FireCite: Lightweight real-time reference string extraction from webpages

Ching Hoi Andy Hong **Jesse Prabawa Gozali** **Min-Yen Kan**
School of Computing
National University of Singapore
{hongchin,jprabawa,kanmy}@comp.nus.edu.sg

Abstract

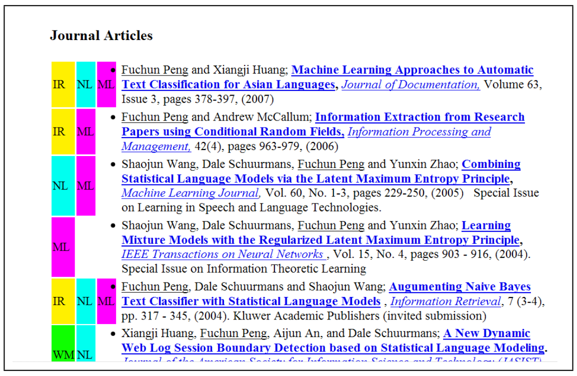
We present FireCite, a Mozilla Firefox browser extension that helps scholars assess and manage scholarly references on the web by automatically detecting and parsing such reference strings in real-time. FireCite has two main components: 1) a reference string recognizer that has a high recall of 96%, and 2) a reference string parser that can process HTML web pages with an overall F_1 of .878 and plain-text reference strings with an overall F_1 of .97. In our preliminary evaluation, we presented our FireCite prototype to four academics in separate unstructured interviews. Their positive feedback gives evidence to the desirability of FireCite's citation management capabilities.

1 Introduction

On the Web, many web pages like researchers' or conference homepages contain references to academic papers much like citations in a bibliography. These references do not always follow a specific reference style. Usually, they make use of HTML formatting to differentiate fields and emphasize keywords. For example in Figure 1, paper titles are displayed in bold.

Depending on personal preference and habit, references found on the Web may be processed in various ways. This process however, can possibly be quite a long chain of events:

1. A researcher finds a PDF copy of the paper and downloads it.
2. He reads the abstract of the paper, then decides to read the rest of it.
3. He prints out the paper and reads it, making annotations along the margin as he reads.
4. He produces a BibTeX entry for the paper.



Journal Articles

IR	NL	ML	• Fuchun Peng and Xiangji Huang; **Machine Learning Approaches to Automatic Text Classification for Asian Languages**, *Journal of Documentation,* Volume 63, Issue 3, pages 378-397, (2007)
IR	ML		• Fuchun Peng and Andrew McCallum; **Information Extraction from Research Papers using Conditional Random Fields,** *Information Processing and Management,* 42(4), pages 963-979, (2006)
NL	ML		• Shaojun Wang, Dale Schuurmans, Fuchun Peng and Yunxin Zhao; **Combining Statistical Language Models via the Latent Maximum Entropy Principle**, *Machine Learning Journal,* Vol. 60, No. 1-3, pages 229-250, (2005) Special Issue on Learning in Speech and Language Technologies.
ML			• Shaojun Wang, Dale Schuurmans, Fuchun Peng and Yunxin Zhao; **Learning Mixture Models with the Regularized Latent Maximum Entropy Principle**, *IEEE Transactions on Neural Networks* , Vol. 15, No. 4, pages 903 - 916, (2004). Special Issue on Information Theoretic Learning
IR	NL	ML	• Fuchun Peng, Dale Schuurmans and Shaojun Wang; **Augumenting Naive Bayes Text Classifier with Statistical Language Models** , *Information Retrieval*, 7 (3-4), pp. 317 - 345, (2004). Kluwer Academic Publishers (invited submission)
WM	NL		• Xiangji Huang, Fuchun Peng, Aijun An, and Dale Schuurmans; **A New Dynamic Web Log Session Boundary Detection based on Statistical Language Modeling**.

Figure 1: A web page with a list of references. Paper titles are displayed in **bold**.

5. He cites the paper in his own work.

This process is too time-consuming for researchers to do for each reference, one at a time. One solution is to collect all the references of interest first. These references can then be processed at a later time. Bibliographic Management Applications (BMAs) do exactly this by allowing the researcher to record interesting references for later use. Alternatively, the references can be recorded manually on paper or in a text file. The paper for each reference can also be printed and organized physically in folders or piles.

Each method has its own disadvantages. Using notebooks, text files or printouts imposes considerable cognitive load on the researcher especially when hundreds of references need to be managed. BMAs seek to relieve researchers from this problem, but are often too complicated to use and maintain. A popular BMA, EndNote, for example, retrieves metadata from online library catalogues and databases, but experience is necessary to know which database or catalogue to search. Considerable time can be lost searching for a computer science paper in a medical database. An automatic, yet lightweight solution is needed.

Since the references are found on the Web, the most suitable location for a BMA is within the web browser itself. In this paper, we propose FireCite[1], a Firefox browser extension which embodies this idea. FireCite 1) automatically recognizes references on web pages, 2) parses these references into title, authors, and date fields, 3) allows the researcher to save these references for later use, and 4) allows a local PDF copy of the paper to be saved for each reference.

At its core, FireCite consists of a reference string recognizer and a reference string parser with accuracies comparable to other systems. Unlike these systems however, as a browser extension, FireCite needs to be fast and lightweight. Bloated extensions can cause the browser's memory footprint to grow significantly, lowering overall performance. An extension must also perform its operations fast. Otherwise, it will detract users from their primary task with the browser. Nah (2004) suggests latencies should be kept within two seconds.

In the next section, we review related work. We then discuss reference string recognition, followed by parsing in Section 3. After component evaluations, we conclude by discussing the user interface of FireCite.

2 Related Work

Recognizing and parsing reference strings has been a task tackled by many, as it is a necessary task in modern digital libraries.

Past work has dealt primarily with clean data, where reference strings are already delimited (e.g., in the References or Bibliography section of a scholarly work). Many works consider both reference string recognition and reference string parsing as a single combined problem. With regards to the task, IEPAD (Chang et al., 2003) looks for patterns among the HTML tags, while (Zhai and Liu, 2005) looks for patterns among the presentation features of the web page. A machine learning approach using Conditional Random Fields is also discussed in a few works (Xin et al., 2008; Zhu et al., 2006).

CRE (Yang et al., 2008) is an automatic reference string recognizer that works on publication list pages. Given such a page, CRE identifies individual reference strings by looking for contiguous common style patterns. The system is based on the authors' two observations: 1) 'reference string records are usually presented in one or more contiguous regions', and 2) 'reference string records are usually presented by using similar tag sequences and organized under a common parent node'. Therefore, the system examines the DOM[2] tree of the web page and identifies adjacent subtrees that are similar. The system then removes subtrees that are unlikely to be reference strings, by comparing their word count against a database of reference strings' word counts. The authors report an F_1 of around 90% for pages where reference strings make up at least 80% of the text on the page, and an F_1 of at least 70% when reference strings make up at least 30% of the page.

Of note is that their testing dataset consists solely of computer science researchers' homepages and publication list pages. There is no indication of how their system will perform for other types of web pages. Although there are many published works on the extraction of semi-structured data from web pages, very few of them deal directly with the issue of reference string extraction. Also, none of the works deal directly with the issue of web pages that do not contain any relevant data. In FireCite's case, this is an important issue to consider, because false positives will be parsed, and as stated previously, almost all web pages will have elements that are not part of any reference string.

As for reference string parsing, the field of Information Extraction (IE) has treated this task as one of its sample applications. As such, many different IE approaches involving different supervised classifiers have been tried.

Such classification methods require a gold standard corpus to train on. The CORA Information Extraction dataset, introduced in (Seymore et al., 1999) consists of a corpus of 500 classified reference strings extracted from computer science research papers, is used as training data. The CORA dataset is annotated with thirteen fields, including *author*, *title* and *date*.

As for classification approaches, (Hetzner, 2008; Seymore et al., 1999) and AutoBib (Geng and Yang, 2004) makes use of Hidden Markov Models (HMM), while ParsCit (Councill et al., 2008) and (Peng and McCallum, 2004) make use of Conditional Random Fields (CRF).

ParsCit's reference string parsing system makes use of CRF to learn a model that can apply meta-

[1]The latest version of the extension is at: https://addons.mozilla.org/en-US/firefox/addon/10766/

[2]Document Object Model. http://www.w3.org/DOM/

data labels to individual word tokens of a reference string. ParsCit's labeling model consists of 7 lexical features (features that make use of the meaning/category of a word, such as whether the word is a place, a month, or a first name) and 16 local and contextual features (features that makes use of formatting information about the current and neighbouring tokens, such as whether the word is in all caps). Its lexical features require the use of an extensive dictionary of names, places, publishers and months. ParsCit achieves an overall field-level F_1 of .94.

Another competitive method, FLUX-CiM (Cortez et al., 2007) also parses plain-text reference strings, based on a knowledge base of reference strings. Initially, labels are assigned to tokens based on the (label, token) pair's likelihood of appearance in the knowlege base. For tokens that do not occur in the knowledge base, a binding step is used to associate them with neighbouring tokens that have already been labelled. The authors report a very high token-level accuracy in terms of F_1 of 98.4% for reference strings in the Computer Science (CS) domain, and 97.4% for reference strings in the Health Sciences domain.

A key difference from other parsing methods is that tokens in FLUX-CiM are strings delimited by punctuation rather than single words (see an example in Figure 2). This comes from an observation by the authors that "in general, in a reference string, every field value is bounded by a delimiter, but not all delimiters bound a field."

$\boxed{\text{Atlas}}$, $\boxed{\text{L}}$., $\boxed{\text{and S}}$. $\boxed{\text{Shamma}}$,
"$\boxed{\text{Joint Acoustic and Modulation Frequency}}$,"
$\boxed{\text{EURASIP JASP}}$, $\boxed{\text{2003}}$.

Figure 2: A tokenised reference string. Each box contains one token.

While both ParsCit and FLUX-CiM have high levels of performance, they are not suitable for our use for two reasons:

- Both systems are large. ParsCit's classifier model plus dictionaries add up to about 10MB. FLUX-CiM requires a database of 3000 reference strings for each knowledge domain, for best performance. Databases of this size will take a significant amount of time to load and to access, negatively impacting the user experience.
- Both systems are not designed to handle web reference strings. Neither system is able to correctly parse a reference string such as the one shown in Figure 3 due to its lack of punctuation and the misleading tokens that resemble publication dates.

Doe, J. 2000 **1942-1945: World War Two and its effects on economy and technology.** Generic Publisher. Generic Country.

Figure 3: A reference string that FLUX-CiM and ParsCit cannot parse correctly.

3 Methodology

FireCite performs its task of reference extraction in two logically separate stages: recognition and parsing. Reference string recognition locates and delimits the start and end of reference strings on a web page, while parsing delimits the internal fields within a recognized reference.

3.1 Recognition

Reference recognition itself can be logically segmented into two tasks: deciding whether references could occur on a page; and if so, delimiting the individual reference strings. We build a rough filter for the first task, and solve the second task using a three stage heuristic cascade.

Algorithm 1 Reference recognition.

```
1: Exclude pages based on URL and absence of keywords
2: Split token stream into a set $S$ of (non-overlapping) sequences, where each sequence
   contains at most one reference string, and no reference string is split across two
   token sequences.
3: Select sequences likely to be reference strings, forming a set $S'$ which is parsed
   into a set of reference strings $C$.
4: Remove sequences with nonsensical parse results from the set of reference strings $C$.
```

We now detail these stages.

Stage 1 immediately discards webpages that do not meet three criteria from subsequent automatic processing. For a page to be automatically processed by subsequent recognition and parsing phases, FireCite requires that the webpage:

- Is from a .edu, .org, or .ac domain. Domains with country identifiers, such as www.monash.edu.au, are also accepted;

- Contains one or more of the words 'Publications', 'Readings', 'Citations', 'Papers', and 'References'.

- Contains one or more of the words 'Conference', 'Academic', 'Journal', and 'Research'.

The included domains include web pages from academic institutions, digital libraries such as CiteseerX [3] and ACM Portal [4], and online encyclopedias such as Wikipedia [5] – basically, web pages where reference strings are likely to be found. The keywords serve to further filter away pages unlikely to contain lists of reference strings, by requiring words that are likely to appear in the headings of such lists.

Stage 1 runs very quickly and filters most non-scholarly web pages away from the subsequent, more expensive processing. This is crucial in improving the extension's efficiency, and ensuring that the extension does not incur significant latency for normal browsing activity.

Stage 2 splits the web page text into distinct chunks. In plain-text documents, we differentiate chunks by the use of blank lines. In HTML web pages, we use formatting tags: `<p>` and `<br>`. Other tags might also indicate a fresh chunk within ordered (`<ol>`) and unordered (`<ul>`) lists, list items are marked by the `<li>` tag. A horizontal rule (`<hr>`) is used to separate sections in the web page. Stage 2 makes use of all these HTML tags to split the web page text into distinct, non-overlapping sequences.

Stage 3 removes sequences that are unlikely to be reference strings, based on their length. Sequences that are too long or short are removed (i.e., with word length $5 < wl < 64$, and token lengths $4 < tl < 48$). These limits are based on the maximum and minimum word and token lengths of reference strings in the CORA corpus.

The sequences that survive this stage are sent to the parsing system, discussed in the next subsection to be parsed.

Stage 4 further removes sequences that are ill-formed. We require that all reference strings include a title and a list of authors after being parsed. Sequences that do not meet these requirements are discarded. Remaining sequences are accepted as valid reference strings.

[3]hosted at http://citeseerx.ist.psu.edu

[4]http://portal.acm.org

[5]www.wikipedia.org

3.2 Parsing

Between Steps 3 and 4 in the recognition process, a reference string is parsed into fields. We treat this problem as a standard classification problem for which a supervised machine learning algorithm can be trained to perform. In implementing our parsing algorithm, recall that we have to meet the criterion of a lightweight solution, which heavily influenced the resulting design.

While a full-fledged reference string parser will extract all available metadata from the reference string, including fields such as publisher name, publisher address and page numbers, we consciously designed our parser to only extract three fields: the title, the authors, and the date of publication. All other tokens are classified as Miscellaneous. There are two reasons for this: 1) for the purposes of sorting the reference strings and subsequently searching for them, these three fields are most likely to be used; 2) restricting classification space to four classes also simplifies the solution, shrinking the model size.

Another simplification was to use a decision tree classifier, as 1) the trained model is easily coded in any declarative programming language (including Javascript, the programming language used by Firefox extensions), and 2) classification is computationally inexpensive, consisting of a series of conditional statements.

Also, instead of the common practice of tokenising a string into individual words, we follow FLUX-CiM's design and use punctuation (except for hyphens and apostrophes) and HTML tags as token delimiters (as seen in the example in Figure 2). This tokenization scheme often leads to phrases. There are a few advantages to this style of tokenisation: 1) considering multiple words as a token allows more complex features to be used, thus giving a better chance of making a correct classification; and 2) reducing the number of tokens per reference string reduces the computational cost of this task.

To classify each phrase, we compile a set of ten features for use in the decision tree, comprising: 1) Lexical (dictionary) features that contain information about the meaning of the words within the token; 2) Local features that contain non-lexical

Feature Name	Description
PfieldLabel (*String*)	The label of the previous token
hasNumber (*Boolean*)	Whether the token contains any numbers
hasYear (*Boolean*)	Whether the token contains any 4-digit number between 1940 and 2040
fieldLength (*Integer*)	The number of characters the token has
hasMonth (*Boolean*)	Whether the token contains any month words (e.g. '**January**', '**Jan**')
oneCap (*Boolean*)	Whether the token consists of only one capital letter e.g. 'B'
position (*Float*)	A number between 0 and 1 that indicates the relative position of the token in the reference string.
hasAbbreviation (*Boolean*)	Whether the token contains any words with more than one capital letter. Examples are 'JCDL', and 'ParsCit'
startPunctuation (*String*)	The punctuation that preceded this token. Accepted values are *period, comma, hyphen, double quotes, opening brace, closing brace, colon, others,* and *none*
endPunctuation (*String*)	The punctuation that is immediately after this token. Accepted values are the same as for startPunctuation

Table 1: List of classifier features

information about the token; 3) Contextual features, which are lexical or local features of a token's neighbours. Table 1 gives an exhaustive list of features used in FireCite.

We had to exclude lexical features that require a large dictionary, such as place names and first names, as such features would add significantly to the loading and execution times of FireCite.

FireCite uses its trained model to tag input phrases with their output class. Before accepting the classification results, we make one minor repair to them. The repair stems from the observation that in gold standard reference strings, both the author and title fields are contiguous. If more than one contiguous sequence of Title or Author classification labels exist, there must be a classification error. When the extension encounters such a situation, FireCite will accept the first encountered sequence as correct, and change subsequent sequences' labels to Miscellaneous (Figure 4).

The parser joins all contiguous tokens for each category into a string, and returns the set of strings as the result.

4 Evaluation

4.1 Recognition

We took faculty homepages from the domains of four universities at random, until a set of 20 homepages with reference strings and 20 homepages without reference strings were obtained. Note that these homepages were sampled from all faculties, not merely from computer science.

Tests were conducted using these 40 pages to obtain the reference string recognition algorithm's accuracy. A reference string is considered found if there exists, in the set of confirmed reference strings C, a parsed text segment c that contains the entire title as well as all the authors' names. Each parsed text segment can only be used to identify one reference string, so if any text segments contain more than one reference string, only one of those reference strings will be considered found.

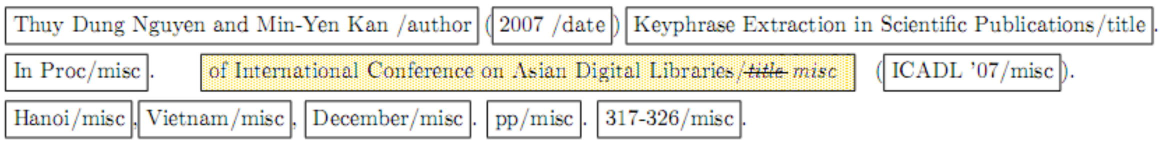
Active stages	Recall	Precision	F_1
1, 2, 3, 4	96.0%	57.5%	.719
2, 3, 4	96.6%	53.6%	.689
1, 2, 4	96.3%	51.6%	.672
1, 2, 3	98.4%	40.9%	.578
1, 2	99.2%	16.1%	.278

Table 2: Results of reference string recognition over forty web pages for five variations of FireCite's reference string recognition

In order to determine the effect of each stage on overall recognition accuracy, some stages of the recognition algorithm were disabled in testing. The results are presented in Table 2. As all test pages come from university domains, all pass the first URL test. When the keyword search is deactivated, all 40 test pages pass Stage 1. Otherwise, 19 pages with reference strings and 6 pages without reference strings pass Stage 1.

The results show that disabling individual stages of the algorithm increases recall slightly, but increases the number of false positives disproportionately more. The fully-enabled algorithm strikes a balance between the number of reference strings found and the number of false positives.

From the above results, we can also see that false positives make up around 40% of the text segments that are recognised as reference strings. However, the majority of reference strings are recognised by the algorithm. In our usage scenario, our output will eventually be viewed by a human user, who will be the final judge of what is a reference string and what is not. Therefore, it is



Thuy Dung Nguyen and Min-Yen Kan /author (2007 /date) Keyphrase Extraction in Scientific Publications/title.
In Proc/misc. of International Conference on Asian Digital Libraries/~~title~~ *misc* (ICADL '07/misc).
Hanoi/misc, Vietnam/misc, December/misc. pp/misc. 317-326/misc.

Figure 4: An example of an incorrectly labelled (highlighted) reference string segment

Page (# of references)	Title	Authors	Date	All Tokens
A (72)	.902	.893	.988	.708
B (52)	.953	.957	.990	.960
C (29)	.684	.304	.774	.651
D (68)	.753	.968	.889	.917
E (8)	.692	.875	1.000	.889
F (45)	.847	1.000	.989	.966
Overall	.836	.916	.948	.878

Table 3: Results of FireCite reference string parsing. Performance figures given are Token F_1. Overall F_1 includes tokens classified as Miscellaneous, and is micro-averaged.

System	Title	Authors	Date	Overall
FireCite	.940	.994	.982	.979
FLUX-CiM	.974	.994	.986	.984

Table 4: Token F_1 of FireCite and FLUX-CiM.

System	Title	Authors	Date	Overall
FireCite	.92	.96	.97	.94
ParsCit	.96	.99	.97	.94
FLUX-CiM	.93	.95	.98	.97

Table 5: Field F_1 of FireCite and other reference string parsers.

more important that we have a high recall rather than high precision. In that respect, this algorithm can be said to fulfill its purpose.

4.2 Parsing

To evaluate the reference string parsing algorithm, we randomly selected six staff publication pages from a computer science faculty. The presentation of each page, as well as the presentation of reference strings on each page, were all chosen to differ from each other. There are a total of 274 reference strings in these six pages. We annotated the reference strings by hand; this set of annotations is used as the gold standard. The six pages are loaded using a browser with FireCite installed. FireCite processes each page and produces a output file with the parsed reference strings. These parsing results are then compared against the gold standard. Table 3 shows the token level results, broken down by web page.

The FireCite reference string parser is able to handle plain-text reference strings as well. A set of plain-text reference strings can be converted into a form understandable by FireCite, simply by enclosing the set of reference strings with `<html>` tags, and replacing line breaks with `<br>` tags. Table 4 shows the token F_1 of the Firecite reference string parser compared FLUX-CiM, while Table 5 shows the field F_1 of FireCite, FLUX-CiM and ParsCit. The test dataset used by all three systems is the FLUX-CiM Computer Science dataset[6] of 300 reference strings randomly selected from the ACM Digital Library. Note that in FireCite and FLUX-CiM, tokens are punctuation delimited whereas in ParsCit, tokens are word delimited.

We feel that above results show that FireCite's reference string parser is comparable to the reviewed systems (although statistically worse), despite its use of a fast and simple classifier and the lack of lexical features that require large dictionaries. The disparity of results between handling web page reference strings and handling plain-text reference strings can generally be attributed to the differences between web page reference strings and plain-text reference strings. Specifically:

- Among the testing data used, the reference strings on one web page (Page C) all begin with the title. However, in the CORA training corpus, all reference strings begin with the authors' names. As a result, in the trained classifier, the first token of every reference string is classified as 'authors'. This error is then propagated through the entire reference string, because each token makes use of the previous token's class as a classifier feature. As shown in Table 3 above, the performance for page C is much worse than the performance for the other pages.

- When web pages are created and edited using a WYSIWIG editor, such as Adobe Dreamweaver or Microsoft Office FrontPage, multiple nested and redundant HTML tags

[6]available at http://www.dcc.ufam.edu.br/ ẽccv/flux-cim/ Computer-Science/

	Min. time	Max. time	Avg. time
With references	90	544	192
W/o references	6	222	74
All pages	6	544	133

Table 6: FireCite execution time tests over 40 web pages. Times given in milliseconds.

tend to be added to the page. Because FireCite treats HTML tags as token delimiters, these redundant tags increase the number of tokens in the string, thus affecting the token position feature of the classifier, causing some tokens to become incorrectly classified.

Some of the inaccuracies can also be attributed to mistakes from reference string recognition. When the reference string is not correctly delimited, text that occurs before or after the actual reference string is also sent to the reference string parser. This affects the token position and previous token label features.

The competitive advantage of FireCite's reference string parser is that it is very small compared to the other systems. FireCite's reference string parser consists only of a decision tree coded into JavaScript if-then-else statements, and a couple of JavaScript functions, taking up a total of around 38KB of space. On the other hand, as mentioned above, FLUX-CiM optimally requires a database of around 3000 reference strings, while ParsCit's classifier model and dictionaries require a total of 10MB of space. These characteristics also make the reference string parser fast. Speed tests were conducted over 40 web pages taken from the domains of four universities, 20 of which contain reference strings and 20 of which do not. The results are summarised in Table 6. From these results we can infer with some confidence that FireCite will add no more than one second to the existing time a page takes to load.

5 Extension Front End

We thus implemented a prototype BMA as a Firefox extension that uses the recognizer and parser as core modules. As such an extension interacts with users directly, the extension's front end design concentrated on functionality and usability issues that go beyond the aforementioned natural language processing issues.

Browser extension based BMAs are not new. Zotero[7] as well as Mendeley[8] both offer BMAs that manage reference (and other bookmark) information for users. However, neither recognizes or delimits free formed reference strings found on general webpages. Both rely on predefined templates to process specific scholarly websites (e.g. Google Scholar, Springer).

In developing our front end, our design hopes to complement such existing BMAs. We followed a rapid prototyping design methodology. The current user interface, shown in Figure 5, is the result of three cycles of development. Up to now, feedback gathering has been done through focus groups with beginning research students and individual interviews with faculty members. Rather than concentrate on the design process, we give a quick synopsis of the major features that the FireCite prototype implements.

One-Click Addition of References: FireCite appends a clickable button to each reference string it detects through the recognition and parsing modules. Clicking this button adds the reference string's metadata to the reference library. The design draws attention to the presence of a reference without disrupting the layout of the webpage.

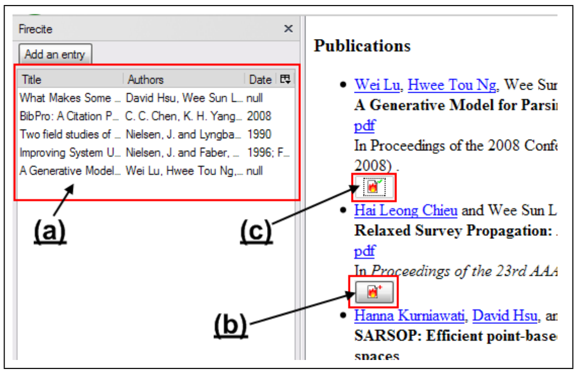
Reference Library: The reference library opens as a sidebar in the browser. It is a local database containing the metadata of the saved references. The library allows reference strings to be edited or deleted, and sorted according to the three extracted metadata fields.

Manual recognition and addition: The core modules occasionally miss valid references. To remedy this, users can manually highlight a span of text, and through the right click context menu, ask FireCite to parse the span and append an "add citation" button. The user may also manually add or edit reference metadata directly in the sidebar. This feature allows the user to add entries from his existing collections of papers, or to add entries for which no reference string can be found (such as papers that have not been published).

PDF download: When a reference is added to the local library, any Portable Document Format (PDF) file associated with the reference string is downloaded as well. Appropriate PDF files are found heuristically by finding a hyperlink leading to a PDF file within the text segment. The downloaded PDF files are stored in a single folder

[7]http://www.zotero.org

[8]http://www.mendeley.com



Title	Authors	Date
What Makes Some ...	David Hsu, Wee Sun L...	null
BibPro: A Citation P...	C. C. Chen, K. H. Yang...	2008
Two field studies of ...	Nielsen, J. and Lyngba...	1990
Improving System U...	Nielsen, J. and Faber, ...	1996; F...
A Generative Model...	Wei Lu, Hwee Tou Ng,...	null

Publications

- Wei Lu, Hwee Tou Ng, Wee Sur
  A Generative Model for Parsi
  pdf
  In Proceedings of the 2008 Conf
  2008) .
- Hai Leong Chieu and Wee Sun L
  Relaxed Survey Propagation: .
  pdf
  In *Proceedings of the 23rd AAA*
- Hanna Kurniawati, David Hsu, ar
  SARSOP: Efficient point-base
  spaces

Figure 5: Screenshot of FireCite prototype illustrating (a) the reference string library, (b) button appended to each reference string, and (c) button state after the reference string has been added to the list.

within Firefox's storage location for the extension, and can be opened or deleted through the sidebar interface. With this feature, the user will not need to juggle his PDF files and reference string library separately.

As a preliminary evaluation, we presented FireCite to four academics in separate unstructured interviews. All four subjects saw the potential of FireCite as a BMA, but not the usefulness of recognising reference strings on the Web. Two of them pointed out that they rarely encounter reference strings while browsing the Web, while another only needs to search for specific, known papers. When asked in detail, it was apparent that subjects do actually visit web pages that contain many reference strings. In DBLP, each entry is actually a reference string. In the ACM Digital Library, in every article information page, there is a list of reference strings that have been extracted from the bibliography of the article using Optical Character Recognition (OCR).

From our study, we conclude that integration with template based recognition (*a la* Zotero) of sites such as DBLP, Google Scholar and ACM Portal, has better potential. As expected, since the subjects all have significant research experience, they have already developed suitable research methods. The challenge is for FireCite to fit into their workflow.

6 Conclusion

This paper describes FireCite, a Firefox extension that can recognise and delimit metadata from reference strings on freeform web pages. FireCite's

"Liquidity-Based Model of Security Design," with Darrell Duffie, Econometrica, 1999, 67, 65-99.

Figure 6: A reference string with one author's name omitted.

Michael Collins and Terry Koo.
Discriminative Reranking for Natural Language Parsing.
Computational Linguistics 31(1):25-69.

Figure 7: A reference string with its year omitted. Part of a list of reference strings organised by their year of publication.

implementation demonstrates it is possible to do these tasks in real-time and with a usable level of accuracy.

We have validated the accuracy of FireCite's embedded recognition and parsing modules by comparing against the state-of-the-art systems, both on web based reference strings that use HTML tags as well as gold-standard reference strings in plain text. FireCite achieves a usable level of reference string recognition and parsing accuracy, while remaining small in size, a critical requirement in building a browser extension. This small model allows FireCite to complete its processing of reference heavy webpages in under one second, an acceptable level of latency for most users. Preliminary user studies show that the FireCite system should incorporate template based recognition of large scholarly sites as well for maximum effectiveness.

Future work on the parsing and recognition will focus on capturing implied contextual information. On some web pages the author may omit their own name, or place the year of publication in a section head (Figures 6 and 7). We are working towards recognizing and incorporating such contextual information in processing.

Acknowledgements

This work was partially supported by a National Research Foundation grant "Interactive Media Search" (grant # R 252 000 325 279).

References

Chia-Hui Chang, Chun-Nan Hsu, and Shao-Cheng Lui. 2003. Automatic information extraction from semi-structured web pages by pattern discovery. *Decis. Support Syst.*, 35(1):129–147.

Eli Cortez, Altigran S. da Silva, Marcos André Gonçalves, Filipe Mesquita, and Edleno S. de Moura. 2007. FLUX-CIM: flexible unsupervised extraction of citation metadata. In *Proc. JCDL '07*, pages 215–224, New York, NY, USA. ACM.

Isaac G. Councill, C. Lee Giles, and Min-Yen Kan. 2008. ParsCit: An open-source CRF reference string parsing package. In *LREC '08*, Marrakesh, Morrocco, May.

Junfei Geng and Jun Yang. 2004. Autobib: automatic extraction of bibliographic information on the web. pages 193–204, July.

Erik Hetzner. 2008. A simple method for citation metadata extraction using hidden markov models. In *Proc. JCDL '08*, pages 280–284, New York, NY, USA. ACM.

Fiona Fui-Hoon Nah. 2004. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology Special Issue on HCI in MIS*, 23(3), May-June.

Fuchun Peng and Andrew McCallum. 2004. Accurate information extraction from research papers using conditional random fields. pages 329–336. HLT-NAACL.

Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. 1999. Learning hidden markov model structure for information extraction. In *AAAI'99 Workshop on Machine Learning for Information Extraction*.

Xin Xin, Juanzi Li, Jie Tang, and Qiong Luo. 2008. Academic conference homepage understanding using constrained hierarchical conditional random fields. In *Proc. CIKM '08*, pages 1301–1310, New York, NY, USA. ACM.

Kai-Hsiang Yang, Shui-Shi Chen, Ming-Tai Hsieh, Hahn-Ming Lee, and Jan-Ming Ho. 2008. CRE: An automatic citation record extractor for publication list pages. In *Proc. WMWA'08 of PAKDD-2008*, Osaka, Japan, May.

Yanhong Zhai and Bing Liu. 2005. Web data extraction based on partial tree alignment. In *Proc. WWW '05*, pages 76–85, New York, NY, USA. ACM.

Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. 2006. Simultaneous record detection and attribute labeling in web data extraction. In *Proc. KDD '06*, pages 494–503, New York, NY, USA. ACM.