Tree-Structured Template Generation for Web Pages

Shui-Lung Chuang
Institute of Information Science
Academia Sinica
Taipei 115, Taiwan
slchuang@iis.sinica.edu.tw

Jane Yung-jen Hsu
Dept. of Comp. Sci. & Info. Eng.
National Taiwan University
Taipei 106, Taiwan
yjhsu@csie.ntu.edu.tw

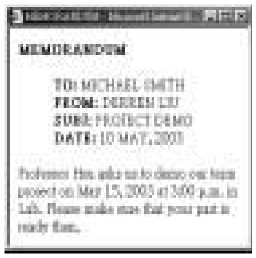
Abstract

As the web becomes an increasingly important source of information, tools for modeling, searching, and extracting information from Web pages are indispensable. By modeling the structure of a Web page defined by its markup tags, one can easily extract target information using structural templates. This paper introduces the Tree Template Automatic Generator (TTAG) that learns tree-structured templates from training Web pages. TTAG was applied to both query-based and frequently updated Web sites, and produced effective templates from a small number of examples. The experiments show that TTAG is a powerful extraction tool for semi-structured information sources.

1 Introduction

As the Web becomes the most popular source of information, valuable data are present in manually encoded or automatically generated Web pages. Techniques for modeling, searching and mining information from Web pages have gained much attention from research in database and artificial intelligence; e.g., syntax parsing grammar [2], delimiter-string pattern [13], and finite-state transducer/automaton [10]. Most approaches treat each Web page as a sequence of strings, even though HTML/XML documents natively have a tree structure. As was pointed out in [11], while the two HTML documents in Figures 1 and 2 use many different tags, they share much of the same structural layout. To take advantage of the structural context of the target fields within a document, this research investigates the global layout of documents.

A document is a combination of three ingredients: *content*, *format* and *structure*. Content is the actual data, format prescribes the presentation of the data, and structure relates the pieces in the document. Techniques for semantic processing of document content are not mature enough for practical use. Structural information, on the other hand,



MEMORANDUM

TO: MICHAEL SMITH
FROM: DERREN LIU
SUBJ: PROJECT DEMO
DATE: 10 MAY, 2003

Professor Hsu asks us to demo our term project on May 15, 2003 at 3:00 p.m. in Lab. Please make sure that your part is ready then.

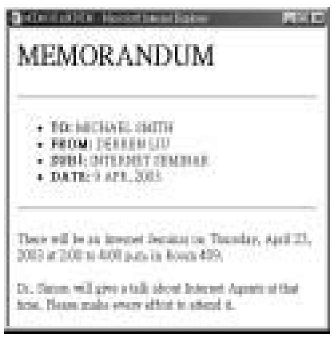
```
<html>
<head><title>MEMORANDUM</title></head>
<body>
<b> MEMORANDUM </b>
<dd> <b>TO:</b> MICHAEL SMITH </dd>
<dd> <b>FROM:</b> DERREN LIU </dd>
<dd> <b>SUBJ:</b> PROJECT DEMO </dd>
<dd> <b>DATE:</b> 10 MAY, 2003 </dd>
<p>Professor Hsu asks us to demo our term
project on May 15, 2003 at 3:00 p.m. in
Lab. Please make sure that your part is
ready then.</p>
</body>
</html>
```

Figure 1. A memo and its HTML source.

can serve as useful clues in identifying and retrieving information. Given a Web document, the markup tags and their nesting relations define a hierarchical structure of content blocks (see Figure 2), which can serve as the basis for deeper analysis of the structural layout of the document.

A group of Web pages often share a similar structure if the pages are designed under the same guideline, or if the pages are generated automatically, e.g., the result pages of a search service or the front page of a daily news. Such regularities allow us to simply examine one or a few similar pages in order to figure out their common structure, which can in turn be used to determine the information contained in additional similar pages. For example, the memos shown in Figure 1 and 2 can be modeled and matched by the same *tree-structured template*. The templates can be generalized to represent document groups with small structural variations. Ideas from traditional tree automata are adopted in designing efficient matching algorithms to identify target information within a document.

Designing effective document templates is a tedious task. This research addresses the issue by developing the TTAG algorithm for automatic generation of tree-structured templates from training documents. TTAG depends on a top-down level-by-level multi-string alignment of markup tags in the document tag trees. In the rest of this paper, we first review some related work and introduce the notion of tree-structured template. The template generation algorithm is then presented in detail, followed by the experiments.



MEMORANDUM

- TO: MICHAEL SMITH
- FROM: DERREN LIU
- SUBJ: INTERNET SEMINAR
- DATE: 9 APR, 2003

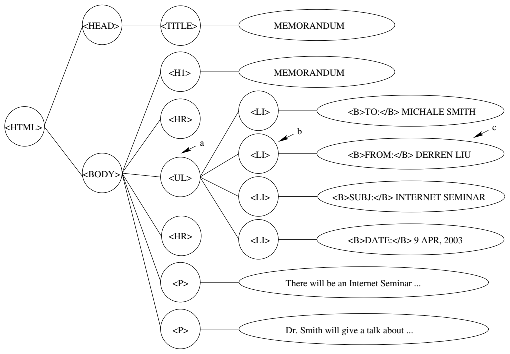
There will be an Internet Seminar on Thursday, April 23, 2003 at 2:00 to 4:00 p.m. in Room 409.

Dr. Simon will give a talk about Internet Agents at that time. Please make every effort to attend it.

(a)

```
<HTML>
<HEAD> <TITLE> MEMORANDUM </TITLE> </HEAD>
<BODY>
<H1> MEMORANDUM </H1>
<HR>
<UL>
<LI> <B>TO:</B> MICHAEL SMITH </LI>
<LI> <B>FROM:</B> DERREN LIU </LI>
<LI> <B>SUBJ:</B> INTERNET SEMINAR </LI>
<LI> <B>DATE:</B> 9 APR, 2003 </LI>
</UL>
<HR>
<P>There will be an Internet Seminar on Thursday,
April 23, 2003 at 2:00 to 4:00 p.m. in Room 409.
</P>
<P>Dr. Simon will give a talk about Internet Agents
at that time. Please make every effort to attend it.
</P>
</BODY>
</HTML>
```

(b)



(c)

Figure 2. Another memo, its markup source, and the corresponding document tree structure.

2 Related Work

There has been much research on Web information extraction [8, 15]. Previous work on wrapper induction/generation is often based on the linear string model, e.g. delimiter string [13] and finite-state transducer [10]. The learned pattern rules mostly capture the local regularities. Muslea et al. [16] proposed *Stalker* to incorporate global structural information. Stalker used a manually constructed embedded catalog tree to hierarchically describe the target fields within the documents. Kosala et al. [12] explores document tree structure in extraction tasks using grammatical induction on tree automata.

In contrast, document templates can model the entire web page of interest, rather than a set of shallow extraction patterns. RoadRunner [6] treated each input page as a list of tokens, and used regular expression as the template. Arasu and Garcia-Molina [1] proposed a more sophisticated and improved approach. Hsu and Yih [11] first proposed template-based mining. Chuang [5] created the initial design of the template generation algorithm. Using the tag trees of Web pages to generate explicit regular tree templates; the proposed approach can benefit from the hierarchical structure information contained in trees.

More recent research aims to identify data records of interests in a data-intensive Web site automatically. In particular, some focused on data extraction from tables and lists [17]. Embley et al. [7] proposed a method using heuristics and domain ontology to perform the task, and Buttler et al. [3] extended it with more heuristics to waive the need of domain knowledge. IEPAD [4] used PAT trees [9] to extract repetitive patterns from the Web page string, and further applied sequence alignment to overcome the inexact match of patterns. According to a comparative study in [14], IEPAD produced too many patterns, most of them are spurious. Liu et al. [14] proposed the MDR algorithm that directly operated on HTML tag trees to identify interested data regions. Their implementation was limited to mining data records formed by HTML table-related tags. Overall, automatically identifying interested data records is difficult, since only users know what they are really interested in. The automatic approaches produce plausible candidates, thereby facilitating the process of labeling training data.

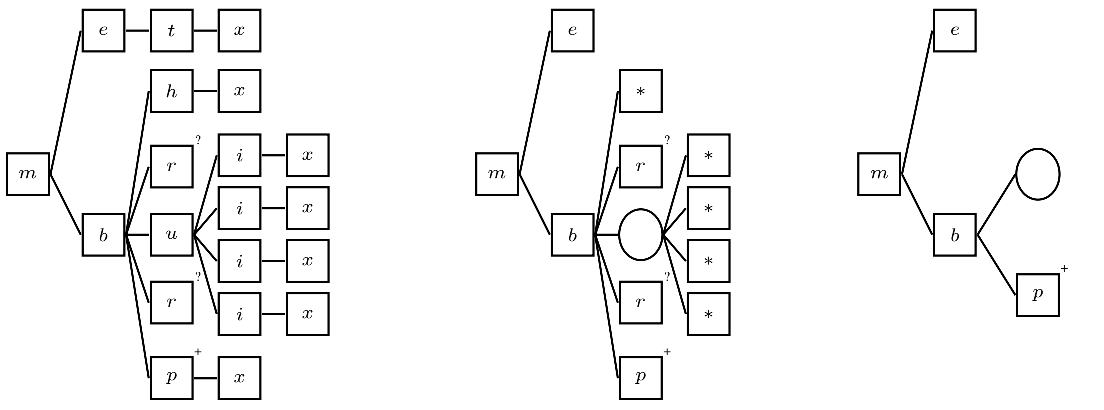
3 Document and Template Tree

A Web page can be parsed into a tree, called *document tree* or *tag tree* (see Figure 2(c)). Each node is associated with a markup tag or a text string. The children of a node can be represented as a sequence or string. Regular expression is a convenient tool for describing a set of similar strings. The superscript ? is optional closure, indicating that an item can appear once or zero; $+$ is positive repeat closure, indicating that an item should appear non-zero times.

A template tree may contain multiple types of nodes. A square denotes an exact node, which can only match a document tree node with the same label a. The wildcard label $*$ can be used to match any label. The ellipse indicates a compound node, which matches any node. It can also match two or more consecutive nodes, which become the compound node's children.

Figure 3 shows several template trees that match the memo document tree shown in Figure 2(c). The template in Figure 3(a) matches any memo with multiple paragraphs and optional block delimiter. The template shown in Figure 3(b) is more relaxed and matches the memo shown in Figure 1. The template in Figure 3(c) is further relaxed by treating all header block as a compound node.

Determining the matching between a template tree and a document tree can be very complicated. Fortunately, many efficient tree-matching algorithms have been designed in the literature [18]. We are now ready to focus on how to generate a template tree for a class of similar documents.



Abbreviation List: m : <html>, e : <head>, t : <title>, b : <body>, h : <h1>, r : <hr>, u : <ul>, i : <li>, p : <p>, x : CDATA.

Figure 3. Several example template trees for matching the memo shown in Figure 2.

4 Template Generation Algorithm

The template generation algorithm is designed to compose a template top-down, beginning from the root and going down one level at a time, as the nature of a tree. For a template node n, its child nodes is determined as follows: First, we find the nodes corresponding to n in the training set, and treat their children as strings. Then, a regular expression pattern for matching all the strings is inferred. Finally, the resulting pattern is translated into a sequence of template-tree nodes, becoming the child nodes of n. In the following subsections, we describe the algorithms for multiple sequence alignment, inducing patterns from an alignment, and the overall template generation algorithm.

4.1 Multiple Sequence Alignment

Sequence alignment matches two or more strings by insertion or deletion. If two aligned symbols are equal, it is an identity; otherwise, it is a mismatch. An insertion or deletion (indel) is one or more letters aligned against "$-$". There can be many possible alignments for a given string set. For example, two possible alignments of strings *abcdcdb* and *abbcdb* are shown below.

$$\begin{array}{ccccccc} a & b & c & d & c & d & b \\ a & b & b & c & d & b & - \end{array} \qquad \begin{array}{cccccccc} a & b & - & c & d & c & d & b \\ a & b & b & c & d & - & - & b \end{array}$$

A good alignment is one with the fewest mismatches and indels. Suppose that each mismatch or indel is penalized with one point. We can use dynamic programming to minimize the distance of two strings (a.k.a editing-distance problem), and get the corresponding alignment via backtracking.

To align more than two strings, we adopt the progressive alignment approach. Given k strings, align any two of the k strings and replace them with the resulting pairwise alignment. This gives an alignment problem with $k-1$ strings. Repeat the process until only one k-way alignment remains.

4.2 Inducing Pattern from an Alignment

Now, we are ready to introduce how to induce a template expression from an alignment of k strings. The pattern must match each of the k training strings, and should be general enough to cover unseen strings. We assume the resulting template is represented in *restricted* regular expression without the notion of disjunction.

A k-string alignment $s = s_1 s_2 \ldots s_n$ of length n, where $(1 \le i \le n)$, can be viewed as an array with k rows. Each column denotes k symbols being aligned into t_i in the result pattern $\hat{\mu}(s) = t_1 t_2 \ldots t_n$.

$$t_i = \begin{cases} \mu(s_i)^? & \text{if } - \in s_i, \\ \mu(s_i) & \text{otherwise.} \end{cases}$$

The mapping function μ is defined to output a symbol if if all input symbols are the same; or to output a more general symbol (e.g. a wildcard $*$) if there's a mismatch. The details of the mapping function μ will be described in Section 4.3. When a column contains the dash symbol, i.e. an indel during alighment, it indicates one or more sequences contain no symbol at the position. The corresponding symbol in the result pattern must be optional. For example, consider the alignment shown below.

$$\begin{array}{cccccccc} a & b & - & c & d & c & d & b \\ a & b & b & c & d & - & - & b \end{array}$$

The generated pattern the alignment should be

$$a \quad b \quad b^? \quad c \quad d \quad c^? \quad d^? \quad b$$

Note that we can further combine some adjacent symbols to generalize the pattern. For example, we can transfer $bb^?$ to b^+, $c^?d^?$ to $(cd)^?$ and $cd(cd)^?$ to $(cd)^+$, and the final pattern becomes $ab^+(cd)^+b$. Such combination of adjacent sub-patterns can only be done if the substituted pattern cover the same input strings. For example, $bb^?$ can be substituted by b^+, but b cannot be replaced by $b^?$. Consider the case for replacing $c^?d^?$ with $(cd)^?$ in the above example, where $c^?$ corresponds to $\left(\begin{smallmatrix} c \\ - \end{smallmatrix}\right)$ in the original alignment, and $d^?$ is derived from $\left(\begin{smallmatrix} d \\ - \end{smallmatrix}\right)$. The dash symbol appears in the same string in both columns, which indicates that the two columns are highly correlated.

Let r, s and t be three patterns. We designed Algorithm 1 for combining two patterns by replacing $r^?s^?$ with $(rs)^?$ and Algorithm 2 that replaces $r^n s^m$ with t^l, where $n, m, l \in \{?, +\} \cup N$.

Algorithm 1 Combine two optional patterns.

```
CombineOption(patterns $r^?$ and $s^?$ from a $k$-string alignment)
 1: $c_1 \ldots c_m \leftarrow$ corresponding columns of $r$ in the alignment
 2: $c_{m+1} \ldots c_n \leftarrow$ corresponding columns of $s$ in the alignment
 3: for each $1 \le i \le k$ do
 4:    if $\exists_{j,l \le n}(c_{ji} = -) \wedge (c_{li} \ne -)$ then /*cannot be combined*/
 5:       return pattern $r^?s^?$
 6: return pattern $(rs)^?$
```

Algorithm 2 Combine two patterns.

```
CombinePattern(two pattern: $r^n$ and $s^m$)
 1: if literal features of $r$ and $s$ are not identical or collisions happen then
 2:    return the pattern $r^n s^m$
 3: $a_1^{r_1} a_2^{r_2} \ldots a_n^{r_n} \leftarrow$ pattern of $r$ by removing parentheses
 4: $a_1^{s_1} a_2^{s_2} \ldots a_n^{s_n} \leftarrow$ pattern of $s$ by removing parentheses
 5: for each $1 \le i \le n$ do
 6:    $t_i \leftarrow \begin{cases} r_i & \text{if } r_i = s_i, \\ ? & \text{else if } \{r_i, s_i\} \in \{\{?, 1\}, \{?, ?\}\}, \\ ?+ & \text{else if } \{r_i, s_i\} \in \{\{?, +\}, \{?, n\}\}, \\ + & \text{otherwise.} \end{cases}$
 7: $l \leftarrow \begin{cases} n+m & \text{if } n, m \text{ are natural number,} \\ ?+ & \text{else if } n = m = ?, \\ + & \text{otherwise.} \end{cases}$
 8: $t \leftarrow a_1^{t_1} a_2^{t_2} \ldots a_n^{t_n}$ with parentheses added back
 9: return pattern $(t)^l$
```

For any given alignment, there are many possible patterns. To find simpler and shorter pattern, we define the cost function η of a pattern p as

$$\eta(p) = \text{number of ? and + in } p.$$

That is, the complexity of a pattern is measured by the number of operations ? and $+$. The length of a pattern is defined as the number of symbols. For example, the length of a^2 is 1 and the length of aa is 2. For patterns with the same η value, the shortest pattern is preferred. The cost function of an alignment s denotes the smallest cost of the corresponding patterns. For example, given $s = \left(\begin{smallmatrix} a & b & c \\ & & c \end{smallmatrix}\right)$, we have $\eta(s) = \eta((ab)^?c) = 1$.

Algorithm 3 presents a dynamic-programming approach to computing the pattern with the smallest η cost from a given alignment. Given an alignment $s = s_1 s_2 \ldots s_n$, we define

$$G_{i,j} = \eta(s[i,j]).$$

to be the smallest cost of the sub-alignment $s[i,j] = s_i s_{i+1} \ldots s_j$ of s. By keeping all possible patterns in each lattice when computing the G matrix, the patterns with the smallest η cost is obtained from computing $\eta(s) = G_{1,n}$.

Algorithm 3 Induce a pattern from a given alignment.

```
InducePattern(an alignment $s = s_1 s_1 \ldots s_n$)
 1: $P, G \leftarrow$ two $n \times n$ matrices
 2: for each $1 \le i \le n, i \le j \le n$ do
 3:    $P_{i,j} \leftarrow \{\hat{\mu}(s[i,j])\}$
 4:    $G_{i,j} \leftarrow \eta(\hat{\mu}(s[i,j]))$
 5: for each $1 \le l \le n-1, 1 \le i \le n-l$ do
 6:    $j \leftarrow i + l$
 7:    for each $i \le k \le j-1$ do
 8:       for all $r \in P_{i,k} \wedge s \in P_{k+1,j}$ do
 9:          for all $p \in$ {CombineOption($r$, $s$),CombinePattern($r$, $s$)} do
10:             if $\eta(p) < G_{i,j}$ then
11:                $G_{i,j} \leftarrow \eta(p)$
12:                $P_{i,j} \leftarrow \{p\}$
13:             else if $\eta(p) = G_{i,j}$ then
14:                $P_{i,j} \leftarrow P_{i,j} \cup \{p\}$
15: return the shortest pattern in $P_{1,n}$
```

4.3 Generating Template

The proposed template generation algorithm is straightforwardly top-down, constructing a template tree from the root and going down one level at a time:

1. Generate an appropriate node n for the root of each training document tree.

2. Generate the children of node n via the restricted regular expression induction.

3. Repeat steps 1–2 for all the subtrees.

To generate a template node from a set of training nodes is to find a feasible label matching the set of training node labels. If all training node labels are the same, it is the output label. If some of them are different, a wildcard label $*$ can be used. If some training nodes are inserted spurious nodes (it's labeled as $\diamond$ and inserted by the labeling process, stated in next section, to group consecutive nodes), a compound node should be used.

4.4 An Illustrative Example

Let's go through a brief example to illustrate the template generation process. Figure 4 shows two search result pages from Yahoo! for queries "Wrapper" and "Star Wars." For

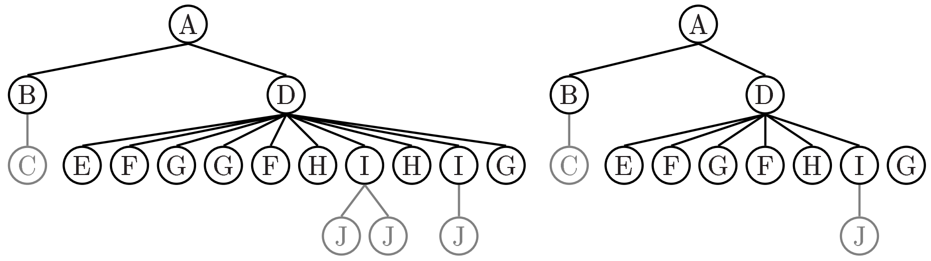
```
<html>
<head><title>Yahoo!  Search Results</title></head>
<body>
<table> An Advertisement Image </table>
<center> Yahoo!  Category Matches </center>
<p> Recreation > Hobbies > Collecting > Food </p>
<p> Computers > Software > Programming Tools </p>

<center> Yahoo!  Site Matches </center>
<p> Recreation > Hobbies > Collecting > Food </p>
<ul>
<li>Jake's Wrapper Collection
<li>Quasi Comprehensive Candy Bar Wrapper Image Archive
</ul>
<p> Computers > PLs > Tcl/TK > Scripts </p>
<ul>
<li>SWIG-Simplified Wrapper and Interface Generator
</ul>
<p> Next 20 Matches </p>
</body>
</html>
```

```
<html>
<head><title>Yahoo!  Search Results</title></head>
<body>
<table> An Advertisement Image </table>
<center> Yahoo!  Category Matches </center>
<p> Entertainment > Movies and Film > Star Wars </p>

<center> Yahoo!  Site Matches </center>
<p> Entertainment > Movies and Film > Star Wars </p>
<ul>
<li>Star Wars Cantina Chat - Star Wars Chat
</ul>
<p> Next 20 Matches </p>
</body>
</html>
```

Figure 4. Two training Web pages.



Symbol	Tag	Concepts
A	<html>	{Category,SiteList}
B	<head>	{}
C	<title>	{}
D	<body>	{Category,SiteList}
E	<table>	{}
F	<center>	{}
G	<p>	{}
H	<p>	{Category}
I	<ul>	{SiteList}
J	<li>	{}

Figure 5. Two training trees represented by node symbols.

the purpose of illustration, the two pages have been simplified, while preserving the skeleton structure of the original pages.

Suppose that the target information to extract is each two-fold entry (highlighted by rectangle boxes in Figure 4): a category indicator, labeled as Category, and a list of link items, as SiteList. By propagating the block names from target nodes to the root, we construct the symbol of each node, consisting of a syntactic part (i.e., a block tag) and the block names contained in the corresponding node. Figure 5 shows the trees obtained after labeling (without showing the text-string nodes). The node <html> contains the blocks Category and SiteList, and the node <ul> contains only the block SiteList. The gray nodes are those marked as superfluous after pruning.

First, start by generating a node to match the roots of training trees. In this example, an exact node with label <html> is appropriate. The next step is to generate the children of the resulting template node. The two training sequences are exactly identical, so it is straightforward to generate the child nodes as <head><body>. The subtree rooted at <head> contains no target information. Thus we proceed to generate the subtree rooted at <body>. Now the two training sequences are $EFGGFHIHIG$ and $EFGFHIG$. The pattern $EFG^{+}F(HI)^{+}G$ is generated. The same process is repeated for the nodes <p> and <ul> at 2nd level of the result sequence because they contain the

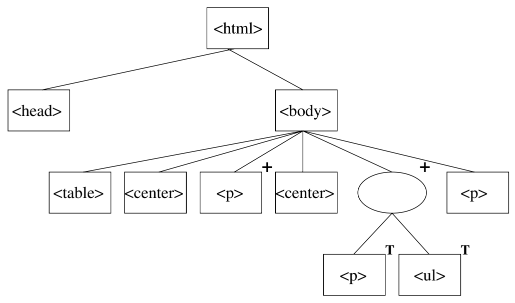


Figure 6. The final template tree.

target information. Since they are the target nodes (we indicate each of them with a superscript T), we terminate the process. Figure 6 presents the final template tree.

5 Experiment

We evaluated our approach on the following two types of information sources: (1) the query-based, e.g., search engines and online shopping stores, and (2) the periodical, e.g., news portals. The reason for choosing these two types was that their result pages were very likely generated by machines, and thus there might exist internal document templates. For query-based sites, we gathered the responses for 50 sample queries. The queries were chosen according

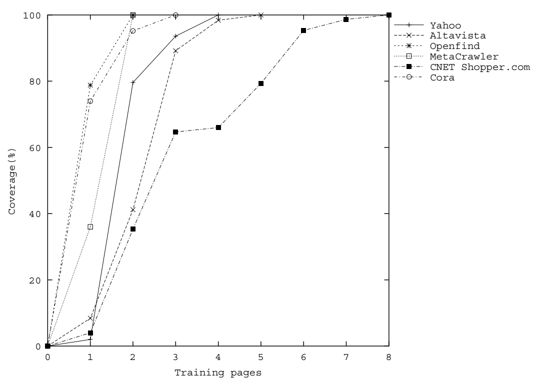


Figure 7. The generation curve on query-based information resources.

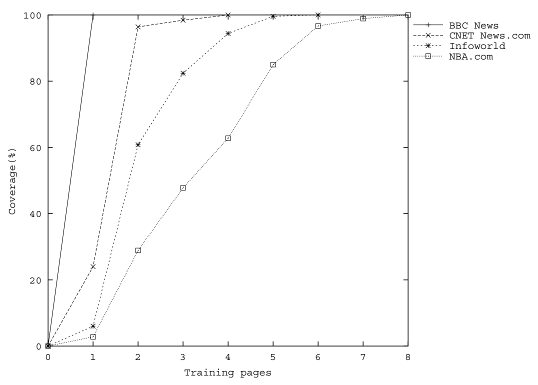


Figure 8. The generation curve on periodical information resources.

to each specific site. For example, the queries for Shopper.com were all computer products, e.g., "Microsoft Office" and "IBM Thinkpad." All response pages of these information sites were lists of items. The target information to be extracted was each individual item. For periodical news information, we collected 50 daily front pages for each site. The extraction targets were the news headlines.

There are two issues to concern: whether a general enough template can be generated for unseen pages and how many training pages are required to produce such a template. The experiment was conducted as follows. For each site, we randomly chose a page for training. The chosen page was labeled and then fed to the template generation module. The resulting template was used to divide the whole sample pages into two sets: a successful set containing the pages whose items could be extracted, and a failed set of pages that couldn't be successfully handled. If the failed set was not empty, a page was randomly chosen from that set, labeled and then added to the training set, according to which a new template was generated. The process was repeated, until all sample pages were successfully handled or there existed pages that couldn't be handled anyhow. To obtain a more reliable performance estimation, the entire process was performed five times for each testing site. The final curves on the coverage rates, i.e., the percentage of the pages in the successful set over the entire testing pages, with respect to the number of training pages for the two types of testing sites are shown in Figure 7 and 8, respectively.

The experimental results show that the proposed template generation approach can successfully handle the surveyed sites. In general, if the pages of the same resource have some deviation, i.e., different number of target items, multiple training pages are required. In our experiment, only the news site BBC requires one page to produce an effective template for extracting its headlines. This is because the front pages of that site are very uniform in their structure. All the other sites need multiple, but few, training instances to produce effective templates.

However, we also found several drawbacks of our approach on actual Internet resources during the experiment. The Web pages are generally designed for being displayed via the Web browsers. The page designers intend to verify the correctness of their HTML pages via the browsers, which have high tolerance on syntactic errors of HTML documents. Even more, some tags are usually misused. For example, <p> is usually used as a wilder blank line, rather than an indicator of a paragraph. The misusing of tags makes that the structure constructed according to markup tags doesn't reflect the structure intended by the author of documents. This motivates the need of a sophisticated parsing tool to construct more accurate document trees.

To have a comparison, we also gave our surveyed sites a try to MDR[1] [14], since their idea of using tag trees was similar to ours and their experiment showed that MDR outperformed other approaches, e.g., IEPAD [4]. For each site, we selected one testing page and fed it to MDR system. Because MDR was an automatic approach, we counted it as correct if one of its output data regions contained the data records that we extracted using our templates. Unfortunately, only the news site BBC could be correctly handled by MDR. The processing of sites NBA.com and MetaCrawler ended up with unexpected fatal program termination. The output of the remaining seven sites mostly contained data regions for navigation or advertisement. The main reason was that the current implementation of MDR was limited to mining data records formed by table-related tags. But most of the data records we wanted to extract from our surveyed sites were not formed by table-related tags (see Figure 4 for an example). Also HTML table had been over-used in designing fancy page layout; most of navigational bars or advertisment lists were arranged using ta-

[1]`http://www.cs.uic.edu/~liub/MDR/MDR-download.html`

ble tags. While MDR did not perform well in our testing, there's not enough evidence to conclude whether TTAG is better. Further comparison and evaluation of the proposed approach with competing methods should be done for more informatin sources.

6 Concluding Remarks

In this paper, we propose a tree-template-based approach to modeling the global structural layout of Web pages according to the nested use of markup tags. Using tree-structured templates has the advantage of preserving the hierarchical structure context of the fields in a document, which may be lost by using linear-string-based approaches. To reduce the tedious labor in coding templates, an automatic template generation algorithm is designed and well engineered. We investigate its possibility by an initial application to Web information extraction, and the experiment results show its effectiveness. Further work is needed to extend the proposed method to XML documents and other semi-structured information sources.

References

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from Web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 337–348, 2003.

[2] N. Ashish and C. A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *Proceedings of the Second IFCIS International Conference on Cooperative Information Systems*, pages 160–169. Kiawah Island, CA, USA, 1997.

[3] D. Buttler, L. Liu, and C. Pu. A fully automated extraction system for World Wide Web. In *Proceedings of the 2001 International Conference on Distrubuted Computing Systems*, pages 361–370, Phoenix, Arizona, May 2001.

[4] C.-H. Chang and S.-L. Lui. IEPAD: Informatin extraction based on pattern discovery. In *Proceedings of the 10th International World Wide Web Conference*, 2001.

[5] S.-L. Chuang. Automatic generation of tree-structured templates for information extraction from html documents. Master's thesis, National Taiwan University, June 1999.

[6] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large Web sites. In *Proceedings of the 2001 International Conference on Very Large Data Bases*, 2001.

[7] D. Embley, S. Jiang, and Y. Ng. Record-boundary discovery in Web documents. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 467–478, Philadelphia, USA, 1999.

[8] T. Guan and K.-F. Wong. KPS: a Web information mining algorithm. *Computer Networks*, 31:1495–1507, 1999.

[9] D. Gusfield. *Algorithms on strings, tree, and sequence*. Cambridge, 1997.

[10] C. Hsu. Generating finite-state transducers for semi-structured data extraction from the Web. *Information Systems*, 23(8):521–538, 1998.

[11] J. Y. Hsu and W.-T. Yih. Template-based information mining from html documents. In *Proceedings of AAAI-97*, pages 256–262, July 1997.

[12] R. Kosala, J. V. den Bussche, M. Bruynooghe, and H. Blockeel. Information extraction in structured documents using tree automata induction. In *Proceedings of the the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*. Helsinki, Finland, 2002.

[13] N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 729–737, 1997.

[14] B. Liu, R. Grossman, and Y. Zhai. Mining data records in Web pages. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[15] I. Muslea. Extraction patterns for information extraction: A survey. In *Proceedings of the AAAI 1999 Workshop on Machine Learning for Information Extraction*, 1999.

[16] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 4:93–114, 2001.

[17] Y. Wang and J. Hu. A machine learning based approach for table detection on the Web. In *Proceedings of the 11st International World Wide Web Conference*, 2002.

[18] K. Zhang, D. Shasha, and J. Wang. Approxmiate tree matching in the presence of variable length don't care. *Journal of Algorithms*, 16:33–66, 1994.