Automatic Data Extraction From Template Generated Web Pages

Ling Ma and Nazli Goharian,
Information Retrieval Laboratory
Department of Computer Science
Illinois Institute of Technology
{maling, goharian}@ir.iit.edu

Abdur Chowdhury
America online Inc
cabdur@aol.com

Abstract
Information Retrieval calls for accurate web page data extraction. To enhance retrieval precision, irrelevant data such as navigational bar and advertisement should be identified and removed prior to indexing. We propose a novel approach that identifies the web page templates and extracts the unstructured data. Our experimental results on several different web sites demonstrate the feasibility of our approach.

Keywords
Automatic template removal, text extraction, information retrieval

1. Introduction
Text extraction from HTML page is a critical preprocess for information retrieval. In this phase, page content must be extracted and irrelevant template data removed.

Hyper Text Markup Language is not designed for structured data extraction at the first place, but mainly for data presentation [1]. HTML pages are usually "dirty", i.e., improper closed tags, improper nested tags, and incorrect parameter value of a tag are examples of such problems. In order to reduce parsing error over ill formed web pages caused by loose HTML standard, World Wide Web Consortium [2] recommend stricter standard Markup Languages, such as XHTML and XML. However, parser of search engine still needs to cope with existing web pages that do not conform to the new standards.

Conventional approach for web page data extraction requires one wrapper for each web site. A wrapper is a program that can extract data from web pages according to the tag structure of the page. Programmers have to manually find the reference point and absolute tag path for the targeted data content. A wrapper is incorrect if it is trained with sample pages missing optional or disjunction elements. A modification to a tag path causes the wrapper to easily break. Therefore, manually coding and maintaining wrapper can be all-consuming work.

A HTML web page contains information presented in several forms, such as text, image and flash, etc. HTML pages from the same web site usually share the same look, layout and structure. A tailored wrapper can extract data from a particular web site because the targeted web pages are usually generated from the same template, and therefore share identical navigation bars. In addition, some sections of the pages are always allocated for advertisements. Navigational bars and advertisements are in fact noise data for information retrieval as they have no direct relevancy with content data.

As shown next in figure 1, we must extract unstructured data from the web page; meanwhile remove irrelevant template navigation bars and advertisements. Based on our observations of html page, we propose a novel approach to achieve this goal. We do not retrieve the multimedia content at the current stage, thus no advertisements are included in extracted text.

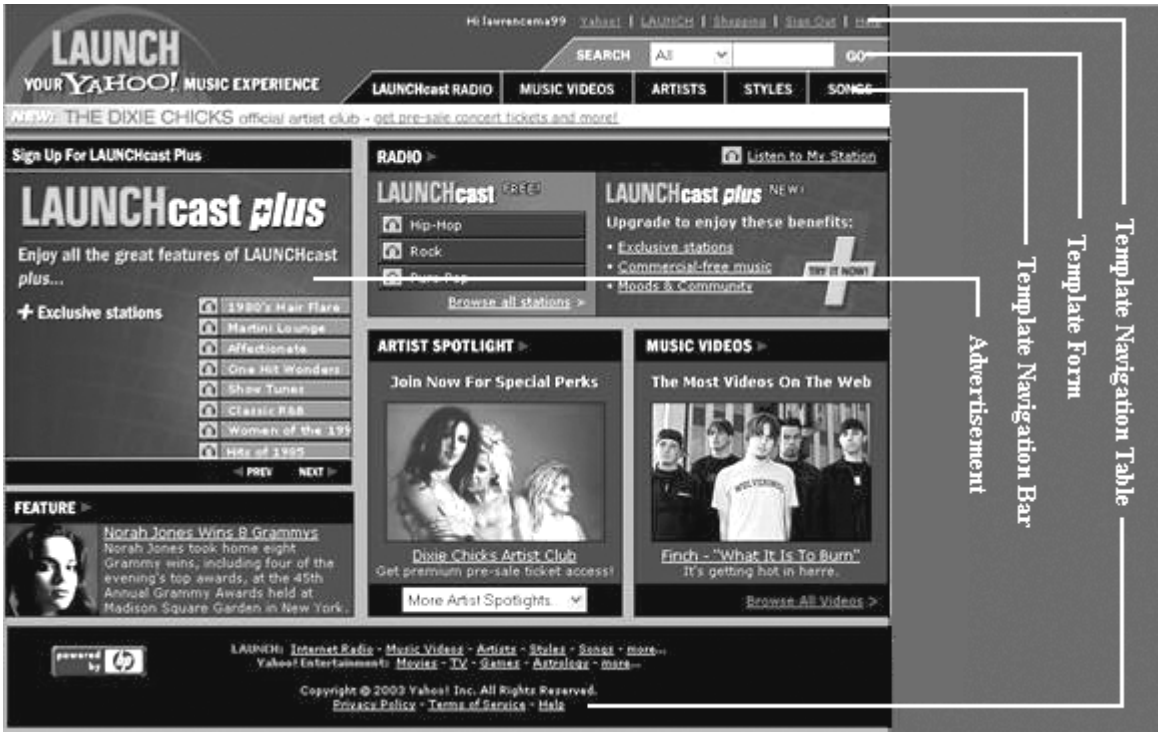

Figure 1. Sample Web page with Template And Advertisement

2. Related Work

The motivation of our work is to extract text from web pages and remove the irrelevant template. Crescenzi et al [3] (Road Runner Project) and Arind Arasu's [4] investigated techniques in recognizing semantic structure of data from large web sites. In comparison, our task does not require grammar induction. Helena Ahonen-Myka [7] and Florian Beil et al [8] designed efficient algorithms for frequent term set clustering problem. We take advantage of page tag tree structure to find the template, therefore identifying template in HTML pages is more straightforward than frequent term set clustering problem.

Ziv Bar-Yossef et al [5] observed templatized pages from professional designed website share the same context for browsing. They also showed that template skews ranking and hence reduces precision. Also, they showed that navigation bar and paid advertisement are in direct violation of Hypertext IR Principles [5]. These observations motivate the attempt to segment web page into smaller but more cohesive pagelets. The notion of pagelet does provide a new angle for the granularity of information retrieval. However, it is difficult to obtain the optimum threshold value k for determining whether a sub parse tree is indeed a pagelet. Similarly, Xiaoli Li et al [6] proposed segmenting web page into topically more focused micro information units (MIU). Their algorithm employs text font property and term set similarity, building HTML page tag tree, merging heading with text, and adjacent similar paragraphs as MIU.

URL and web page for commercial sites backed by dynamic commodity database are usually generated dynamically. Corresponding to different category of products, slightly different template is applied. Thus there can be more than one template from a web site. Assuming pages that look similar belong to the same template, we cluster html pages based on Crezenti et al's [9] algorithm, and then identify elements of web pages that were generated by a common template. Crezenti et al analyzed tag and link property of html pages. Features such as distance from the home page, tag periodicity, URL similarity, and tag probability were applied as measures of HTML page similarity. Their method produces high accuracy in clustering pages generated from the same template.

3. Our Approach

3.1 Observations

Observation 3.1.1: Most professional web sites' navigational bars are coded with table or image map powered by JavaScript. We observed that most text oriented web site such as cnnfn.com, netflix.com and nba.com use table menu as the navigation bar. A bar item can be either an image link or text link. Submenu appears when mouse is over its parent main menu item. Another form of navigation bar is image map, which is a single image corresponding to multiple links defined by area.

Observation 3.1.2: HTML table serves the purpose of confining the size of navigation bar either vertically or horizontally. We found that either width or height parameter of a table is defined to confine navigational bar within a limited region of a page. This property of navigation bar is easy to interpret: the page editor must leave enough space in the center of the page for essential information.

Definition 3.1 table text chunk
A table text chunk is a consecutive terms chunk extracted from a common HTML table. Using a single token as the minimum granularity for counting document frequency, results in unnecessary computation and incorrect result. The granule of our method is not a single token, but table text chunk. Table text chunk is a similar primitive discussed in the web page copy detection system [11]. According to our experiments, it is a much better primitive because template generation program does not attempt to "add salt" in template to prevent template extraction.

3.2 Algorithm Pseudo-code List

```
Algorithm: Template table text chunk removal algorithm
Input:          Clustered web pages
Output:  textMap: discovered template table text chunks.
                * {table text chunk, document frequency} pairs saved in hash map
Variables:      Integer: Threshold;        * document frequency threshold for template discovery
                Buffer: textChunk;         * temporary table text chunk

1. For Each web page g in the cluster
2.      p ← html parser of g
3.      While p has more HTML nodes      * a html node can be any tag, string element, etc
4.              n ← next html node extracted by p;
5.              Update current tag path;
6.              If n is not a table node
7.                      textChunk ← textChunk + extracted text of n
8.              If n is a table node
9.                      If textChunk is already in textMap
10.                             If textChunk is in a different document
11.                                     Increment document frequency for current text chunk;
12.                     else
13.                             Put textChunk in textMap with document frequency 1;
14.                     Clear textChunk Buffer;
15.     End While
16. End For
17. While textMap has more {textChunk, document frequency} pairs
18.     h ← next item in textMap;
19.     If document frequency of h ≥ threshold
20.             Print textChunk of item h
21. End While
```

Figure 2. Template Table Text Chunk Removal Algorithm

3.3 Algorithm Discussion

Given web pages clustered by Crezenti's [9] method, our algorithm has two passes over the HTML pages cluster. In the first pass, we map each table text chunk into a different hash value. If the document frequency of a particular table text chunk equals or exceeds the document frequency threshold, we label it as a candidate template component. The labeled template table text chunks are then stored in a hash map for future check up. In the second pass, each table text chunk is compared to the keys in hash map. We do not output the table text chuck if it is indeed already in the hash map. Thus, the output of second pass is the extracted data without navigation bars. To facilitate token indexing, HTML symbol decoding is required. For example, " " and "&lt" should decoded as "space" and "<" respectively, according to HTML character entity references [12].

The algorithm requires a table text chunk look up for each table during first pass over cluster of *n* web pages. Therefore, it is efficient to run the algorithm on clustered web pages to be able to handle the processing due to the memory limitations. Furthermore, we can reduce *n* by randomly picking a fraction of pages from the cluster. The tradeoff still produces good result, which is presented in figure 4.

4. Experimental Framework and Preliminary Results

Our experimental platform is a Pentium III 650Mhz Dell Inspiron 4000 notebook with 192MB memory. We used eight different clustered HTML collections to obtain templates covering various type of topics or structures, such as news, sports, e-business, entertainments and fashion. For its simplicity and flexibility to parse the html pages, we used HTMLParser [10]. The metrics we used to measure our system is defined under section 4.1.

4.1 Results

For each collection, we picked some number of pages, for which the size and the features are listed in the table, along with the total and unit execution time of our program on each collection. The unit execution time is calculated as the total execution time divided by size of the collection. The measures used to gather statistics and identify the correctness of our approach are described in the definitions 4.1.1 and 4.1.2.

Definition 4.1.1 Average template fraction:
Template tokens are text terms and symbols contained by web page template. Let *t* be the number of template tokens extracted, and *p* be average number of text tokens extracted per web page among a cluster of pages. The average template fraction (atf) is calculated as:

Average template fraction (atf) = t / p.

atf indicates the ratio of the template tokens to the text tokens in a web page. A low *atf* value indicates that the page is information intensive.

Table 1 shows the name, size and characteristics of the data we used in our experimentations.

Table 1. Experimental Data and the Execution Time

Web Page Collection	*Pages*	*Size*	*Page Feature*	*Total Exec Time*	*Unit Exec Time*
Cnnfn.com	8	341k	Table	1713 ms	5.0 ms / k data
NBA.com	11	486k	Table	2233 ms	4.6 ms / k data
Netflix.com	9	477k	Table + Image links	2193 ms	4.6 ms / k data
Ebay.com	41	2370k	Table + forms	10655 ms	4.5 ms / k data
Amazon.com	13	759k	Table	3024 ms	4.0 ms / k data
Disneylandsource.com	8	132k	Image Maps	1002 ms	7.6 ms / k data
Elle.com	16	355k	Image bar +Image links	2043 ms	4.4 ms / k data
Launch.yahoo.com	9	385k	Flash + Image links	2244 ms	5.8 ms / k data

Table 2. atf and ttdr of Web Page Collection

Web page Collection	*Average Template Fraction*	*Template Table Discovery Ratio*
Cnnfn.com	34.1%	1.00
NBA.com	18.1%	0.86
Netflix.com	17.8%	1.00
Amazon.com	19.3%	1.00
Ebay.com	22.5%	1.00
Disneylandsource.com	54.1%	1.00
Elle.com	19.3%	1.00
Launch.yahoo.com	29.3%	1.00

Definition 4.1.2 Template table discovery ratio:
A *HTML node* is a html tag. It can be a single tag like
 or a composite tag like <form> </form>. A html node can have children nodes, which are lexically nested in its parent node. Let *nd* be the number of template HTML nodes discovered, and *nt* be the actual number of template HTML table nodes manually counted. We define:

Template table discovery ratio(ttdr) = nd / nt

A ttdr of 1 indicates every template table is discovered; a *ttdr* larger than 1 indicates some tables are counted as part of the template by mistake; and a *ttdr* less than 1 means some template tables are not discovered. Table 2 illustrates the generated results.

4.2 Discussion on the Results

We observed different characteristics of different types of web sites. As shown in table 2, entertainment web sites such as disneylandsource.com and launch.yahoo.com tend to have a high *atf* value. In disneylandsource.com a page is typically composed of image or navigation bar, which is a large image map. Launch.yahoo.com present pages with embedded Flash and images. Similar to disneylandsource.com, there is not much template table text for launch.yahoo.com. Elle.com is an exception because template of elle.com is mainly image map. Elle.com presents information by image bar and image text. The navigation bar of elle.com is an image link table. In fact, quite few texts from elle.com are extracted. There is hardly any html text content besides images in such web sites.

We observed a problem caused by applying document frequency threshold. An identical template table text chunk can appear more than once in different locations of one page. Our program does not count a redundant template navigation bar at different location of a page, and therefore template table discovery ratio is 0.86 for nba.com. A naïve solution for this problem is to count tag path as second key of textMap in figure 2, which may still not solve the problem if two template table text chunks have the same tag path.

Although a cluster includes many html pages from the same template, document frequency threshold should not be set to the total number of pages. Some "optional" template-table-chunks do not appear in all samples. For example, netflix.com DVD reviews have a table for "watch preview". This table shows up for most of the DVDs but not for some of the old movies. In other cases, the combinations of template tables form a template variant, for which, we further lowered document frequency threshold to identify the template variants. However, if we lower document frequency threshold too much, template table discovery value rises above one.

For example, as illustrated in Fig 3, if we assume nothing else but template tables appear in more than 3 pages of Amazon web page cluster, and lower document frequency threshold from 4 to 3, *ttdr* value rises to 1.17. A higher *ttdr* value indicates that our algorithm treats the variant navigation table and advertisements as new template elements.

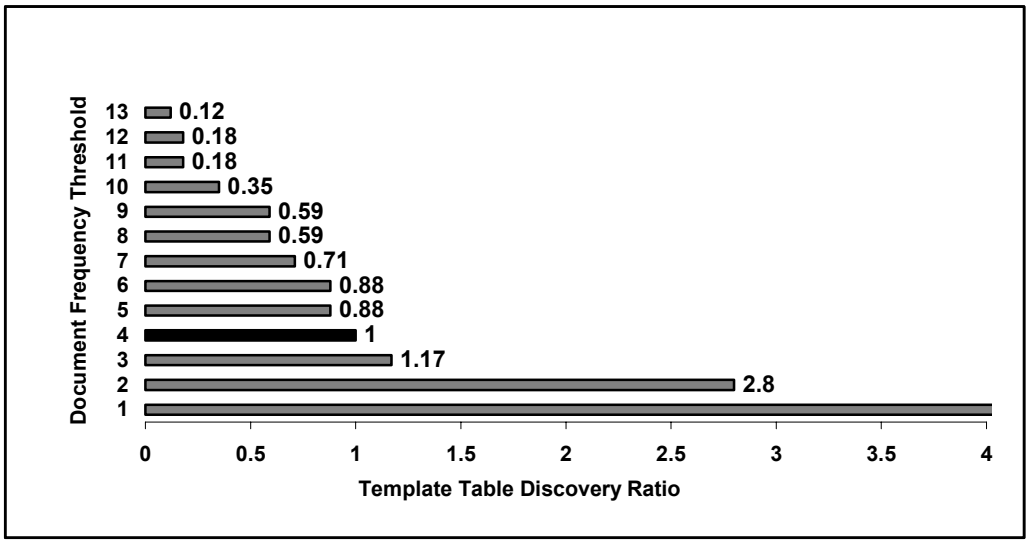

Figure 3. ttdf – df Threshold for Amazon Collection

We also found that most template tables appear in every page of the collection. Our approach can be even faster by running on less number of pages. In one experiment, we randomly picked 9 out of the 13 Amazon web pages. The program could still recognize 86% of the template tables. This trade-off saves the execution time for large web page clusters. There are 41 web pages in ebay.com cluster, picking 9 pages randomly achieved *ttdr* of 1, which means every template table was still discovered. Meanwhile, processing time of our program is reduced to 27% of the running time when using all 41 pages. Figure 4 illustrates the template table discovery ratio (ttdr) when picking different number of pages from Amazon collection, with document threshold set to 4.

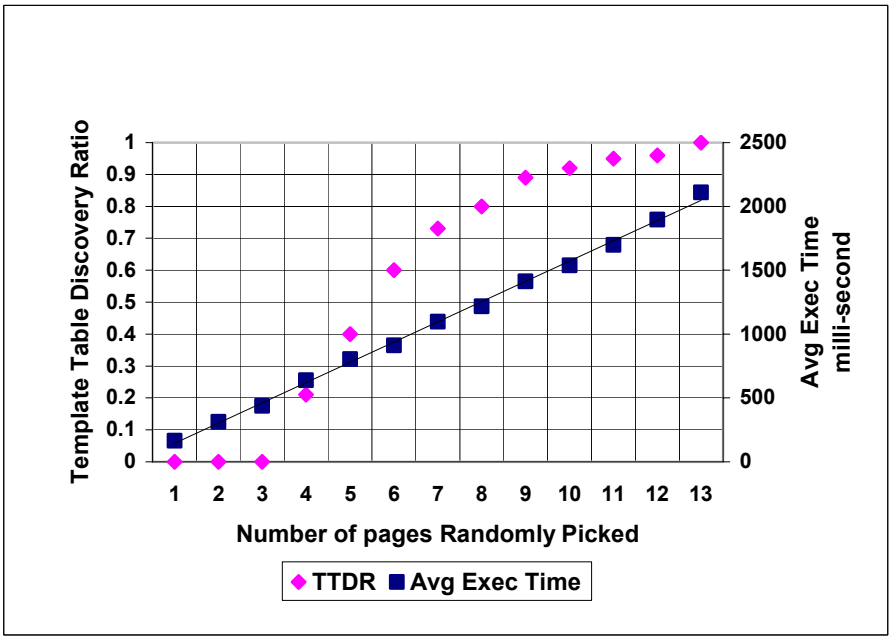

Figure 4. ttdr, Avg Exec Time – number of pages picked for Amazon Collection

5. Conclusions and Future Work

In conclusion, we designed a novel approach to identify the web page template and extract unstructured data. Our algorithm is fine tuned for accuracy and efficiency. Experimental results on several different web sites established the feasibility of the algorithm.

We also found several challenges through our investigation. Many HTML web pages incorporate dynamic technique such as javascript and PHP script. For instance, sub menu of navigation bars are rendered by "document.write" method of javascript on the event of mouse over. Image map are also implemented with javascript code. We certainly don't want to include any template navigational bar in the indexing phase. However, there are cases when page content are rendered with javascript methods. For example, the link the bottom of google home page "Make google your home page" appears only if you use other site as your home for Internet Explorer. In our future work to avoid the risk of missing critical information we need to handle those scripts.

Another issue is handling images with descriptive functions in the web pages. It is not good to simply remove any image from the page, as we all know that a picture sometimes worth one thousand words. For example, fashion site like elle.com prefers image text rather than a plain text. Image text certainly facilitates readers' understanding; on the other hand it prohibits data extraction. Recently Google enables the search for images by the image name. From our observation, the "ALT" parameter of an image link sometimes also describes the image content, if edited by a human editor. We include both image name and "ALT" value in our data extraction process, while getting rid of worthless values such as "img9", "spacer" or "graphic". We are working on evaluating the effect of including informational image name and image name parsing rules.

Acknowledgements

We thank Valter Crescenzi and Ziv Bar-Yossef for giving us insightful suggestions. We also thank Don Park for giving us information on the existing parsers.

References

[1] Jussi Myllymaki, Effective Web Data Extraction with Standard XML Technologies , World Wide Web Conference (WWW10), 2001.

[2] W3C HyperText MarkUp language Home page, http://www.w3.org/MarkUp/

[3] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, RoadRunner: Automatic Data Extraction from Data-Intensive Web Sites, International Conference on Management of Data and Symposium on Principles of Database Systems (SIGMOD02), 2002.

[4] Arvind Arasu, Hector Garcia-Molina, Extracting Structured data from Web pages, Stanford University, Technical Report, 2002.

[5] Ziv Bar-Yossef , Sridhar Rajagopalan, Template Detection via Data Mining and its Applications , World Wide Web Conference (WWW02), 2002.

[6] Xiaoli Li, Bing Liu, Tog-Heng Phang, Mingqing Hu, Using Micro Information Units for Internet Search, ACM Special Interest Group on Knowledge Discovery in Data (SIGKDD02), 2002.

[7] Helena Ahonen-Myka, Discovery of Frequent Word Sequences in Text, Template Detection via Data Mining and its Applications, University of Helsinki, 2002.

[8] Florian Beil, Martin Ester, XiaoWei Xu, Frequent Term-based text clustering , ACM Special Interest Group on Knowledge Discovery in Data (SIGKDD02).

[9] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, Wrapping-Oriented Classification of Web pages , Symposium on Applied Computing (SAC02), 2002.

[10] HTMLParser, www.htmlparser.sourceforge.net/

[11] Narayanan Shivakumar, Hector Garcia-Molina, Building a scalable copy detection system, ACM Digital Library (ACMDL96),1996.

[12] Character Entity References in HTML 4, www.w3.org/TR/REC-html40/sgml/entities.html