# Applying Pattern Mining to Web Information Extraction

Chia-Hui Chang, Shao-Chen Lui, and Yen-Chin Wu

Dept. of Computer Science and Information Engineering
National Central University, Chung-Li, 320, Taiwan
chia@csie.ncu.edu.tw, {anyway, trisdan}@db.csie.ncu.edu.tw

**Abstract.** Information extraction (IE) from semi-structured Web documents is a critical issue for information integration systems on the Internet. Previous work in *wrapper induction* aim to solve this problem by applying machine learning to automatically generate extractors. For example, WIEN, Stalker, Softmealy, etc. However, this approach still requires human intervention to provide training examples. In this paper, we propose a novel idea to IE, by repeated pattern mining and multiple pattern alignment. The discovery of repeated patterns are realized through a data structure call *PAT tree*. In addition, incomplete patterns are further revised by pattern alignment to comprehend all pattern instances. This new track to IE involves no human effort and content-dependent heuristics. Experimental results show that the constructed extraction rules can achieves 97 percent extraction over fourteen popular search engines.

**Keywords:** information extraction, semi-structured documents, wrapper generation, pattern discovery, multiple alignment

## 1 Introduction

*Information extraction* (IE) is concerned with extracting from a collection of documents the information relevant to a particular extraction task. For instance, the meta-search engine MetaCrawler extracts the search results from multiple search engines; and the shopping agent Junglee extracts the product information from several online stores for comparison. With the growth of the amount of online information, the availability of robust, flexible IE has become a stringent necessity.

Contrast to "traditional" information extraction which roots in natural language processing (NLP) techniques such as linguistic analysis, Internet information extraction rely on syntactic structures identification marked by HTML tags. The difference is due to the nature of Web such that the page contents have to be clear at glance. Thus, "itemized list" and "tabular format" have been the main presentation style for Web pages on the Internet. Such presentation styles together with the multiple records contained in one documents contribute the so called semi-structured Web pages.

The major challenge of IE is the problem of scalability as the extraction rules must be tailored for each particular page collection, To automate the construction of extractors (or wrappers), recent research has identified important wrapper classes and induction algorithms. For example, Kushmerick et. al. identified a family of wrapper classes and the corresponding induction algorithms which generalize from labeled examples to extraction rules [9]. More expressive wrapper structure are introduced lately. *Softmealy* by Hsu and Dung [6] uses a wrapper induction algorithm to generate extractors that are expressed as finite-state transducers. Meanwhile, Muslea et al. [10] proposed *"STALKER"* that performs hierarchical information extraction to redeem Softmealy's inability to use delimiters that do not immediately precede and follow the relevant items with extra scans over the documents (see [11] for a complete survey).

In all this work, wrappers are induced from training examples such that landmarks or delimiters can be generalized from common prefixes or suffixes. However, labeling these training examples is sometimes time-consuming. Hence, another track of research is exploring new approaches to fully automate wrapper construction. For example, Embley et. al. describe a heuristic approach to discover record boundaries in Web documents by identifying *candidate separator tags* using five independent heuristics and selecting a consensus separator tag based on a heuristic combination [3]. However, one serious problem in this one-tag separator approach arises when the separator tag is used elsewhere among a record other than the boundary.

On the other hand, our work here attempts to eliminate human intervention by pattern mining. The motivation is from the observation that useful information in a Web page is often placed in a structure having a particular alignment and order. For example, Web pages produced by Web search engines generally have regular and repetitive patterns, which usually represent meaningful and useful data records. In the next section, we first give an example showing the repeated pattern formed by multiple aligned records.

## 2 Motivation

One observation from Web pages is that the information to be extracted is often placed in a structure having a particular alignment and forms *repetitive patterns*. For example, query-able or search-able Internet sites such as Web search engines often produce Web pages with large itemized match results which are displayed in a particular template format. The template can be recognized when the content of each match is ignored or replaced by some fixed-length string. Therefore, repetitive patterns are formed. For instance, in the example of Figure 1, the sequence "<LI>Text(_)<I>Text(_)</I>" is repeated four times, when all text strings between two tags such as "Congo", "Egypt", "Belize" etc. are replaced by token class *Text(_)*.

This is a simple example that demonstrates a repeated pattern formed by tag tokens in a Web page following a simple translation convention. In practice, many search-able Web sites also exhibit such repeated patterns since they

<H1>Country Code</H1><UL>
<LI>Congo<I>242</I>
<LI>Egypt<I>20</I>
<LI>Belize<I>501</I>
<LI>Spain<I>34</I>
</UL>

**Fig. 1.** Sample HTML page

usually extract data from relational database and produce dynamic Web pages with a predefined format style. Therefore, what we ought to do is kind of reverse engineering to discover the original format style and the content we need to extract. Meanwhile, we also find that extraction patterns of the desired information (called *main information block* as defined in [3]) often occur regularly and closely in a Web page. These observations motivate us to look for an approach to discover repeated patterns and validation criteria to filter desired repeats that are spaced regularly and closely.

Since HTML tags are the basic components for data presentation and the text string between tags are exactly what we see in the browsers. Hence, it is intuitive to regard the text string between two tags as one unit as well as each individual tag. This simple version of *HTML translation* will be used in the following paper where any text string between two tags is translated to one unit called Text(_) and every HTML tag is translated to a token Html(<tag>) according to its tag name.

Such translation convention enables the show-up of many repeated patterns. By *repeated patterns*, we mean any substring that occurs twice in the encoded token string. Thus, not only the sequence "Html(<LI>) Text(_) Html(<I>) Text(_) Html(<I>)" conforms to the definition of repeated pattern but also the subsequence "Html(<LI>) Text(_) Html(<I>)," "Text(_) Html (<I>) Text(_)", "HMLT(<I>)Text(_)Html(</I>)," etc. To distinguish from these repeats, we define *maximal repeats* to uniquely identify the longest pattern as follows.

**Definition** Given an input string $S$, we define maximal repeat $\alpha$ as a substring of $S$ that occurs in $k$ distinct positions $p_1, p_2, ..., p_k$ in $S$, such that the ($p_i$-1)th token in $S$ is different from the ($p_j$-1)th token for at least one $i, j$ pair, $1 \leq i < j \leq k$ (called left maximal), and the $(p_x + |\alpha|)$th token is different from the $(p_y + |\alpha|)$th token for at least one $x, y$ pair, $1 \leq x < y \leq k$ (called right maximal).

The definition of maximal repeats is necessary for identifying the well-used and popular term, repeats. Besides, it also captures all interesting repetitive structures in a clear way and avoids generating overwhelming outputs. In the next section, we will describe how the problem of IE can be addressed by pattern discovery.

## 3 IE by Pattern Discovery

To discover patterns from an input Web page, first an encoding scheme is used to translate the Web page into a string of abstract representations, referred to here as tokens. Each token is represented by a binary code of length $l$. To enable pattern discovery, we utilizes a data structure called a *PAT tree* [4] in which repeated patterns in a given sequence can be efficiently identified. Using this data structure to index an input string, all possible repeats, including their occurrence counts and their positions in the original input string can be easily retrieved. Finally, the discovered maximal repeats are forwarded to the validator, which filters out undesired patterns and to produces a candidate pattern.

### 3.1 Translator

Since HTML tags are the basic components for document presentation and the tags themselves carry a certain structure information, it is intuitive to examine the tag token string formed by HTML tags and regard other non-tag text content between two tags as one single token called Text(_). Tokens seen in the translated token string include tag tokens and text tokens, denoted as Html(<tag_name>) and Text(_), respectively. For example, Html(</a>) is a tag token, where </a> is the tag. Text(_) is a text token, which includes a contiguous text string located between two HTML tags.

Tags tokens can be classified in many ways. The user can choose a classification depending on the desired level of information to be extracted. For example, tags in the BODY section of a document can be divided into two distinct groups: block-level tags and text-level tags. The former defines the structure of a document, and the latter defines the characteristics, such as format and style, of the contents of the text. Block level tags include categories such as headings, text containers, lists, and other classifications, such as tables and forms. Text-level tags are further divided into categories including logical markups, physical markups, and special markups for marking up texts in a text block.

The many different tag classifications allow different HTML translations to be generated. With these different abstraction mechanisms, different patterns can be produced. For example, skipping all text-level tags will result in higher abstraction from the input Web page than all tags are included. In addition, different patterns can be discovered and extracted when different encoding schemes are translated.

For example, when only block-level tags are considered, the corresponding translation of Fig. 1 is a token string: "Html(<H1>)Text(_)Html(</H1>) Html(<UL>)Html(<LI>) Text(_)Html(<LI>)Text(_)Html(<LI>)Text(_)Html(<LI>) Text(_) Html(</UL>)", where each token is encoded as a binary strings of "0"s and "1"s with length $l$. For example, suppose three bits encode the tokens in the Congo code as shown in Fig. 2. The encoded binary string for the token string of the Congo code will be "100110 101000 010110 010110 010110 010110 001$" of 3*13 bits, where "$" represents the ending of the encoded string.

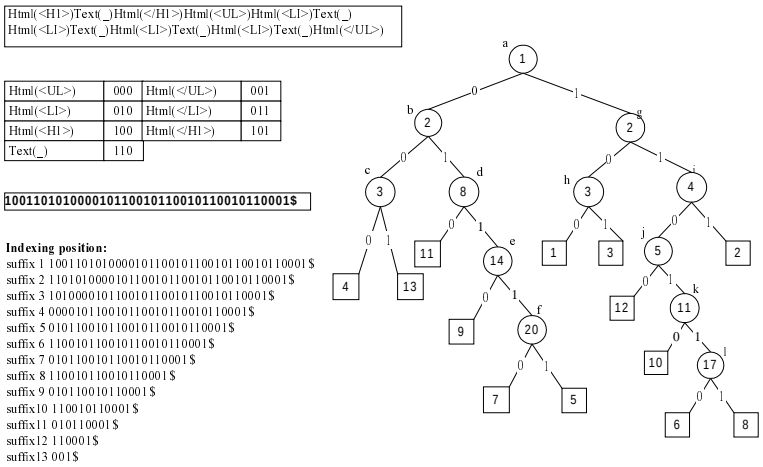


**Fig. 2.** The PAT tree for the Congo Code

### 3.2 The PAT Tree

Our approach for pattern discovery uses a PAT tree to discover repeated patterns in the encoded token string. A PAT tree is a Patricia tree (Practical Algorithm to Retrieve Information Coded in Alphanumeric [11]) constructed over all the possible suffix strings. A Patricia tree is a particular implementation of a compressed binary (0,1) digital tree such that each internal node in the tree has two branches: zero goes to left and one goes to right. Like a suffix tree [4], the Patricia tree stores all its data at the external nodes and keeps one integer, the bit-index, in each internal node as an indication of which bit is to be used for branching. For a character string with $n$ indexing point (or $n$ suffix), there will be $n$ external nodes in the PAT tree and $n-1$ internal nodes. This makes the tree $O(n)$ in size.

When a PAT tree is to index a sequence of characters (or tokens here) not just 0 or 1, the binary codes for the characters can be used. For simplicity, each character is encoded as fixed-length binary code. Specifically, given a finite alphabet $\Sigma$ of a fixed size, each character $x \in \Sigma$ is represented by a binary code of length $l = \lceil \log_2 |\Sigma| \rceil$. For a sequence $S$ of $n$ characters, the binary input $B$ will have $n * l$ bits, but only the $[i * l + 1]$th bit has to be indexed for $i = 0, \ldots, n-1$.

Referring to Fig. 2, a PAT tree is constructed from the encoded binary string of the Congo example. The tree is constructed from thirteen sequences of bits, with each sequence of bits starting from each of the encoded tokens and extending to the end of the token string. Each sequence is called a "semi-infinite string" or "sistring" in short. Each leaf, or external node, is represented by a square labeled by a number that indicates the starting position of a sistring. For example, leaf 2 corresponds to sistring 2 that starts from the second token in the token string.

Each internal node is represented by a circle, which is labeled by a bit position in the encoded bit string. The bit position is used when locating a given sistring in PAT tree.

Virtually, each edge in the PAT tree has a edge label. For example, the edge labels between node $d$ and $e$ are "101100", the 8th bit to 13th bit for suffix 9, 7, and 5. Edges that are visited when traversing downward from root to a leave form a path that leads to a sistring corresponding to the leave. The concatenated edge labels along the path form a virtual path label. For example, the edge labels "1", "10", and "1..." on the path that leads from root to leave 2 form a prefix "1101...", which is a unique prefix for sistring 2.

As shown in Fig. 2, all suffix strings with the same prefix will be located in the same subtree. Hence, it allows surprisingly efficient, linear-time solutions to complex string search problems. For example, string prefix searching, proximity searching, range searching, longest repetition searching, most frequent searching, etc. [4, 5] Since every internal node in a PAT tree indicates a branch, it implies a different bit following the common prefix between two suffixes. Hence, the concatenation of the edge-labels on the path from the root to an internal node represents one repeated string in the input string. However, not every path-label or repeated string represents a maximal repeat. Let's call the $(p_k - 1)$th character of the binary string $p_k$ the *left character*. For a path-label of an internal node $v$ to be a maximal repeat, at least two leaves (suffixes) in the $v$'s subtree should have different left characters. By recording the occurrence counts and the reference positions in the leaf nodes of a PAT tree, we can easily know how many times a pattern is repeated. Hence, given the pattern length, occurrence count, we can apply postorder traversal to the PAT tree to enumerate all repeats.

The essence of a PAT tree is a binary suffix tree, which has also been applied in several research field for pattern discovery. For example, Kurtz and Schleiermacher have used suffix trees in bioinformatics for finding repeated substring in genomes [8]. As for PAT trees, they have been applied for indexing in the field of information retrieval since a long time ago [4]. It has also been used in Chinese keyword extraction [1] for its simpler implementation than suffix trees and its great power for pattern discovery. However, in the application of information extraction, we are not only interested in repeats but also repeats that appear regularly in vicinity. Discovered maximal repeats have to be further validated or compared to find the best one that corresponds to the information to be extracted.

### 3.3 Pattern Validation Criteria

In the above section, we discussed how to find maximal repeats in a PAT tree. However, there may be over 60 maximal repeats discovered in an Web page. To classify these maximal repeats, we introduce two measures regularity, and compactness as described below. Let the suffixes of a maximal repeat $\alpha$ are ordered by its position such that suffix $p_1 < p_2 < p_3 \ldots < p_k$, where $p_i$ denotes the position of each suffix in the encoded token sequence.

**Regularity** of a pattern is measured by computing the standard deviation of the interval between two adjacent occurrences $(p_{i+1} - p_i)$, that is, the sequence of spacing between two adjacent occurrences $(p_2 - p_1)$, $(p_3 - p_2)$, ..., $(p_k - p_{k-1})$. Regularity of the maximal repeat $\alpha$ is equal to the standard derivation of the sequence divided by the mean of the sequence.

**Compactness** is a measure of the density of a maximal repeat. It is used to eliminate maximal repeats that are scattered far apart beyond a given bound. Compactness is defined as $k * |\alpha| / \sum_{i=2}^{k} p_i - p_{i-1}$, where $|\alpha|$ is the length of $\alpha$ in number of tokens.

The value of regularity is located between 0 and 1 while the value of density is greater than 0. Ideally, the extraction pattern should have regularity equal to zero and compactness equal to one. To filter potentially good patterns, a simple approach will be to use a threshold for each of these measures above. Implicitly, good patterns have small regularity and density close to one. Therefore, only patterns with regularity less than the regularity threshold and density between the density thresholds are considered validated patterns.

## 4 Performance Evaluation

We first show the number of validated maximal repeats validated by our system using fourteen state-of-the-art search engines, each with ten Web pages. There are several control parameters which can affect the number of maximal repeats validated, including encoding scheme, minimum pattern length, occurrence count, and threshold values for regularity and compactness. Given the minimum length 3 and count 5, the effect of different encoding scheme is shown in Table 1. Conform to general expectation, higher-level encoding scheme often results in less patterns. From this table, we can also see how each control parameter filters patterns where the thresholds are decided by the following experiments. The value of density can be greater than one because maximal repeats may be overlapped. For example, suppose a maximal repeat $\alpha$ occurs ten times in a row. In such case, $\alpha$ will has regularity 0 and density 1. In addition, $\alpha\alpha$, $\alpha\alpha\alpha$, etc. are also qualified for regular maximal repeats, only with density greater than 1.

**Table 1.** No. of Patterns validated with different encoding scheme

| Encoding Scheme | Maximal Repeat | Regularity $< 0.5$ | Compactness $> 1.5$ | Compactness $< 0.25$ |
|---|---|---|---|---|
| All-tag | 117 | 39 | 22 | 7.6 |
| NoPhysical | 88 | 41 | 25 | 6.5 |
| NoSpecial | 82 | 29 | 18 | 5.7 |
| Block-level | 66 | 32 | 17 | 3.9 |

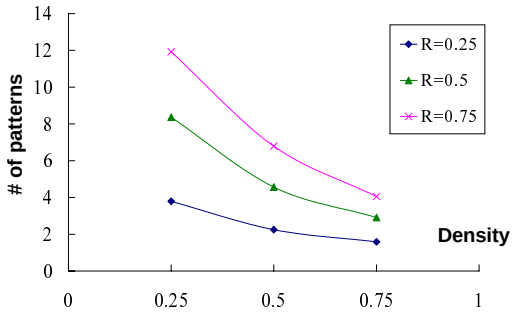


**Fig. 3.** # of patterns sucessfully validated

Fig. 3 shows the effect of various regularity and density thresholds using all-tag encoding scheme. Basically, low regularity threshold and high density threshold reduce the number of patterns, but could have missed good patterns. Therefore, the thresholds are chosen empirically to include as many good patterns as possible.

Table 2 shows the performance of different encoding scheme measured in retrieval rate, accuracy rate and matching percentage. Retrieval rate is defined as the ratio of the number of desired data records enumerated by a maximal repeat to the number of desired data records contained in the input text. Likewise, accuracy rate is defined as the ratio of the number of desired data records enumerated by a maximal repeat to the number of occurrence of the maximal repeat. A data record is said to be *enumerated* by a maximal repeat if the matching percentage is greater than a bound determined by the user. The matching percentage is used because the pattern may contain only a portion of the data record.

With the simple encoding scheme of using block-level tags, our approach could discover patterns which extract 86% records with matching percentage 78%. Nearly half the test Web sites are correctly extracted (with matching percentage greater than 90%). Among them, nine of the fourteen Web sites have retrieval rate and accuracy rate both greater than 0.9. However, examining other discovered patterns, many are incomplete due to exceptions. In the next section, we will further improve the performance by occurrence partition and multiple string alignment.

## 5 Constructing Extraction Pattern

Generally speaking, search engines utilize a "while loop" to output their results in some template. However, they may use "if clauses" inside the while loop to decorate the text content. For example, the keywords that are submitted

**Table 2.** Performance of different encoding scheme

| Encoding Scheme | Retrieval Rate | Accuracy Rate | Matching Percentage |
|---|---|---|---|
| All-tag | 0.73 | 0.82 | 0.60 |
| NoPhysical | 0.82 | 0.89 | 0.68 |
| NoSpecial | 0.84 | 0.88 | 0.70 |
| Block-level | 0.86 | 0.86 | 0.78 |

to search engines are shown in bold face for Infoseek and MetaCrawler, thus, breaking their "while loop" patterns.

From the statistics above, we summarize that "maximal repeat" and "regularity" are the two primary criteria we filter candidate patterns. However, we also found that the extraction pattern may not be maximal repeats and regular. For example, the regularity of the pattern for *Excite* is greater than default regularity threshold 0.5 because a banner is inserted among the search results, dividing the ten matches into two parts. Besides, the *"if-effect"* often hinders us from discovering complete patterns. These issues are what we would address in the following.

### 5.1 Occurrence Partition

To handle patterns with regularity greater than the specified threshold 0.5, these patterns are carefully segmented to see if any partition of the pattern's occurrences satisfies the requirement for regularity. By definition, the regularity of a pattern is computed through all occurrences of the pattern. For example, if there are $k$ occurrences, the $k-1$ intervals (between two adjacent instances) are the statistics we use to compute the standard deviation and the mean. However, in examples such as *Lycos*, the search result is divided into three blocks. Such occurrences increase the regularity over all instances. Nonetheless, the regularity of the occurrences in each information block is still small. Therefore, the idea here is to segment the occurrences into partitions so that we can analyze each partition individually.

We don't really have to apply clustering algorithm on this matter, instead, a simple loop can accomplish the job if the occurrences are ordered by their position aforehand. Let $C_{i,j}$ denotes the set of occurrences $p_i, p_{i+1}, ..., p_j$ and initialize $s = 1, j = 1$. For instance $p_{j+1}$, if the regularity of $C_{s,j+1}$ is greater than $\theta$ then output $C_{s,j}$ as a partition and assign $j+1$ to $s$.

Once the partitions are separated, we can then compute the regularity for each individual partition. If a partition includes occurrences more the minimum count and has regularity less than threshold $\epsilon$, the pattern as well as the occurrences in this partition are outputted. Note that the threshold $\epsilon$ is set to a small value much less than 0.5 to control the number of outputted patterns. With this modification, the performance is improved greatly. As shown in Table 3, the retrieval rate is increased to 93% and accuracy rate to 90%. The only tradeoff is the increased number of patterns from 3.9 to 8.9.

**Table 3.** Performance of advanced technique

| Advanced Technique | Retrieval Rate | Accuracy Rate | Matching Percentage |
|---|---|---|---|
| Occurrence Partition | 0.93 | 0.93 | 0.84 |
| Multiple Alignment | 0.97 | 0.94 | 0.90 |

### 5.2 Multiple String Alignment

For the tough work regarding incomplete pattern discovered, the technique for multiple string alignment is borrowed to find a good presentation of the critical common features of multiple strings. For example, suppose "adc" is the discovered pattern for token string "adcwbdadcxbadcxbcadc". If we have the following multiple alignment for strings "*adcwbd*", "*adcxb*" and "*adcxbd*":

$$\begin{array}{c} a\ d\ c\ w\ b\ d \\ a\ d\ c\ x\ b\ - \\ a\ d\ c\ x\ b\ d \end{array}$$

The extraction pattern can be *generalized* as "$adc[w|x]b[d|-]$" to cover these three instances. Specifically, suppose a validated maximal repeat has $k+1$ occurrence, $p_1$, $p_2$, ..., $p_{k+1}$ in the encoded token string. Let string $P_i$ denote the string starting at $p_i$ and ending at $p_{i+1}-1$. The problem is to find the multiple alignment of the $k$ strings $\mathcal{S} = \{P_1, P_2, ..., P_k\}$ so that the generalized pattern can be used to extract all records we need.

Multiple string comparison is a natural generalization of *alignment* for two strings which can be solved in $O(n * m)$ by *dynamic programming* to obtain optimal *edit distance*, where $n$ and $m$ are string lengths. As an example of *two string alignment*, consider the alignment of two strings *acwbd* and *adcxb* shown below:

$$\begin{array}{c} a\ -\ c\ w\ b\ d \\ a\ d\ c\ x\ b\ - \end{array}$$

In this alignment, character $w$ is mismatched with $x$, two $d$s are opposite hyphens (or called space), and all other characters match their counterparts in the opposite string. If we give each match a value of $\beta$, each mismatch a value of $\gamma$, and each space a value of $\delta$, the *two string alignment problem* is to optimize the *weighted distance* $D(P_1, P_2) \equiv (nmatch * \beta + nmis * \gamma + nspace * \delta)$, where $nmatch$, $nmis$, and $nspace$ denote the number of mismatch, match, and space, respectively ($nmatch = 3$, $nmis = 1$, and $nspace = 2$ here).

Extending dynamic programming to multiple string alignment yieds a $O(n^k)$ algorithm. Instead, an approximation algorithm is available such that the score of the multiple alignment is no greater than twice the score of optimal multiple alignment [5]. The approximation algorithm starts by computing the *center string* $S_c$ in $k$ strings $\mathcal{S}$ that minimizes *consensus error* $\sum_{P_i \in \mathcal{S}} D(S_c, P_i)$. Once the center string is found, each string is then iteratively aligned to the center string to construct multiple alignment, which is in turn used to construct the extraction pattern.

For each patterns with density less than one, the *center star* approximation algorithm for multiple string alignment is applied to generalize the extraction pattern. Suppose the generalized extraction pattern is expressed as "$c_1c_2c_3...c_n$", where each $c_i$ is either a symbol or a subset of $\Sigma \cup \{-\}$ containing symbols that can appear at position $i$. An additional step is taken to generate pattern of this form '$c_jc_{j+1}c_{j+2}...c_nc_1c_2...c_{j-1}$" for position $j$ with single symbol of the following special tags such as <DL>, <DT>, <TR> or <P>, <BR>, <HR>, because extraction patterns often begin or end up with them[1].

We adopt this additional step because the generated extraction pattern may not be the beginning of a record. The experimental results show that with the help of multiple string alignment and the additional step, the performance is improved to 97% retrieval rate, 94% accuracy rate and 0.90 matching percentage. The high percentage of retrieval rate is pretty encouraging. The ninety percent of matching percentage is actually higher in terms of the text content retrieved. For those Web sites with matching percentage greater than 85%, the text contents are all successfully extracted. What bothers is the accuracy rate, since the extraction pattern generalized from multiple alignment may comprehends more than the information we need. For example, the generalized rule for *Lycos* will extract information in all three blocks while only the information in one block is what we desired, causing lower accuracy rate.

## 6 Summary and Future Work

Information extraction from Web pages is a core technology for comparison-shopping agents [2], which Doorenbos et. al. regard as improvement in the axe of tolerating unstructured information. The characteristics of regularity, uniformity, and vertical separation enable the possibility of learning. In this paper, we have presented an unsupervised approach to semi-structured information extraction. We propose the application of PAT trees for pattern discovery in the encoded token string of Web pages. Once the PAT tree is constructed, we can easily traverse the tree to find all maximal repeats given the expected pattern frequency and length. The discovered maximal repeats are further filtered by three measures: regularity and compactness. The filtering criteria aim to keep the number of patterns as small as possible while at the same time have all interesting patterns. Furthermore, occurrence partition is applied to handle patterns with regularity greater than the default threshold. Finally, multiple string alignment is applied to patterns with density less than one to generalize extraction pattern. Thereby, the extraction module can simply adapt pattern matching algorithm to extract all records.

The extraction rule generalized from multiple string alignment has achieved 97% retrieval rate and 91% accuracy rate. The whole process requires no human intervention and training example. Comparing our algorithm to others, our approach is quick and expressive. It takes only three minutes to extract 140 Web

[1] Other tags include <TABLE>, <TD>, <UL>, <OL>, <LI>, <DD>.

pages. The extraction rule allowing alternative tokens and missing tokens, can tolerate exceptions and variance in the input.

We are currently applying this approach against more test data formatted in tabular form, which perform at the level of 80% retrieval rate. As more variances occur in input pages, it becomes even difficult to have good multiple string alignment. In such cases, the scoring of edit distance between two strings and the algorithm to construct multiple alignment become more important. In addition, filtering of the constructed patterns can also provide a reasonable number of patterns for user to choose.

## Acknowledgements

This work is sponsored by National Science Council, Taiwan under grant NSC89-2213-E-008-056. Also, we would like to thank Lee-Feng Chien, Ming-Jer Lee and Jung-Liang Chen for providing their PAT tree code for us.

## References

1. Chien, L.F. 1997. PAT-tree-based keyword extraction for Chinese information retrieval. In *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*. pp.50–58. 1997.
2. Doorenbos, R.B., Etzioni, O. and Weld, D. S. A scalable comparison-shopping agent for the World-Wide Web. In *Proceedings of the first international conference on Autonomous Agents*. pp. 39–48, NewYork, NY, 1997, ACM Press.
3. Embley, D.; Jiang, Y.; and Ng. Y.-K. 1999. Record-boundary discovery in Web documents. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*. pp. 467–478, Philadelphia, Pennsylvania.
4. Gonnet, G.H.; Baeza-yates, R.A.; and Snider, T. 1992. New Indices for Text: Pat Trees and Pat Arrays. *Information Retrieval: Data Structures and Algorithms*, Prentice Hall.
5. Gusfield, D. 1997. *Algorithms on strings, trees, and sequences*, Cambridge. 1997.
6. Hsu, C.-N. and Dung, M.-T. 1998. Generating finite-state transducers for semi-structured data extraction from the Web. *Information Systems*. 23(8):521–538.
7. Knoblock, A. et al., ed., 1998. *Proc. 1998 Workshop on AI and Information Integration*, Menlo Park, California.: AAAI Press.
8. Kurtz, S. and Schleiermacher, C. 1999. REPuter: fast computation of maximal repeats in complete genomes. *Bioinformatics* 15(5):426–427.
9. Kushmerick, N.; Weld, D.; and Doorenbos, R. 1997 Wrapper induction for information extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (IJCAI).
10. Muslea, I.; Minton, S.; and Knoblock, C. 1999. A hierarchical approach to wrapper induction. In *Proceedings of the 3rd International Conference on Autonomous Agents* (Agents'99), Seattle, WA.
11. Muslea, I. 1999. Extraction patterns for information extraction tasks: a survey. In *Proceedings of AAAI'99: Workshop on Machine Learning for Information Extraction*