. This improved the Item Accuracy (because all the contexts were found) and the Reference Accuracy (because each reference's contexts were found).

Figure 3 shows the actual results of analyzing the 22 references in this paper, from the precision point of view. Precision is how many of the recognized elements were correct. Recall is the percentage of reference data elements that are determined correctly. Figure 4 shows recall. Of the 22 references, 20 were detected and analyzed (the other two, which appeared in the second column of the paper, were accidentally merged with the fifth reference).

Actually the results weren't all that bad (77.73% Average Reference Accuracy and 88.89% average Item Accuracy). The measurements were lower than they could have been primarily because of the reference-context problem caused by poor scan translation.

The bibliographic data was extracted quite well. Occasionally titles and journal names were scrambled (see Table 1). Categories 3 and 4 of the erroneous journal names are probably impossible to use as a URL lookup mechanism. An example of Class 3 error is:

```
IEEE !I!rans. Inform. Theory...
```

With some further editing, Class 2 journal names probably could be accurately detected and used. A Class 2 example is:

```
Proc. Fifth Data Comm. Symp.,
  Snowbird, Utah, pp
```

Here, only the final "pp." has to be trimmed off. So, 13 of the 20 references, or 65%, could probably be converted into URLs of online copies, if any exist.

Table 1: Number of Correctly Extracted Journal Names

Class 1	No journal name in reference	0
Class 2	Name extracted, extra information	13
Class 3	Name extracted, with scan errors	6
Class 4	Name failed to be extracted	1

6 Collection Level Results

To get a general feeling for how well ACM papers could be processed using our methodology, we went to the journals and proceedings and selected at random one paper from each of the different collections, of which there were 60. It became immediately apparent that PDF files are not all encoded in the same way. Here are the results:

Result	Number
Papers which did not deflate	38
Deflated papers that broke PJ	4
Reference Section not found	4
References not individualized	6
Produced analyzable HTML output	8
Total	60

Papers which did not deflate actually decreased a bit in size when run through the decompression

Precision of Recognized Data

Reference (1)	Bib Data (2)	Real Context (3)	Bogus Context (4)	Precision (2+3)/ (2+3+4)
1	4	1	11	.45
2	3	1	8	.50
3	4	1	7	.71
4	5	1	0	1.00
5	3	7	0	1.00
6	3	2	0	1.00
7	3	1	0	1.00
8	3.5	1	13	.35
9	4	1	0	1.00
10	5	0	13	.38
11	3	0	12	.25
12	4	0	7	.57
13	5	0	14	.36
14	3	0	15	.20
15	3	1	0	1.00
16	3	1	0	1.00
17	3	2	0	1.00
18	3	0	7	.43
19	3	1	0	1.00
20	3	1	0	1.00
21	0	0	0	None
22	0	0	0	found
Totals	70.5 (out of 90)	22 (out of 40)	107	.46

Figure 3: The accuracy with which 22 reference literals were parsed. Bib Data refers to authors, title and year. Real contexts are contexts that indeed contained this reference. Bogus contexts are contexts that did not really belong to this reference. If all the correct BibData and Contexts were found, we would have had 119 elements in all. With the bogus contexts, we add 107 more elements. So, precision for full recall would have been $119/226 = 0.53$, which is pretty disappointing. The actual scraping had a precision of $92.5/199.5 = 0.46$.

Recall of Data Retrieved

reference (1)	number of elements correct (2)	total elements (3)	recall (2)/(3)	Why recall is less than 1.00
1	5	5	1.00	
2	4	4	1.00	
3	5	5	1.00	
4	6	6	1.00	
5	10	14	.71	
6	5	6	.83	
7	4	5	.80	
8	4.5	5	.90	Title has
9	5	5	1.00	B&table
10	5	6	.83	(should be
11	3	4	.75	Bistable)
12	4	5	.80	
13	5	7	.71	
14	3	3	1.00	
15	4	4	1.00	
16	4	5	.80	
17	5	5	1.00	
18	3	5	.60	
19	4	5	.80	
20	4	6	.67	
21	0	4	.00	References
22	0	5	.00	not found
Totals	92.5	119	0.78	

Figure 4: Unless otherwise noted, the less than perfect recall is due to contexts not found because the reference anchor was misspelled (see Figure 1). Column 2 includes both Bib Data and Real Context (see Figure 3 headings). Since the overall recall, or average reference accuracy, is $92.5/119 = 0.78$, we say on average that 17 of the 22 references parsed correctly when computing Item Accuracy.

program. Most likely these were compressed using an algorithm other than "flate". None of these papers could be processed further.

The papers that broke PJ got a class cast exception involving a PjReference object. We will look more into this at some future date. The remaining 18 papers, or 30% of the starting set, produced HTML output. This is not an encouraging result. Although this is a statistically small sample, it does seem to indicate that with current tools we can process only a quarter of the ACM digital library.

7 Other Converters

Although we wrote our own pdf-to-html converter, there are other converters available on the Internet. Many of these simply convert the PDF source into images which are then served up as HTML pages. These are not useful for our purposes.

The `pdftohtml` converter from the Greenstone digital library project (`<http://nzdl.org>`) did work for papers that did not deflate, but the HTML that was produced was rather poor. This code also produces a `.ppm` file for each page, but the images were upside down and backwards.

The pdftohtml converter from Stuttgart, based on Xpdf, is a C program downloadable from <http://www.ra.informatik.uni-stuttgart.de/~gosho/pdftohtml>. It is much faster than Greenstone's, perhaps because it does not generate .ppm files.

Neither of the pdftohtml converters avoided the scan errors gotten from the deflation algorithm. Also both converters missed the second column of references (21 and 22), just as our translator did. They were either appended to the 5th reference, or interleaved with references 5 and 6.

8 Conclusion

Automatic techniques applied to the ACM digital library will only go so far. Analyzing PDF source is really hard. We like the PJ package very much. We must locate more PDF-decompression tools, ones which will handle the remaining formats, such as LZW. The prevalence of scan errors indicates that ACM was wise to use a proprietary, fuzzy matching algorithm to match reference literals against existing bibliographic data.

9 Acknowledgements

This work was funded in part by NSF grant IIS-9907892, "Integrating and Navigating Eprint Archives through Citation-Linking". This project is part of the International Digital Library Initiative.

References

[1] The Open Citation Project. <http:opcit.eprints.org>.

[2] H. Atkins. The ISI Web of Science: Links and electronic journals: How links work today in the Web of Science. *D-Lib Magazine: The Magazine of Digital Library Research*, 5(9), 9. <http://www.dlib.org/dlib/september99/atkins/09atkins.html>.

[3] D. Bergmark. Automatic extraction of reference linking information from online documents. Technical Report TR 2000-1821, Cornell Computer Science Department, Nov. 2000.

[4] D. Bergmark and C. Lagoze. An architecture for automatic reference linking. In *ECDL, September 2001*, Sept. 2001. <http://www.cs.cornell.edu/bergmark/www10.pdf>,<http://link.springer-ny.com/link/service/series/0558/papers/2163/21630115.pdf>.

[5] K. Bollacker, S. Lawrence, and C. Giles. CiteSeer: an autonomous web agent for automatic retrieval and identification of interesting publications. In *Proceedings of the 3rd ACM Conference on Digital Libraries*, pages 116–123, 1998.

[6] K. Bollacker, S. Lawrence, and C. Giles. Discovering relevant scientific literature on the web. *IEEE Intelligent Systems*, 15(2):42–47, 2000.

[7] C. L. Borgman and J. Furner. *Scholarly Communication and Bibliometrics*. 2001.

[8] P. Caplan and W. Arms. Reference linking for journal articles. *D-Lib Magazine: The Magazine of Digital Library Research*, 5(7/8), July/August 1999. <http://www.dlib.org/dlib/july99/caplan/07caplan.html>.

[9] B. Cronin. Bibliometrics and beyond: Some thoughts on webometrics and influmetrics, 2000.

[10] S. Harnad and L. Carr. Integrating, navigating and analyzing open eprint archives through open citation linking (the OpCit project). *Current Science Online*, 79(5), Sept. 2000. <http://www.cogsci.soton.ac.uk/~harnad/Papers/Harnad/harnad00.citation.htm>.

[11] S. Hitchcock, L. Carr, S. Harris, J. Hey, and W. Hall. Citation linking: Improving access to online journals. In R. B. Allen and E. Rasmussen, editors, *Second ACM International Conference on Digital Libraries*, pages 115–122, Philadelphia,PA, July 1997.

[12] S. Lawrence, C. L. Giles, and K. Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999. <http://www.researchindex.com.

[13] I. Pila. CrossRef: The central source for reference linking. <http://www.crossref.org/>.

[14] B. Schatz. Information analysis on the net: The interspace of the 21st century. In *Proc. 1997 (34th) Annual Clinic on Library Applications of Data Processing*, pages 110–126. Graduate School of Library & Information Science, U. of Illinois, Oct. 1995.

[15] J. White. ACM opens portal to computing literature. *Communications of the ACM*, 44(7):14–16, July 2001.