Automatic Extraction of Reference Linking Information from Online Documents

Donna Bergmark*
Cornell Digital Library Research Group

CSTR 2000-1821

Abstract

The Web, with its explosive growth, is becoming an efficient resource for up-to-date information for the scientific researcher. Informal online archives are repositories for technical reports. Proceedings are more and more commonly published on the Web. The collection of online journals is growing. Indeed, a good number of online journals are "born digital". Many researchers simply put their papers up on their own web site. The large volume of online material makes it quite desirable to be able to access cited documents immediately from the citing paper. Implementing this direct access is called "reference linking".

Some reference linking services exist today. A number of commercial publishers, recognizing the significant value-added nature of reference linking, have banded together to form the CrossRef organization. The CrossRef publishers share their metadata, which enables them to interlink their journals. This metadata is not, however, available without a fee to organizations or individuals outside of CrossRef.

The vast majority of online scholarly literature is accompanied by little or no metadata. Since it is desirable to link up this literature as well, the problem of automatically reference linking online scholarly literature in the absence of metadata and author intervention is a problem very much worth considering.

This paper explores this problem in detail, and presents some algorithms for extracting metadata from online texts and linking full-text documents together. The extent to which reference linking of the online literature can be done automatically is therefore the main topic of this paper.

*CNRI/Darpa Grant #2057/57-02 and NSF Grant # IIS-9907892

1 Introduction

Linking documents together seems to be a natural proclivity of scholars. From following the World Book Encyclopedia's "see also's" to the immediate success of Eugene Garfield's Citation Index in the 60's and 70's to the Web today, following links to related information is irresistible. HTML would not be half so popular the language it is today were it not for its support for anchors and links and the HTTP protocol.

Reference linking lies somewhere between Citation Index's static link discovery and the Web's author-inserted links. It finds references in online technical material to other online material and then turns these into anchors by embedding them in a link to an online copy of the cited work, or to a service which can provide a copy of the cited work. A consortium of scholarly publishers, CrossRef, is already doing this for their own journals [2]. According to their promotional literature, CrossRef plans to do a substantial amount of reference linking:

> At the outset, more than three million articles across thousands of journals will be linked through CrossRef, and more than half a million more articles will be linked each year thereafter. Such linking will enhance the efficiency of browsing and reading the primary scientific and scholarly literature. It will enable readers to gain access to logically related articles with one or two clicks – an objective widely accepted among researchers as a natural and necessary part of scientific and scholarly publishing in the digital age. – *www.crossref.org*

By sharing their metadata with each other, the publishers interlink their journals. But this covers only a tiny portion of the online literature. What about scholarly papers that exist online in repositories, archives, people's home pages, vendor sites, and so on? Can these be interlinked as well, with minimal human intervention?

Our project in reference linking is directed towards exploring the extent to which reference linking information can be extracted automatically from online documents, without author or editorial intervention.

This work is part of a larger project, OpCit, which is discovering techniques for linking very-large-scale preprint archives[11]. While OpCit at Southampton focuses on large and fairly regular collections of literature, we at Cornell are focusing on fewer but irregularly formatted papers, with a variety of reference styles. The project as a whole thus is moving toward bringing reference linking value to the scholarly side of the Web [5].

2 Reference Linking Tasks

Reference linking means turning references within an online document into "live references" so that while viewing a scholarly paper or document on your screen, you can follow references in that paper to other network accessible papers and view those as well (in separate windows on your terminal). It is especially attractive to develop this technology because of the increasing number of technical journals and magazines available online[15].

Most commonly, references are found in a late section of an article; this section is often labeled `References`, `Bibliography`, or `List of References`, making it relatively easy for a computer program to locate them. Within the reference section, individual references increasingly include URLs pointing directly to online information. In many cases, references that do not include URLs mention published journal articles that can also be resolved to an online copy. For example, ACM journals can be found in print as well as online, assuming that the reader or the reader's institution is a subscriber to the ACM Digital Library. SFX [23] is a leading contender for resolving references while taking the user's context into consideration. However, it is often the case the URLs must be "discovered". That is also a task of reference linking.

We split reference linking applications into two parts[3]: full text *analysis* and full text *presentation*. To date, we have been working mostly on the analysis part. The presentation of the text, including live links, is another project.

Analysis considers a paper to be composed of three parts: the front matter or header material (title, authors, etc.), the body (containing reference anchors and their contexts), and the reference section (containing the reference strings).[1] Based on this decomposition, we have defined four smaller analysis tasks for each item[2] analyzed:

- **Header Material -**
  - Determining general data about the analyzed item (authors, title, year of publication) by parsing the header material
- **The Body -**
  - Scanning the body of the text for reference anchors (e.g. `[10]`) and collecting the contexts of these reference anchors
- **The Reference Section -**
  - Analyzing the reference strings in the reference section
  - Matching the reference anchors to the tags on the reference strings

These tasks implicitly create two other tasks: determining where header material ends and the body begins; and determining where the body ends and the Reference Section begins. Section 5 of this paper discusses these six (total) tasks in detail, posing them as programming problems along with the algorithmic solutions we use.

3 Document Names

In talking about extracting reference linking information from full text, it is important to be clear about the difference between analyzed items and the *works* cited by those items. Both are writings (or creations), but with an important difference: the analyzed item actually exists online, and we know its current location.

References, on the other hand, are to works which may or may not exist online. They may no longer exist anywhere. The problem is to identify the work just by

[1]We have been deliberately casual about the term "reference" up to this point, but now we distinguish between a tag placed in the paper's text and the actual citation at the end of the paper. The former will called the *reference anchor* and the latter will be the *reference string*.

[2]The word *item* is defined by the library community as an actual copy of a work, while *work* is the abstract notion for a publication of which zero or more copies might actually exist[22]. In our work, we are concerned only with online items.

parsing the reference string. If the item is part of an online journal or is in some repository, then it has a Document Object Identifier[19] consisting of the repository name, a separator, and the unique name of that object within the repository. For example, the DOI of a D-Lib paper looks like `10.1045/december99-miller`. But many online references, e.g. on a researcher's homepage, do not have a DOI. What do we use as a name then? The problem of getting unique DOIs for random archives or repositories is an open question.

In CrossRef, a publisher can use metadata (title, author, etc.) to look up a DOI from the CrossRef database. But this service is not (yet) generally available to reference linking projects such as those at Cornell and University of Southampton. While plans are underway to add SFX to CrossRef, which would make it available to general users who have been granted access to some of these online collections, other works are best represented by a URN synthesized from the work's bibliographic information.

We could have used a single, central, unique integer for document identification, but in the interest of promoting distributed object oriented approaches we preferred to avoid this serial bottleneck. We chose to construct our own URNs by concatenating three strings: the first author's last name (or "*" if unknown), the 4-digit year of publication (or "*" if unknown), and the first 20 characters of the lower-cased title. This becomes a sufficiently precise hash key for looking up a work to see if it has been previously analyzed as an item, or seen before as a reference in another item. Similarly, the project at Southampton uses the year, month, and article number of an arXiv item as its URN. Manufacturing a URN out of bibliographic data gives one a distributable way of making a good key, because with bibliographic data in hand one can compute the key directly, without doing table lookups.

Incomplete URNs can be completed as more items are analyzed. For example, the year might be missing in one reference to a work, but supplied in another.

We synthesize URNs for both items and references, though where DOIs or URLs are available, we keep them as well. Thus a single work will have one synthetic URN, zero or more DOI's, and zero or more URLs. Having a DOI means that the work was analyzed as an item in a repository and its references are available.

4 Preprocessing Online Documents

One problem with automatic reference linking is that not all formats (bitmaps, TeX, PDF, PostScript, etc.) are equally easy to parse. For this reason before an online document is analyzed, the first step is usually to transform the document into a format more susceptible to analysis. The two most common target formats are ASCII and XHTML (the XML version of HTML).

ResearchIndex (formerly CiteSeer)[13] uses a version of **pstotext** that inserts font tags into the document as the document is converted from PostScript/PDF into ASCII.[3] Similar approaches are used in analyzing OCR conversions from bit maps. Summers [21] derives paper segments (such as title and authors) from inspecting the geometric layout of a scanned document. More recently, Caton [7] pointed out that presentation directives can be used to generate tags that help navigate a document. In general, these various formats with their font notations can

[3]The **pstotext** program comes with the GhostScript package[14].

be converted to HTML tags, and then analyzed by our XHTMLAnalyzer software.

The software from Southampton[10], which reference-links PDF files found in the arXiv repository at Los Alamos, uses Acrobat tools to convert PDF into ASCII text prior to analysis. Likewise, [9] discusses the preprocessing of Word documents into a form that can be analyzed. In general, the conversion tools listed in Table 1 are recommended for preprocessing full-text documents into a form that can be analyzed.

Full-text format	Conversion algorithm	Analyzable format	Layout info
ASCII	no conversion	ASCII	none
HTML	Tidy/JTidy	XHTML	HTTP tags
DVI	dvips, pstoascii	ASCII	fonts
PostScript	pstoascii	ASCII	fonts
PDF	pdfps, pstoascii	ASCII	fonts
bitmaps	OCR	ASCII	? depends
Word	save as HTML, Tidy/JTidy	XHTML	HTTP tags

Table 1: Conversion tools to prepare for parsing. See also TOM Conversion Service, `http://tom.cs.cmu.edu/`.

The reference linking project at Cornell has so far dealt only with HTML documents. It turns out that most HTML documents are not well-formed and are therefore difficult to parse. Although HTML parsers exist (see the javax Swing package for example), they are not well documented and do not have much functionality. For that reason, we first convert the HTML into well-formed XML (i.e. XHTML) using the very excellent JTidy[20] package. It cleans up the tags, lowercases them, tries to resolve problems. Only in ambiguous cases, does Tidy give up and output nothing. If XHTML can be obtained, it can be analyzed by an XML parser, of which several good ones exist, including jaxp from Sun and Xerces from the Apache project.

JTidy cannot unambiguously make every HTML document into a parseable XHTML document. Using the april2000 version of JTidy on D-Lib[4] papers, for example, we found that 220 out of the 280 papers, or 79%, could be converted into XHTML with no fatal errors (albeit numerous warnings). For the interested reader, Figure 1 shows example snippets of HTML which could not be tidied. The first one is missing the `a` element tag in front of the `href` attribute; the second one uses an unknown tag, `<it>`, undoubtedly for italic but that is not what HTTP uses; and the third has a malformed `<TD>` element. The fourth line only gets a warning because Tidy can discard the unexpected `</a>`.

We do not store the preprocessed documents. We keep only the information collected during analysis. There may be problems down the line if the online paper is changed and deviates too much from the collected information. But see annotation work at Berkeley [18] for possible solutions to this problem.

[4]D-Lib is an online journal which has been appearing 11 times a year since July 1995.

```
<href="http://www.minitel.fr">http://www.minitel.fr</a>

in <it>Proceedings of the 20th Annual International ACM SIGIR...

<TABLE>... <TD WIDTH=2<BR></TD>  ... </table>

<center><img src="images/book.gif" border"0"></a></center>
```

Figure 1: HTML snippets whose HTTP tags cannot be converted into XML by Tidy, usually because it is not a legal tag. The last line is repaired by Tidy.

5 Extracting Reference Information from On-line Documents

We now turn to a detailed discussion of the reference linking tasks listed in Section 2. Each task is described and problems in carrying out the task are delineated. Where appropriate, some solutions and working algorithms to attack the problems are presented. We assume that the document being processed has been converted into XHTML, as discussed in the previous section.

5.1 Extracting an Item's Metadata

Why is it important to have an item's bibliographic data when analyzing it for reference linking applications? The main reason is that since we are analyzing this item, we have the online location of this item. It is a linkable copy of a work. In order to know what work that is, we need to know the item's bibliographic data, such as title and authors and year of publication. Once we have determined the work of which this item is a copy, then if we come across this work in a reference list in the future, we already know that we have a linkable reference, and we know its location.

Either the item is accompanied by metadata, as is the case with Open Archive items [26] and more recent D-Lib papers, or else it has to be extracted from the text of the paper. We do the latter.

To extract the metadata for an analyzed item, layout clues are necessary. Usually presentation information (such as font changes) is used to determine what the title is. Titles usually occur in a large font, near the beginning of a paper.

Metadata Extraction Algorithm (for XHTML)
set `title1` = value of `<title>` element if there is one Scan for any of the following: `<H1>text</H1>`, `<H2>text</H2>`, `<font size="+3">text</font>`, `<font size="+2">text</font>`, `<font size="5">text</font>` **set** `title2` = "`text`" **if** `title2` is shorter than `title1`, **then** scan for subtitle and append to `title2`

In HTML, one can assume the title is contained in an <H1> or <H2> element, although it happens sometimes that the title is simply set off by a <FONT> element that increases the text size.

Multiline titles can be extracted from HTML documents by reconciling the parsed title with the <title> element if one exists. It is very helpful if the <title> element contains a ':' separating the main title from the subtitle. If there does appear to be more title to be scanned, then look for <h3> or <font size="+2">. In the general case, the title must be assumed to be set off in its own paragraph and/or be terminated by the font reverting to normal size.

Once the title has been located, the authors come next. Determining the authors of an unmarked-up document is particularly difficult. Although it is relatively easy to determine where the author section is, parsing that text for author names is problematic because it is difficult to separate author names from institution names.

However, markup tags do help, plus the presence of commas in the text is a clue. Any tag denotes the end of an author's name; since last names don't come first in the front matter, commas usually denote the end of one author's name. Here is the algorithm we use to parse out the author name strings:

Metadata Extraction Algorithm (cont.)
Rule 1. Always use the first line after the title as an author string. Rule 2. Always use text set off by any of the following as a string of author names: <p> text </p> <center> text </center> text Rule 3. Individual author names are terminated by any tag, such as , or by a comma. Rule 4. The author name section is termination by the first header, such as <h3>.

Finally the date of publication can sometimes be determined from the item's URL or DOI. It is rarely contained in the text of the document itself. Finding the publication year can be very difficult, and warrants further research.

5.2 Locating the Body of the Text

The body of the text starts when there are no more authors listed in the header material. The most effective algorithm is to check scanned text for something that looks like a section heading (but not as big as the main title) and contains the word `Abstract`, `Introduction`, or `Contents`. When found, the format of the header should be remembered for later use, when looking for the Reference Section.

5.3 Finding Reference Anchors in the Text

Once into the body of the text, one needs to locate the reference anchors. Figure 2 lists some of the formats found in D-Lib.

Scanning for strings as shown in the left column of Figure2 is straightforward. Initially each sentence in the text is searched for a '(', '[', or '{'. Differences among them are:

```
[1] or [1,3] or [8-10]
[Bruce and Wayne ]
[Bruce et al.]
(Bruce & Wayne, 1998)
(Bruce,1998)
(Bruce, 1998, Wayne, 1999)
(Bruce et al., 1998)
(CNRI, 1997)
{Digital Library Initiative}
[Bruce, 1996; Wayne, 1999]
See Hakkala (1996)
Bruce and others (1997)
Bruce and Wayne (1998)
```

Figure 2: Some reference anchor formats, as found in D-Lib papers.

- Only the '[' can be followed by digits (unless it is a "lone year")
- Parenthesized references must have a year included in them, in order to distinguish them from parenthesized expressions.

We handle numerical ranges by replacing `[1-3]` with `[1][2][3]`. Comma-ed and semi-colon-ed lists are similarly broken up into individual references; `(Smith, 1998; Jones, 1999)` for example is normalized to `[Smith, 1998][Jones, 1999]`.

This all works quite reliably. One problem is how to handle some authors' penchant for using the reference as a part of speech, e.g. `Caplan and Guenther (1996) explored the difficulties` ... This needs to be parsed into `[Caplan and Guenther, 1996]` in order to match its reference string which could look like:

```
Caplan, Priscilla, and Rebecca Guenther. (1996). Metadata for
Internet Resources: The Dublin Core Metadata Elements Set and ...
```

Here is our algorithm for handling this problem:

Algorithm for References used as Parts of Speech:

if a lone year is seen, e.g. (1996)
then do
Scan backwards from the year to the beginning of the sentence or to an uncapitalized word or strange punctuation, then
do
accept `Namelist = {` `Name [, Name]`$^+$ `"and" Name | Name Etal | Name "and others"}`
end do
output '[' `NameList` ', ' `year` ']'
 	
where:
`Name` is a capital letter followed by small letters, `-`, or `'` and can end in a comma
`Etal = "et al." | "et.  al." | "et.  al"`

This illustrates the problems involved with pulling reference anchors out of the text. For many online journals, there is no "house" reference style, and certainly not for author deposited papers in archives. The good thing is, though, that once the format of the references is determined, it holds throughout the paper. ResearchIndex uses an interesting technique to determine which format is used: it first counts the number of '(' and '[' in the text. Whichever is more frequent is taken as an important hint for finding reference anchors.

We find reference anchors by examining each portion of the body, sentence by sentence. Initially three parsers are run on each sentence. When one returns more hits than the other for the same sentence, that parser, or a variant of it, becomes the only text parser used for the remainder of the paper. For example, if the first reference found is "(Brown and Allen, 1999)" further references will be assumed to match the parenthesized list of authors and year pattern. We have found that 6 grammars are sufficient to find the references in D-Lib articles:

```
SQUARE_BRACKETS_AROUND_NUMERALS
PARENTHESES_AROUND_NAMES_AND_YEAR
SQUARE_BRACKETS_AROUND_ACRONYMS
PARENTHESES_AROUND_COMMAED_NAMES_AND_YEARS
BRACKETS_AROUND_COMMAED_NAMES_AND_YEARS
CURLY_BRACKETS_AROUND_ACRONYMS
```

Once we get to a sentence that contains more than one reference, it becomes a *context*. A given reference might appear in one or more contexts. These references and containing contexts should be saved for later, when they can be matched up with reference strings (see Section 5.6).

Find Contexts Algorithm (see below)

Given: a sentence, S
If there is a designated parser P **do:**
    Let `refsInText` = set of references in S using P
    If `refsInText` $\neq \emptyset$, save S as a context
end
Else do:
    Let `u` = set of references in S of form [...]
    Let `v` = set of references in S of form (... year)
    Let `w` = set of references in S of form `{...}`
    `refsInText` = max of `u`, `v`, `w`
    If `refsInText` $\neq \emptyset$:
        Save S as a context
        Set `P` = the parser that produced `refsInText`
end

Problems with finding contexts include:

Premature sentence termination - Assuming "." ends sentences, then false endings due to "et al." or "etc." or "44.4" could cause truncated contexts, if not treated as special cases. Fortunately, it is relatively easy to check for these.
Context and Reference Anchor Disassociation - A few authors will put their reference anchors outside of the sentence to which it relates, e.g.

```
In the past, this has been a big deal.[8]  However, no more.
```

The anchor clearly belongs to the first sentence, but will be analyzed as part of the second. The best algorithm here is note that the terminating period of the first sentence is followed by a typical anchor delimiter (here, "`[`") and save the sentence just in case. Then if analysis of the second sentence reveals the presence of an anchor at the beginning, and it is followed by a capital letter, then the anchor can be put with the first sentence instead of the second.

False Contexts - These are where a text fragment, for example `[1907]` is spuriously parsed to be a reference anchor. No harm is done unless it happens to match one of the reference strings, in which case a false context will appear for that reference.

5.4 Locating the List of References

Current reference linking data extraction tools all seem to use the same technique for locating the references section: look for a heading like `References`. The Southampton software that uses deciter (we'll call it DLS Version 1999) scans references from the end of the paper forward and stops when the reference section heading is found. This is very efficient in terms of scan time. We would do the same, but we wanted to scan the paper from the top in order to pick up the contexts. Otherwise, the algorithm is the same: recognize a section header that says something similar to "References". Here is the list of headings found while analyzing D-Lib papers:

```
References
Bibliography
Notes and References
Note and References
<section#.> References
```

Note that it is important to examine only section *headings* for the bibliography keyword; finding `References` in a table of contents doesn't count. Additional complications involved with locating the Reference Section are:

No Reference Section - The references appear as footnotes rather than being collected at the end (a format popular in some circles). In this case, the reference strings have to be located and collected while scanning the body of the text. When the end of the paper is reached, the collected reference strings are treated just as though they had all come at the end of the paper in a bibliography. References, in either case, are assigned an ordinal value depending on the relative order in which the reference string was encountered, *vis a vis* other reference strings.

References are in a Different File - An unsolved problem is what to do with references that are in a separate file from the repository item. For example, some HTML documents contain a link to a separate page that holds the reference strings. Determining the actual set of files that comprise a document is an open, important problem

Reference Section Loses its Markup - Since the reference section is located by examining section headers, it is crucial that the section header be there. For example, JTidy could remove the `<H3>` markup due to other syntax problems. A related problem is when the first section header after the front matter is inappropriately tagged.

5.5 Parsing the Reference Strings

This task basically involves decomposing tagged reference strings into separate elements, extracting key bibliographic data, constructing a temporary URN for the work being described (see Section 3), and then using the tag on the reference string to retrieve the context(s).

To parse the reference strings, we first remove the tag (e.g. the `10.`), and pass the remainder of the reference string to Southampton's `deciter` routine. This routine outputs the metadata gleaned from the reference string, encoded in XML. Our software then does a DOM parse of this XML to extract the information of interest, synthesize a URN, and then use the URN to look up the work in the database of previously encountered works. A work corresponding to a reference string is added to the database even if it is not linkable, since someday it might become linkable. In the long run, one would really want to use this metadata to look up the work on a name server or from the CrossRef database.

The DLS deciter routine and the ResearchIndex's `adddoc` Perl module both contain many clever techniques for pulling apart a reference string, but parsing problems abound. One problem is that the reference string might not even name a work. It might be a "note". In general, it is impossible to tell a reference from a note, since both have tags and either might involve a URL. Confusing a note for a reference string results in very strange bibliographic data.

One useful clue is that "notes" generally do not contain a year. Ignoring dateless reference strings is one strategy that works for most existing documents. A quick check of 20 D-Lib papers that contained a total of 258 notes and references showed that there were only 31 false hits using this approach. In other words, most of the references (92%) had a year, and most of the notes (67%) had no year. The false hits come from web site pages accompanied by the date at which they were seen, books that are in press with no year given, or journal articles that use volume number in lieu of year.

Another problem is that authors' names are spelled in many different ways. We have our own code to parse authors' names in the front matter, but use Southampton's `deciter` to parse author names in the reference strings. The parsing technique we use for front matter was borrowed from the `Text::BibTex::Name` PERL module (thanks to Gregory Ward for his writeup in CPAN). This technique assumes every name consists of 4 parts: first name plus initials; the "von" part which is lower case and included in a list of names like "de", "van", "von", and "le"; the last name; and an honorific (Jr., III, etc).

Ward's Author Name algorithm

Let `Aname` be a single author name string
if number of commas in `Aname` $= 0$, parse `Aname` into
 	
`First [von] Last [jr]`
 	
if number of commas in `Aname` $= 1$, parse `Aname` into
 	
`Last [Jr], First [von] | First [von] Last, Jr`
 	
if number of commas in `Aname` ≥ 2, parse `Aname` into
 	
`Last, Jr, First [von]`
 	
Note that only two of the four rules apply to front matter, because in front matter last names do not come first.

The `deciter` concentrates on separating the string of author names from the rest of the reference string, and then splits it into separate names at the point of outputting the XML. First and middle names are also replaced by initials at this point. It works pretty well when the names are in a somewhat standard format, like `Ballard, J.G. and G. Nagy.` but is thrown off by the less standard `J.G. Ballard, G. Nagy.`.

Other parsing difficulties occur. Some papers put the title first in the reference string, and then the authors, e.g.

```
[Barley, 1998] Title of a paper about barley.  Jones, Jim.
```

Since this order of title and author is so rare, we do not attempt to pick out the author names in this case. Sometimes there are no authors at all, and then the title could be mangled into author names.

A final note on parsing reference strings: existing tools find it best not to parse them left to right, but to first isolate the year (if present) and the page range (if present). After the tag is stripped off, the author string should be scanned next because almost all reference strings start with the names of the authors. The author string is terminated by the title (`"` or `‘`) and/or are ended by a `.` which does not follow a single capital letter or *et al.* The title runs until the terminal punctuation or the year, page range, or URL.

5.6 Matching Anchors to Reference Strings

An attractive feature of the ResearchIndex is its ability to show citations in context. This is the main reason for doing the task described in section 5.3, parsing for references and saving the sentences that contain them. Looking up the context involves matching reference anchors to reference tags.

Very often anchors and tags match, as in `[10]` and `10.`; however, it is necessary also to handle cases where they almost, but not quite, match. It is not that unusual to comes across this situation:

Reference in the text:	**Tag on the reference string:**
`[Bordon and Locks, 1998]`	`Bordon, Fred and Goldie Locks.`

Our context-matching algorithm supports both exact and approximate matching:

Context-matching algorithm:
Let `refString` = the reference string for whose contexts we are searching `tag` = the tag on the reference string (or null if there is none) $\texttt{contextRefs}_j[\ ]$ = fixed size list of references contained in context sentence j, $0 \leq j < \mathrm{NC}$ `NC` = number of context sentences found while scanning the body of the text **Find** contexts for this reference

```
contexts = {} (empty set)
currentValue = 0
for j=0 to NC-1 do
   for each ref r in contextRefsj[] do:
      if tag == r (exact match)
         if currentValue < 2
            contexts = {j}; currentValue=2
      else if r is in refString somewhere
         if currentValue <= 1
            add j to contexts; currentValue=1
```

Our approximate matching algorithm first checks for an exact match between a reference and the tag; if successful, keep this context. Otherwise do an approximate match: for each significant word in the reference (e.g. `(Bruce & Wayne, 1998)` would have three words), see if each word is in the `refString` somewhere, matching left to right.

Another aspect of our algorithm is the "quality index". This has three values:

```
0 - no match yet
1 - approximate match was found
2 - exact match was found
```

The index can only increase. This helps prevent picking up some bogus contexts. In other words, if we have seen an exact match between one anchor and a tag, we do not accept approximate matches for the other contexts and references.

Another type of problem occurs when the tags are actually part of the bibliographic data (see Figure 3). In this case, it is best to say that the reference tag is null, and the context will be found by having each one of its words (tokens) found in the reference string, going left to right. That is, each token `Bergmark`, `Lagoze`, and `1999` are found in this reference string, so we have located one context for this reference.

Finally reference strings might have tags that are generated by the browser:

```
<ol><li> reference string </li>...</ol>
```

This case is handled by noting that a list is being generated and simply maintain a separate numeric reference counter.

```
Bergmark, Donna and Carl Lagoze. 1999.  "This is
the title of some work," in Some Journal.

[Bergmark and Lagoze 1999]
```

Figure 3: Example of a null tag. The first two lines are the reference string, while the next line is how the reference appears in the text. The reference string is not tagged, but the anchor can still be associated with the string.

6 Linkable References

Having picked out the references and matched them to their contexts, a key question remains: making the anchors into live links. Another term for this problem is *link resolution*. Link resolution includes finding online locations for the referenced paper and retrieving it.

Determining whether or not a reference is linkable can be done at analysis time or at presentation time. The former is *static* link resolution, the latter *dynamic*.

The ResearchIndex and Open Journal [12] projects do static link resolution, building citation databases with canonical representation of sources and targets. This is most efficient, because the databases can be updated overnight, online retrieval does not have to wait for all of link resolution to occur, and each new paper need have its references processed only once. Static link resolution is usually done in conjunction with item analysis.

The alternative is to resolve the links on the fly, as is done in the SFX [23, 24, 25] system. This has the advantage that if a paper has moved its location, it can still be retrieved. Another advantage is that if there are several ways to get a copy of the paper, this will pick the appropriate one based on the user's location (or *context*). Dynamic resolution is usually done at presentation time.

Our system, which separates the analysis from the presentation, can do link resolution dynamically by analyzing a paper "on-the-fly" or statically, collecting the information once for later reuse. An object-oriented API [3] makes this possible by creating a surrogate for each document. The first time a surrogate is instantiated for a document, the surrogate parses it and collects reference linking information. This information becomes part of the data encapsulated by the surrogate. The encapsulated data is distributed through the API for use by client programs.

Link resolution is orthogonal to our work in reference link analysis. Dynamic link resolution can occur whether the analysis is static or dynamic, since analysis precedes the resolution. Static link resolution, of course, implies batch or offline analysis. One can use the reference linking API to build up a collection of reference linking information in the form of Item surrogates.

In either case, the product will be more or less helpful to the researcher depending on the integrity of the links. Inadequate parsing of reference strings can render an online reference unlinkable, just as could mistakes in hand-prepared metadata. While links inserted into an online paper by the author(s) can generally be relied on, especially if the paper was published recently[4], references with missing URLs remain a problem to be solved, and finding the right links depends in large part on how accurately the reference strings are interpreted by the reference string parser.

Like OpCit's work with arXiv, the best way to discover linkable references is to process lots of items; the more items you process, the more locations you know.

Once analysis is complete, certain reference anchors embedded in the body of the text can be made into anchors for live links. The references should be accompanied with sufficient data to support a variety of linking behaviors. At minimum, this data should include one or more locators for the item referenced, assuming the referenced item is linkable (i.e. is on the network, is locatable, and the user has access permissions to it).

Our approach is to enclose all references, linkable or not, in a custom element which can be translated into JavaScript, an XLink[27] or an OpenURL[8]. An example of such a tag can be seen in Figure 4. The example shows a `<reflink>` element containing an anchor, `[5]`. Attributes of the element include zero or more URLs where a copy of the work being cited can be found. Sufficient metadata is included in the tag that additional online copies might be found at a later date.

```
<reflink ord="5" author="last-name-of-first-author"
title="title of this work"
year="1999"
url="http://www.some.org/filename">[5]</reflink>
```

Figure 4: Example of a custom element tag for references

7 Citations

Citations are the inverse of references. If B is one of A's references, then A is one of B's citations (think of the Science Citation Index).

Citations are not necessarily a part of reference linking, but once reference data extraction has been implemented, it is tempting to move on to providing citation services. Once an archive item has been analyzed, it becomes a citation to every work in its reference section. This fact can be maintained in a citation database. Such a citation database is being developed at Southampton for arXiv. Our software is also set up to allow for citation information to be collected.

Clearly, a work may accumulate a number of citations over time, as new items are analyzed which refer to this work. Being able to look up what citations a work has is the basis of the Citation Index, a popular resource in the Science community.

An example should illustrate the problem. Suppose we have analyzed item A, and from its bibliographic data we have determined that A corresponds to the work W_A. Suppose that A contains a reference to work B (recall that all references are works because reference strings contain *only* bibliographic data.) Call this cited work W_B. Now suppose at some later time we come across an archive item X, and from its bibliographic data we determine that it actually corresponds to work W_B. That is, it turns out that item X is a copy of a work that has been cited by A. At this point, our citation service can tell the person who is looking at item X that one of its citations is work W_A, a copy of which can be found at the address for item A.

The main problem that comes up here is the equality relationship on works. In order to collect all of the citations to work W together in one place, one must know whether given work W_1 and work W_2 are really the same work? Yes, if they have the same title, the same year, same journal, same pages and the same authors. For online literature, we dispense with journal and pages, so we are left with the title, year, and authors. Note that this implies also that we have an equality function for year, title, and author names. Year is pretty easy to match, and titles match if they are normalized first (i.e. lower-cased, say), but authors can be tricky.

The document names discussed in Section 3 are key to tracking this information of what cites what. Hence accurate reference parsing, which leads to better URNs, enables good citation services.

8 Results

For automatic extraction of reference linking information, we thought that the analysis would have to be at least 80% accurate in order to allow any interesting reference linking applications to be build on this software. So, we looked at how good we were at extracting bibliographic data. The correctness was determined by human review of the results. To gauge the performance of the algorithms described in this paper, we present two graphs based on parsing a random set of 70 D-Lib papers, four of which failed to convert to XML. Figure 5 plots the Item Accuracies of the remaining 66 papers, and Figure 6 plots some typical Reference Accuracies.

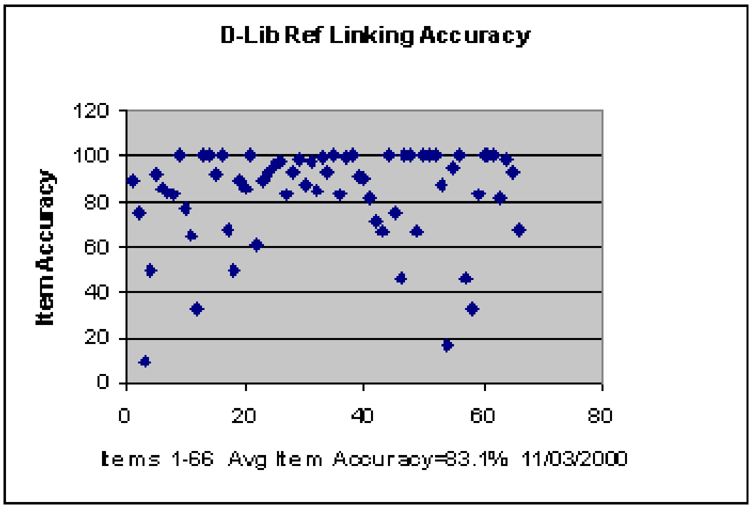

Figure 5: Item Accuracies for a set of 66 D-Lib papers

For each item analyzed, the item accuracy metric is the number of elements parsed correctly divided by the total number of elements in the item. Specifically, the elements used are: the item's title, the item's authors (each author counts as one element), the item's year of publication, the reference contexts (each context in the document counts as one element) and the average reference accuracy times the number of references. The reference accuracy for one reference string is the per centage of its elements that are correctly parsed. These elements include: title,

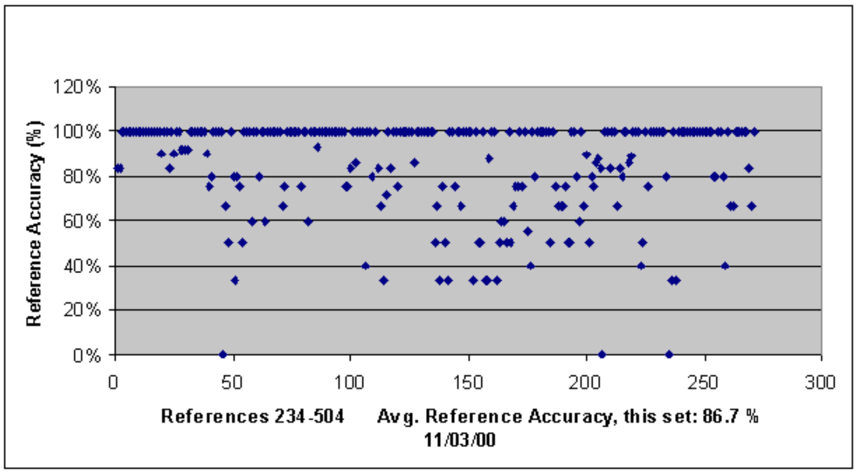

Figure 6: Reference Accuracies for a set of 66 D-Lib papers, batch 2.

each author, year, contexts, and URL if present.

Perfect automatic extraction of reference linking information is not impossible.

9 Related Work

The work reported here on automatic reference linking is strongly related to automatic librarianship, recently proposed by Arms [1] as a way to control the costs of research libraries. While not directly related to library work, this project has much in common with automated library procedures.

ResearchIndex is arguably the most well-known project having to do with reference linking and scanning online documents. It does not actually do reference linking; it is more an automated citation index. But that said, the algorithms, techniques, and Perl software are directly useful for reference linking projects, where the source documents are PDF, Postscript, or HTML. This software is publically available.

Sergei Brin ([6]) has done interesting work on automatic extraction of bibliographic information from the Web. Brin relies on incrementally refining grammar definitions, similar to what we do when parsing text for reference anchors.

And of course, there is related work going on at Cornell. SFX, an advanced reference linking system that takes the user's context into account, is being developed by Van de Sompel, now a member of the Digital Library Research Group at Cornell. There are many ways in which our reference linking software is complementary to SFX.

For an overall view of what is happening about reference linking, refer to the report of last summer's NISO/DLF/CrossRef meeting[16].

10 Conclusions

Automatic extraction of bibliographic data and reference linking information from the online literature is hard, but not impossible. This paper has explicitly laid out six main problem areas, and proposed solutions to each. Results from using prototype software based on these tools and algorithms were presented.

The prototype software might be used in several ways in addition to reference linking. It can quickly check whether every reference string in the reference section was actually cited somewhere in the paper. Another use for this software would be to assist in self-submission of papers and preprints to the Open Archives. Using this API, one could not only harvest metadata for archive items but also reference metadata. Both use the Dublin Core Element Set for encoding this metadata.

But the almost-achievable over-arching goal is to automatically link online papers together. Perfection is not really needed; if only one online copy of a desired reference can be found and linked to, then that is likely all that the user needs. The next question to be tackled is how to constructively obtain these online locations, when the URL has not been included in the reference string nor has an item for that reference been analyzed.

11 Acknowledgment

The comments of Steve Hitchcock during the preparation of this paper were very helpful. I also appreciated the clarity of Les Carr's code, used to solve some of the parsing problems described in this paper. Herbert Von de Sompel also made a number of useful suggestions regarding the content and organization of this paper. Finally, we are grateful for support from CNRI/DARPA and from NSF.

References

[1] W. Arms. Automated digital libraries: How effectviely can computers be used for the skill tasks of professional librarianship. *D-Lib Magazine: The Magazine of Digital Library Research* <http://www.dlib.org>, July 2000.

[2] H. Atkins, C. Lyons, H. Ratner, C. Risher, C. Shillum, D. Sidman, and A. Stevens. Refererence linking with DOIs: A case study. *D-Lib Magazine: The Magazine of Digital Library Research* <http://www.dlib.org, 6(2), February 2000. <http://www.dlib.org/dlib/february00/02risher.html>.

[3] W. Arms D. Bergmark and Carl Lagoze. An architecture for reference linking. Technical report, Cornell University, Digital Library Research Group, October 2000. TR 2000-1820.

[4] D. Bergmark. Link accessibility in electronic journal articles. Technical Report TR 2000-1793, Cornell Computer Science Department, March 2000. <http://www.cs.cornell.edu/bergmark/LinkAnalysis.ps>.

[5] D. Bergmark and C. Lagoze. Reference linking the web's scholarly papers. Sujbmitted to *World Wide Web 10, May 2001*, May 2001. `<http://www.cs.cornell.edu/bergmark/www10.pdf>`.

[6] Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at EDBT '98*, 1998. Available online at <http://www-db.stanford.edu/ sergey/extract.ps>.

[7] Paul Caton. Markup's current inbalance. In *Extreme Markup Languages 2000*. Graphic Communications Association, 2000.

[8] Herbert Van de Sompel, Patrick Hochstenbach, and Oren Beit-Arie. OpenUrl syntax description, May 2000. <http://sfx1.exlibris-usa.com/OpenURL/openurl.ht ml>.

[9] Irina Golfman. Structured content out of Microsoft Word. In *Extreme Markup Languages 2000*, Montreal, Quebec, August 2000.

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

[11] S. Hitchcock, L. Carr, Z. Jiao, D. Bergmark, W. Hall, C. Lagoze, and S. Harnad. Developing services for open eprint archives: globalisation, integration and the impact of links. In *5th ACM Conference on Digital Libraries, San Antonio, Texas, June 2 - June 7, 2000.*

[12] Steve Hitchcock, Les Carr, Wendy Hall, Stephen Harris, S. Probets, D. Evans, and D. Brailsford. Linking electronic journals: Lessons from the Open Journal project. *D-Lib Magazine: The Magazine of Digital Library Research* <http://www.dlib.org>, December 1998.

[13] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999. <http://www.researchindex.com>.

[14] Tomas Merz. GhostScript user manual. Online version of the manual can be found at <ftp://ftp.cs.wisc.edu/ghost/gs5man_e.pdf>.

[15] Bernard M. E. Moret. ACM's Journal of Experimental Algorithmics: Bridging the gap between theory and practice. *The Journal of Electronic Publishing*, 3(1), September 1997. <http://www.press.umich.edu/jep/03-01/JEA.html>.

[16] NISO. NISO/DLF/CrossRef workshop on localization in reference linking:meeting report, July 2000. Available online at `http://www.niso.org/CNRI-mtg.html`.

[17] Norman Paskin. E-citations: actionable identifiers and scholarly referencing, 1999. <http://www.doi.org/citations.pdf>.

[18] T. A. Phelps and R. Wilensky. Multivalent documents. *Communications of the ACM* , 43(6), June 2000.

[19] Andy Powell. Resolving DOI based URNs using Squid. *D-Lib Magazine: The Magazine of Digital Library Research* <http://www.dlib.org>, June 1998.

[20] Andy Quick. Java HTML tidy. <http://www3.sympatico.ca/ac.quick/jtidy.html>.

[21] Kristen Summers. Near-wordless document structure classification. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR '95)*, pages 462–465, Montréal, August 1995. A copy of this paper can be found online at <http://www.cs.cornell.edu/Info/People/summers/classify.html>.

[22] Elaine Svenonius. *The Intellectual Foundation of Information Organization*. M.I.T. Press, 2000.

[23] Herbert Van de Sompel and Patrick Hochstenbach. Reference linking in a hybrid libary environment, part 2: SFX, a generic linking solution. *D-Lib Magazine: The Magazine of Digital Library Research* <http://www.dlib.org>, 5(4), April 1999.

[24] Herbert Van de Sompel and Patrick Hochstenbach. Reference linking in a hybrid libary environment, part 1: Frameworks for linking. *D-Lib Magazine: The Magazine of Digital Library Research* <http://www.dlib.org>, 5(4), April 1999.

[25] Herbert Van de Sompel and Patrick Hochstenbach. Reference linking in a hybrid libary environment, part 3: Generalizing the SFX solution in the sfx@ghent & sfx@lanl experiment. *D-Lib Magazine: The Magazine of Digital Library Research* <http://www.dlib.org>, 5(10), October 1999.

[26] Herbert Van de Sompel and Carl Lagoze. The Santa Fe Convention of the Open Archives Initiative. *D-Lib Magazine: The Magazine of Digital Library Research* <http://www.dlib.org>, 6(2), February 2000.

[27] W3C. XML linking language (XLink) version 1.0, July 2000. <http://www.w3.org/TR/xlink/>.