Automatic Information Extraction from Semi-Structured Web Pages By Pattern Discovery

Chia-Hui Chang

Dept. of Computer Science and Information Engineering

National Central University, Chung-Li 320, Taiwan

chia@csie.ncu.edu.tw

Chun-Nan Hsu

Institute of Information Science

Academia Sinica, Taipei 115, Taiwan

chunnan@iis.sinica.edu.tw

Shao-Chen Lui

Dept. of Computer Science and Information Engineering

National Central University, Chung-Li 320, Taiwan

anyway@db.csie.ncu.edu.tw

Abstract

The World Wide Web is now undeniably the richest and most dense source of information, yet its structure makes it difficult to make use of that information in a systematic way. This paper proposes a pattern discovery approach to the rapid generation of information extractors that can extract structured data from semi-structured Web documents. Previous work in *wrapper induction* aims at learning extraction rules from user-labeled training examples, which, however, can be expensive in some practical applications. In this paper, we introduce IEPAD (an acronym for Information Extraction based on PAttern Discovery), a system that discovers extraction patterns from Web pages without user-labeled examples. IEPAD applies several pattern discovery techniques, including PAT-trees, multiple string alignments and pattern matching algorithms. Extractors generated by IEPAD can correctly extract structured data records with attribute values and can be generalized over unseen pages from the same Web data source. Experimental results show that IEPAD achieves 96% average retrieval rate with extractors discovered from one example page for 14 sample Web data sources. For ten of these data sources, IEPAD achieves 100% retrieval rate with extractors discovered from less than five example pages.

Keywords

Information extraction, semi-structured documents, wrapper generation, PAT trees, multiple string alignment

1 Introduction

1.1 Information Extraction and Web Intelligence

The problem of information extraction is to transform the contents of input documents into structured data. Unlike information retrieval, which concerns how to identify relevant documents from a collection, information extraction produces structured data ready for post-processing, which is crucial to many applications of text mining. Therefore, information extraction from Web pages is a crucial step enabling content mining and many other intelligent applications of the Web. Examples include meta-search engines [23], which organize search results from multiple search engines for users, and shopping agents [8], which compare prices of products from multiple Web merchants.

Information extraction has been studied for years, but mostly concentrated on free-text documents. In this case, linguistic knowledge such as lexicons and parsers can be useful. However, a huge volume of information on the Web is rendered in a *semi-structured* manner, that is, in tables, itemized lists, and the likes, where linguistic knowledge only provides limited hints. Another difference is that tables, itemized lists, etc. on different Web sites have their own unique layout and format. Virtually no general "grammar rule" can describe all possible layouts and formats so that we can not have one extractor for all Web pages. As a result, each table/itemized list may require a specific extractor, which makes it impractical to program extractors by hand. Previously, researchers proposed several *wrapper induction* approaches for rapid generation of extractors for Web pages [24, 25, 14, 18, 22, 12]. Basically, these approaches exploit machine learning

techniques to generate a specialized extractor for each Web data source. Their work produce accurate extraction results, but the generation of the extractors still requires human-labeled/annotated Web pages as *training examples*, and for each new Web site a new set of labeled training examples must be collected.

Our work here attempts to eliminate the need of user-labeled training examples. The idea is based on the fact that data on a semi-structured Web page is often rendered in some particular alignment and order and thus exhibit regular and contiguous pattern. By discovering these patterns in target Web pages, an extractor can be generated. We have a pattern discovery algorithm that can be applied to any input Web page without training examples. This greatly reduces the cost of extractor construction. A huge number of extractors can now be generated and sophisticated Web intelligence applications become practical.

Recently, efforts have been made to create *"the Semantic Web"* [1] that offers a well-organized, wide-open machine-comprehensible environment available to all kinds of intelligent agents. Our work shares the same vision but takes a different approach. The Semantic Web takes a *knowledge representation* approach to the problem and aims at building a huge standard ontology of human knowledge in XML [27]. As stated in Berners-Lee et al.'s article [1], "the Semantic Web will enable machines to comprehend *semantic documents and data*, not human speech and writings." Our work here, instead, is to enable machines to comprehend the contents of semi-structured Web pages that already prevail in the World-Wide Web, which contains a huge volume of information. A new term *"the Deep Web"* has been coined [26, 29] to refer to this huge gold mine of knowledge that we are targeting. We think mining the Deep Web and creating the Semantic Web are complementary and information extraction research can have critical impact on both efforts. After all, XML markup only provides one of many possible semantic interpretations of a document. When an application needs to extract data at a different granularity, or to integrate data in different domains, we still need a specialized information extractor to provide other interpretations. Moreover, information extraction can also help mark up legacy data.

1.2 The IEPAD Architecture

Our approach to Web page information extraction has been implemented into a system called **IEPAD**. Figure 1 shows the component diagram of IEPAD, which includes three components:

- *Pattern discoverer* accepts an input Web page and discovers potential patterns that contain the target data to be extracted.
- *Rule generator* contains a graphical user interface, called `pattern viewer`, which shows patterns discovered. Users can then select the one that extracts interesting data and then the *rule generator* will "remember" the pattern and save it as a *extraction rule* for later applications. Users can also use pattern viewer to assign attribute names of the extracted data.
- *Extractor* extracts desired information from similar Web pages based on the designated extraction rule.

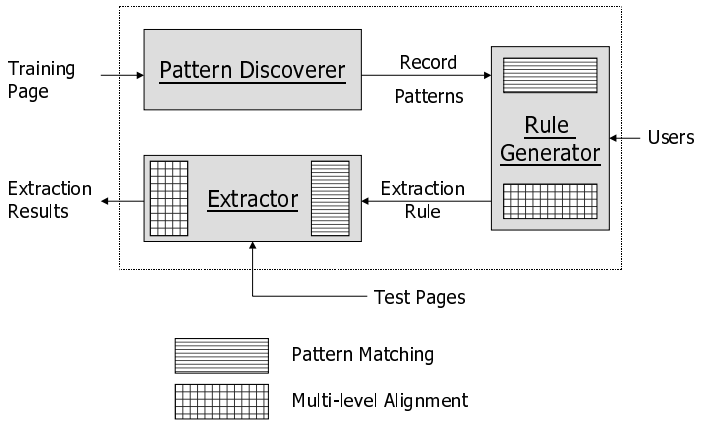


Figure 1: The IEPAD Architecture

Like many "wrapper induction" systems, IEPAD first discovers patterns to recognize record boundaries and then apply multi-level alignment to divide these records into attributes. The patterns discovered by pattern discoverer are sufficient to extract structured data for all records from any input Web page. The need of rule generator is to enhance attribute extraction in each record and improve the speed of extraction so that extraction rules can be reused for Web pages with the same pattern, usually from the same Web site, generated by a CGI script. There is no need to re-discover their pattern for each extraction.

The pattern discoverer consists of a token encoder, a PAT-tree constructor, a pattern filter, and a extraction rule composer. The output contains a set of patterns discovered in the tokenized Web page. The PAT-tree technique [20, 10] is the key that enables efficient and accurate discovery of the patterns for data records in the Web page. As for extraction of individual attribute values in each record, the analysis is implemented in the pattern viewer and the extractor to segment data records into blocks of attribute values.

The approach here differs from the previous work in wrapper induction in that our extraction rules are based on patterns while theirs are more or less based on *delimiters*. More precisely, their extraction rules use delimiters to determine which string on the Web page corresponds to a data attribute (or information slot). For example, a delimiter-based rule may state that:

> Attribute "price" starts by a prefix string `<blink>$` and ends by a suffix string with a digit followed by a HTML tag `</blink>`.

Then a string that looks like `<blink>$129.99</blink>` will match this rule and `129.99` will be extracted as "price." In contrast, for the same example task, our extraction rule simply states that:

> The strings containing attribute "price" have this pattern: `<blink>$<text></blink>`,

where `<text>` matches any text string. The reason we use pattern-based rule instead of delimiter-based rule is that the data to be extracted are often generated based on some predefined HTML templates (e.g.

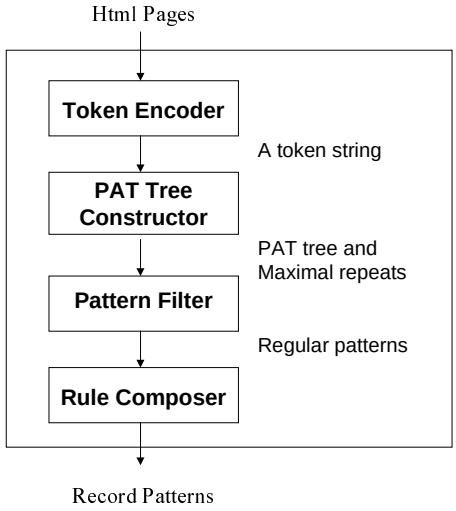


Figure 2: Flow chart of the pattern generator

job postings, flight schedules, query results from search engines, etc.). This naturally inspires the idea to discover such templates (or patterns) since the occurrences of these patterns are usually aligned `regularly` and `contiguously` to allow for easy comprehension. These characteristics also allow for pattern discovery that automates the generation of extraction patterns of these templates. In other words, the task of extraction rule generation can be solved by pattern discovery without user-labeled training examples that are required for previous wrapper induction systems.

1.3 Organization

The remainder of this paper is organized as follows. In section 2, we present the pattern discoverer. In section 3. we describe the pattern viewer for rule generation and the implementation of the extractor module. Section 4 reports experimental results. Section 5 compares our work with related work. Finally, we draw the conclusions in section 6.

2 Pattern Discoverer

Figure 2 gives a flowchart of the pattern discovery process. Given an HTML page to IEPAD, the token encoder will tokenize the page into a string of abstract representations, referred to here as a *token string*. Each token is represented by a binary code of fixed length. The PAT tree constructor [10, 20] takes the binary string to construct a PAT tree. The pattern discoverer then uses the PAT tree to discover patterns, called *maximal repeats*. These maximal repeats will be fed to a filter, which filters out undesired patterns and produces candidate patterns. Finally, the rule composer revises each candidate pattern to form an extraction rule in regular expression. The following subsections describe how these parts work.

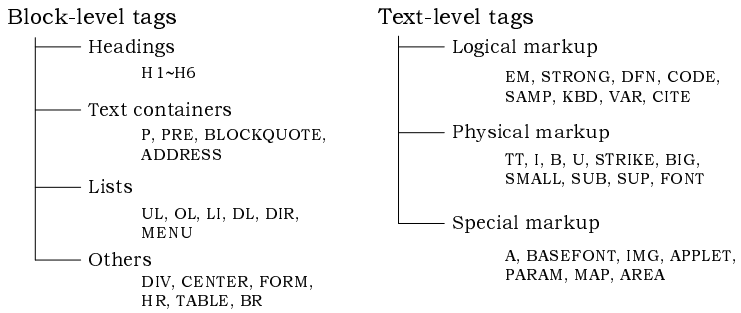


Figure 3: Tag classification

2.1 Web Page Encoding

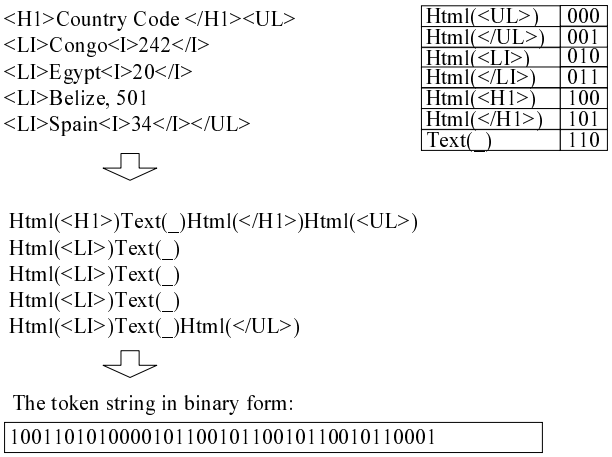
Since HTML tags are the basic components for document presentation and the tags themselves carry a certain structural information, it is intuitive to examine the tag token string formed by HTML tags and disregard other text content between two tags to see the display template. Hence, the simplest abstraction is as follows:

1. Each tag is encoded as a tag token Html(<tag_name>)
2. Any text between two tags are regarded as a special token called <TEXT>

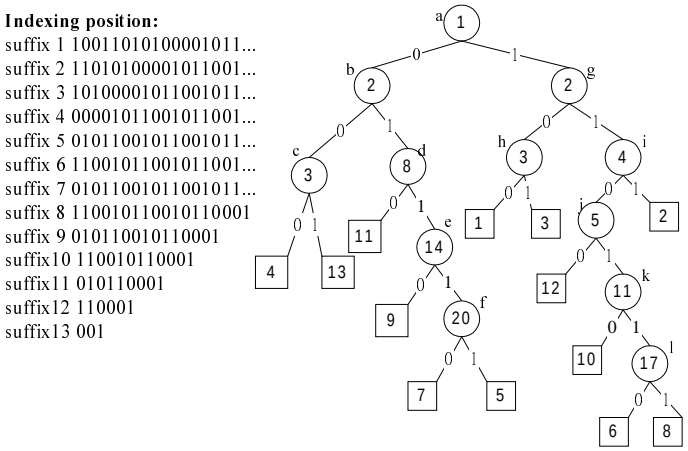
There are various ways to encode a Web document. With different abstraction mechanisms, different patterns can be produced. For example, HTML tags, according to their functions, can be divided into two distinct groups: `block-level tags` and `text-level tags`. The former defines the structure of a document, and the latter defines the characteristics (format and style, etc.) of the text contents (See Figure 3 for a classification of block-level tags and text-level tags [28]). Block level tags include headings, text containers, lists, and other classifications, such as tables and forms. Text-level tags are further divided into three categories including logical tags, physical tags, and special tags for marking up texts in a text block. The many different tag classifications allow different HTML translations to be generated. The user can choose an encoding scheme depending on the level of desired information to be extracted. For example, skipping all text-level tags will result in higher abstraction (called *block-level encoding*) from the input Web page than all tags are included. As shown in Figure 4(a), the Congo code (an example used in [19]) can be translated into a string of 13 tokens using block-level encoding.

2.2 Constructing PAT Trees for Maximal Repeats

By repeats, we generally mean any substring that occurs at least twice in a string. To better capture the idea of repeats and also reduce the number of patterns discovered, the concept of `maximal repeats` is used to refer to the longest patterns. The idea is to extend a repeat in both directions to its longest. We call a repeat `left maximal` (right maximal) if the repeat can not be extended on the left (right) direction (See



(a)



(b)

Figure 4: The Congo code and its PAT tree

[3]) for all occurrences. We say a repeat is maximal if it is both left maximal and right maximal. A formal definition is given as follows:

Definition Given an input string S, we define maximal repeat α 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`).

To automatically discover patterns, a data structure called `PAT trees` is used to index all suffixes in the encoded token strings. A PAT tree is a Patricia tree (Practical Algorithm to Retrieve Information Coded in Alphanumeric [20]) 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 shows the different bit between suffix strings in the same subtree. Like a suffix tree [11], the Patricia tree stores all its suffix strings at the external nodes. For a token string with n indexing point (or n suffixes), there will be n external nodes in the PAT tree and $n-1$ internal nodes. This makes the tree $O(n)$ in size. 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 tandem repeats in genomes [17]. It has also been used in Chinese keyword extraction [7] for its simpler implementation than suffix trees and its great power for pattern discovery.

PAT trees organize input in such a way that all suffixes with the same prefix are stored in the same subtree. Therefore, it has some nice characteristics for pattern discovery:

- First, all suffixes in a subtree share a common prefix, which is the `path label` that leads from the tree root to the subtree root.
- Second, the number of leaves in the subtree is exactly the number of occurrences of the path label.
- Third, each path label represents a right maximal repeat in the input.

To build the encoded token string into a PAT tree, each tag token is denoted by a fixed length binary representation. Suppose three bits encode the tokens in the Congo code as shown in Figure 4(a). The encoded binary string for the token string of the Congo code will be a binary string "100110 101000 010110 010110 010110 010110 001" of 3*13 bits. Referring to Figure 4(b), a PAT tree is constructed from the encoded binary string of the sample example. The tree is constructed from thirteen bit sequences. Each leaf, or external node, is represented by a square labeled by a number that indicates the starting position of the string. For example, leaf 2 corresponds to suffix 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 indicating the first different bit for suffix strings in the subtree rooted at the internal node. For example, the first different bit between suffix 5 and 9 is 14 as indicating by the subtree root node e.

As shown in the PAT tree, all suffix strings with the same prefix will be located in the same subtree. Hence, it provides surprisingly efficient, linear-time solutions to the problems of complex string search, including string prefix searching, proximity searching, range searching, longest repetition searching, most frequent searching, etc [20, 11]. 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 right maximal repeat in the input string. However, not every path-label or repeated string represents a maximal repeat. For example, the path label for node j is not left maximal since suffix 6, 8, 10 and 12 all have the same left character `Html(<LI>)`. 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. Let's call such a node v `left diverse`. Followed by the definition, the property of being left diverse propagates upward in the PAT tree. Therefore, all maximal repeats can be found in linear time to the tree size .

Consequently, given the minimum repeat count k and pattern length $|\alpha|$, we can simply traverse the PAT tree in postorder to enumerate all path labels to discover all right maximal repeats. At each internal node, we verify left maximality by checking the left tokens of all leaves (suffixes). If all left tokens are the same, then this repeat is not left maximal and can be extended.

2.3 Sifting For Regular and Contiguous Patterns

As described above, most information we want is generated based on some predefined templates and is commonly aligned `regularly` and `contiguously`. To discover these display patterns, two measures, called "variance" and "density", are defined to evaluate whether a maximal repeat is a promising extraction pattern. Let the occurrences of a maximal repeat α are ordered by its position such that $p_1 < p_2 < p_3 \ldots < p_k$, where p_i denotes the position in the encoded token string.

Variance of a pattern is computed by the coefficient of variance of the interval between two adjacent occurrences $(p_{i+1} - p_i)$. That is, the ratio of the standard deviation of the interval and the mean length of the interval.

$$variance(\alpha) = \frac{\sigma(\{d_i | 1 \leq i < k, d_i = p_{i+1} - p_i\})}{(p_k - p_1)/(k-1)} \tag{1}$$

Density is defined as the percentage of repeats in the interval between the first and the last occurrences of the repeat. That is,

$$density(\alpha) = \frac{(k-1) * |\alpha|}{p_k - p_1} \tag{2}$$

where $|\alpha|$ is the number of tokens in α.

Generally speaking, machine-generated Web pages often embed relevant information in templates which has small variance and large density. To sift for potentially good patterns, a simple approach is to use

a threshold for each of these measures. Only patterns with variance less than the variance threshold and density greater than the density threshold are considered candidate patterns.

The above approach is easy for implementation. However, it can fail to extract some layout templates if the variance threshold is not set properly. The reason is that regular patterns can sometimes have large variance coefficient. For example, advertisement banners inserted among the search results can divide the occurrences into several parts like "Lycos" does. These exceptions result in maximal repeats with large variance.

Occurrence Clustering

To handle patterns with variance greater than the specified threshold, the occurrences of a pattern are carefully clustered to see if any partition of the pattern's occurrences can form an independent and regular block. The idea here is to cluster the occurrences into partitions so that we can examine each partition to see if it is regular enough. A simple loop can accomplish this one-dimension clustering. Let $C_{i,j}$ denotes the list of occurrences $p_i, p_{i+1}, ..., p_j$ in increasing order. Initialize i and j to 1. For each p_{j+1}, if the variance coefficient of $C_{i,j+1}$ is less than a constant then p_{j+1} is included as part of the current partition; otherwise, $C_{i,j}$ is exported as a partition and a new partition is created by assigning $j+1$ to i.

Once the occurrences are partitioned, we can then compute the variance for each individual partition. If a partition includes more than the minimum occurrence count and has variance less than the threshold v_ϵ, the pattern as well as the occurrences in this partition are exported. Note that the threshold v_ϵ is set to a small value close to zero to control the number of generated partitions.

2.4 Composing Extraction Patterns

In addition to the large variance, patterns with density less than one cause another problem we need to deal with. Since PAT trees compute only "exact match" patterns, templates with exceptions can not be discovered through PAT trees. To allow inexact or approximate matching, the technique `multiple string alignment` is used.

Suppose a candidate pattern has k occurrences, $p_1, p_2, \ldots, p_k$ 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 $k-1$ strings $\mathcal{S} = \{P_1, P_2, \ldots, P_{k-1}\}$ so that the generalized pattern can be used to extract all records we need. For example, suppose "adc" is the discovered pattern for token string "*adcbdadcxbadcxbdadc*". Suppose we have the following multiple alignment for strings "*adcbd*", "*adcxb*" and "*adcxbd*":

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

The extraction pattern can be `generalized` as "$adc[x|-]b[d|-]$" to cover these three instances. This regular expression of record patterns is able to handle exceptions such as missing attributes, multiple attribute values, and variant attribute permutations that might occur in Web pages [14]. The last two exceptions can

be considered as the case of missing attributes. For example, the context-based rule "$[U|-]N[A|-][M|-]$" can generalize over different permutations of four attributes: $(U, N, A, M), (U, N, A), (U, N, M), (N, A)$.

Multiple string alignment is a generalization of the `alignment` for two strings that can be solved in $O(n * m)$ by `dynamic programming` to obtain optimal `edit distance`, where n and m are string lengths. Extending dynamic programming to multiple string alignment yields a $O(n^k)$ algorithm. Alternatively, an approximation algorithm is available such that the score of the multiple alignment is no greater than twice the score of the optimal multiple alignment [11]. The approximation algorithm starts by computing the `center string` S_c in k strings that minimizes `consensus error`. 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 pattern with density less than 1, the `center star` approximation algorithm [11] for multiple string alignment is applied to generalize the extraction pattern. Note that the success of this technique lies in the assumption that extraction patterns often occur contiguously together. If the multiple alignment results in extraction patterns with too many alternatives, such a pattern is unlikely to be interesting (For f alternatives, there might be 2^f permutations). Therefore, we set an upper bound that there can be at most m mismatches. m is the maximum error, usually set to ten in our experiments.

Pattern Rotation

Suppose a generalized 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. Since c_1 might not be the start of a record, we use a (right) rotating procedure to compare the pattern with the `left string` of the first occurrence p_1 from right to left. The purpose is to find the correct starting position and generate a new pattern "$c_jc_{j+1}...c_nc_1...c_{j-1}$". If the left character of the first occurrence is the same as c_n, we rotate the pattern to "$c_nc_1c_2...c_{n-1}$". The process will be continued until the left character of the first occurrence is not the same as the last token; and when the rotation stops, the final pattern is the output. Similarly, we use a left rotating procedure to compare the pattern with the `right string` of the last occurrence p_k from left to right. If the right character of the last occurrence is the same as the first token, we rotate the pattern into "$c_2c_3...c_nc_1$". The process will be continued until the right character of the last occurrence is not the same as the first token; and when the rotation stops, the final pattern is the output. With this pattern rotation, the correct record boundary can be identified.

In summary, we can efficiently discover all maximal repeats (with pattern length and occurrence count greater than default thresholds) in the encoded token string through the constructed PAT tree $\mathcal{T}$. Second, with variance coefficient and density, we can sift the maximal repeats for promising patterns. For patterns with large variance, occurrence partition can cluster records into blocks of interest. As for low density pattern, multiple string alignment is applied to produce more complete extraction pattern based on the assumption of contiguous occurrences.

3 Rule Generation and Data Extractor

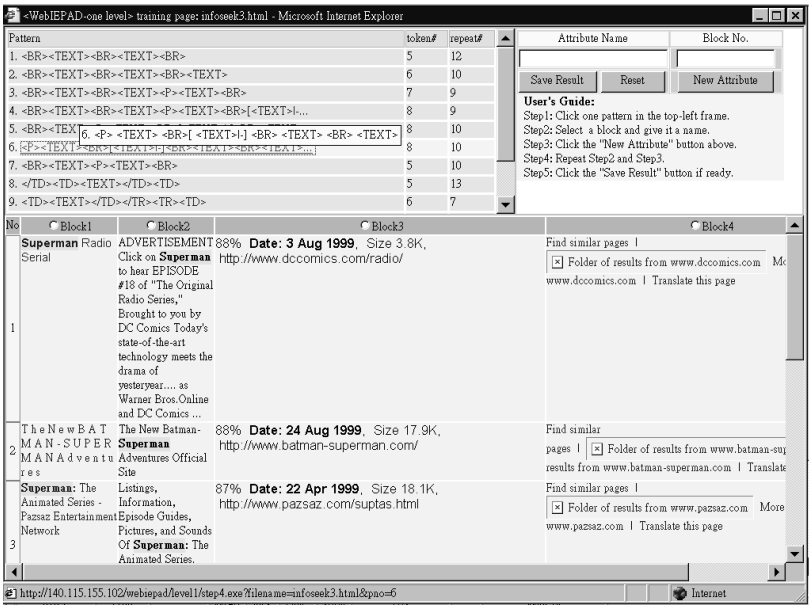
The maximal repeats discovered automatically by pattern discoverer corresponds to data records appearing in a semi-structured Web page. This discovery can be completed without any prior knowledge from the user. Since there might be more than one useful pattern, an interface is necessary for users to choose a proper record extraction pattern. The purpose of the pattern viewer is designed to provides a graphic user interface so that the user can view the extracted content and select the desired information. Note that this selection is different from labeling training examples, since the users are not asked to label examples for pattern discovery but to select one of the discovered patterns.

Another function of the pattern viewer is to allow users to assign attribute names and information slots in a record. Since record patterns only tells where the records are located, there must be some way to designate attributes in a record. Through pattern viewer, users not only specify the record pattern but also the attributes in a record. The saved extraction rule will in turn be used to extract information from other similar Web pages. The user can also adjust the parameters (including the encoding scheme, the minimum pattern length, the minimum occurrence count, the variance and the density thresholds) to generate good record patterns through pattern viewer. Hence, the pattern viewer has at least four functions: adjusting parameters, selecting patterns, specifying information slots, and generating extraction rules. Once the extraction rule is specified, users can then test the rule on unseen Web pages by data extractor.

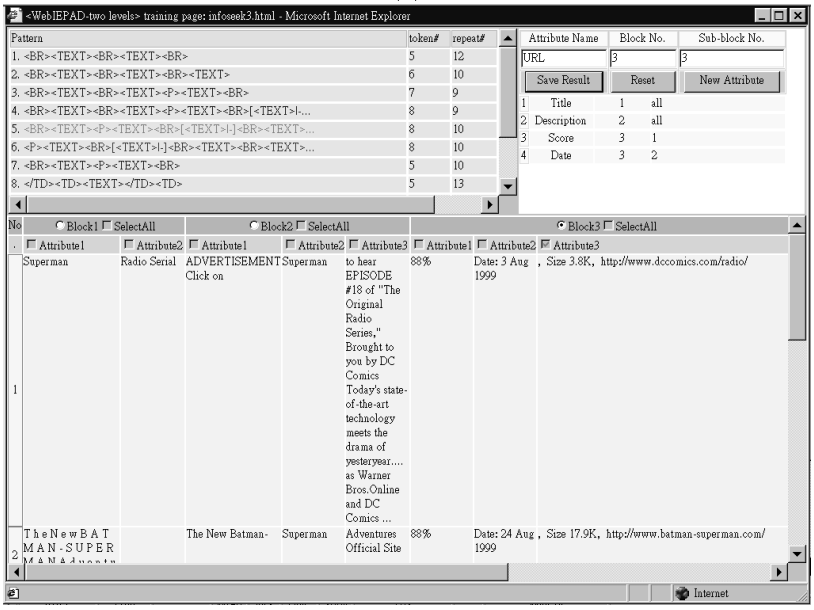
Figure 5 shows the snapshots of the pattern viewer. The upper-left window shows the discovered record patterns and the lower window shows the corresponding extracted data. We can see in Figure 5(a) the user chooses the sixth record pattern and the extracted data are displayed in the lower window. Note that all the records extracted are further divided into blocks (or slots) and aligned for selection through the upper-right window, where the users can type in the attribute names and select the desired information blocks (see the upper-right window of Figure 5(b)) by clicking the check boxes above each block. We will explain how extracted data are aligned in the following section.

3.1 Information Division

Given the user specified record pattern, the pattern viewer first matches it in the encoded token string to find all occurrences of the pattern, then aligns each occurrence to the pattern. This gives a straightforward segmentation of the records where each token represents an attribute. Recall that we have recorded the starting position of each token in the Web page during the encoding phase. With this information, we can always trace the corresponding data segment in the Web page for any tokens. Since patterns are composed of tag tokens and text tokens and only `text tokens` and some special tag tokens (which contain hyperlinks such as <A> and <IMG> tags) might contain data to be extracted, we can show only the contents that are encoded as text tokens and the hyperlinks that are embedded in <A> or <IMG> tags accordingly. The procedure to divide the set of all records that a pattern α can match is outlined in Figure 6. In summary, if there are m text tokens and special tag tokens in a record extraction pattern, all matched token strings



(a)



(b)

Figure 5: The record patterns and the user interface (a) one-level alignment (b) two-level alignment

```
Block_division(α, R) {
    for each r_i in R do
        r'_i= Align(α, r_i);
        m= 0;
        for j=1 to |α| do
            if α[j]=<TEXT> or <A> or <IMG> then
                m= m + 1;
                if r_i[j]≠'-' then block[i][m]= r'_i[j];
                else block[i][m]= null;
                endif
            endif
        endfor
    endfor
    return block;
}
```

Figure 6: Block division procedure

can be divided into m blocks. For example, as shown in Figure 5(a), there are four text tokens in the fifth candidate pattern, "<P><TEXT>
[<TEXT>]
<TEXT>
<TEXT>". Each matched token string of this pattern can be divided into four blocks.

Although the alignment can divide the record information into several blocks, this may not be sufficient. When higher level abstraction, say block-level encoding, is used for pattern discovery, the content in a block may contain not only text but also text-level tags. If we would like to extract "finer" information, the content in each block has to be further processed. Sometimes, lower-level encoding scheme can be employed to save further process. However, it becomes much difficult to discover and compose the record pattern since the success of the pattern discovery approach lies in good abstraction of the Web pages.

To extract finer contents, a compromised method is to employ `multiple string alignment` and apply block division again. Let the encoding for record boundary be the first-level encoding scheme. We will apply a second-level encoding scheme to the text contents in each block and align these encoded token strings for further block division as outlined in Figure 7. In the Multi-level alignment procedure, the contents in each column of the block matrix are translated through a lower-level encoding scheme. The center-star multiple string alignment is then applied to compose a consensus pattern. Finally, block division procedure can be used to divide these contents according to the consensus pattern.

For example, the text contents that belong to the third block in Figure 5(a) can be further aligned to a generalized pattern "[<FONT>]<TEXT><B><TEXT></B><TEXT> [</FONT>]". With three text tokens inside, the contents will be further segmented into three sub-blocks, where the first text token corresponds to "score", the second text token corresponds to "date" and the third token corresponds to page size and URL field. The same step can be applied until the desired information can be successfully separated from others. As we shall see in next section, two-level's encoding can extract the target information quite well for most Web data sources in our experiments.

```
MultiLevel_alignment(block, π){
    for j=1 to m do
        for i=1 to k do
            r_i = Encoding(block[i][j], π);
        endfor
        R= {r_1, r_2, ..., r_k};
        β_j= CenterStarMultipleAlignment(R);
        A_j= Block_division(β_j, R);
    endfor
    A= A_1 + A_2 + ... + A_m;
    return A;
}
```

Figure 7: Multi-level alignment

3.2 The Extractor

It is easier to extract data from searchable data sources than from hand-crafted static pages since these Web pages are produced by programs with some predefined templates. For these data sources, we can select one page as the input to our system and choose the proper pattern as the extraction rule. Once the pattern viewer have successfully divided all records into small information slots, the desired information slots can be specified in an extraction rule that can then be used to extract other Web pages fetched from the same Web site.

The extraction procedure is as outlined in Figure 8. The procedure is pretty similar to that of the pattern viewer. According to the input extraction rule, the extractor first translates the Web pages into the token string based on the first-level encoding scheme and then match all occurrences of the record extraction pattern in the encoded token string. The record extraction is achieved through a pattern matching algorithm. Typical pattern matching algorithms, like the Knuth-Morris-Pratt's algorithm or Boyer-Moore's algorithm [16, 2], can do the work. Note that each extraction rule composed by multiple string alignment actually represents several patterns since the patterns are expressed in regular expression with alternatives. In other words, there are alternatives routes. Therefore, a token string can match several patterns. In such cases, the longest match is considered.

After applying the pattern matching procedure, the block division procedure is applied to the extracted data records and returns the resulting block matrix. This is the first level division. The deeper level division of the block matrix can be achieved through the call of multi-level alignment procedure in the loop. The above procedure is exactly the same as that for pattern viewer, except for the final step where each attribute value can be extracted according to the information slots indicated in the extraction rule. Figure 9 shows that a test page (infoseek4.html) is uploaded and the extraction results are shown in the lower window.

```
Extractor(page, rule){
    S= Encoding(page, rule.scheme_1);   //Translate page into token string;
    R= Boyer_Moore(S, rule.α);   // Find all occurrence of pattern α;
    block_1= Block_division(rule.α, R);
    for j=2 to l do
        A= MultiLevel_alignment(block_{j-1}, rule.scheme_j);
        block_j= A;
    endfor
    Extract designated information slots for each attribute;
}
```

Figure 8: Extractor procedure

Figure 9: Two level attribute value extraction (test page: infoseek4.html)

Table 1: Data description

Data source	# of records	# of attributes	missing	unordered	layout template variance	layout template density
AltaVista	10	4	Yes	No	0.05	0.62
DirectHit	10	4	Yes	No	0.04	0.20
Excite	10	4	Yes	No	0.09	0.89
HotBot	10.2	4	Yes	No	0.24	0.62
Infoseek	15	3	Yes	No	0.12	0.46
MSN	10	3	No	No	0.36	0.27
NorthernLight	10	3	Yes	Yes	0.10	0.86
Sprinks	20	4	Yes	No	0.18	0.30
Webcrawler	25	3	No	No	0.14	0.97
Yahoo	20	4	Yes	No	0.08	0.56
Okra	18.5	4	Yes	No	0.05	1.00
Bigbook	14.2	6	No	No	0.00	1.00
IAF	5.9	6	Yes	Yes	0.00	0.80
QuoteServer	3.7	18	No	No	0.00	1.00

4 Experimental Results

The experiments here use two test data sets. The first one contains test pages from ten popular search engines. We collect 100 test pages for each data source. The test data can be downloaded from `http://www.csie.ncu.edu.tw/~chia/webiepad`. The second data set are Okra, IAF, BigBook, and QuoteServer, taken from Kushmerick's work [18]. These four sources have also been used in Hsu, et al. [13] and Muslea's paper [22] for the purposes of comparison. Table 1 shows the basic description of each data source. The first four columns show the number of records in each page, the number of attributes in each record, the existence of missing attribute in a record, and unordered exceptions such as multiple values for one attribute or variant attribute permutations, respectively. The next two columns show the variance coefficient and density of the correct pattern which will be described below.

The average document size is 28K bytes and 11K bytes for the above two data sets, respectively. The search results of the first data set typically contain at least 10 data records and more advertisements, while test pages in the second data set contain less data records and advertisements. In addition to block-level encoding scheme, we also conduct experiments on All-tag encoding scheme and three other encoding schemes which skip logical, physical, and special tags respectively. For example, the No-Physical encoding scheme skips physical markups, including <TT>, <I>, <B>, and <U>, etc. Table 2 shows the comparison on the length of encoded token string. The results of the No-Logical encoding scheme are not shown because logical markups are less used in HTML files (only 0.4%) and the difference is not obvious from that of all-tag encoding scheme. Basically, the higher the abstraction level, the shorter the length. Whichever the data set, the size of the encoded token string is much smaller than the document size. The number of tokens after translation is about 4 to 5% the page size for the lowest-level encoding scheme when all tags are considered

Table 2: Data size with different encoding schemes

Data Set	Search Engines (Doc size=28K)		Okra, etc. (Doc size=11K)	
Encoding	# of tokens	Percentage	# of tokens	Percentage
All-tag	1023	4.0%	584	5.0%
No-Physical	839	3.0%	473	4.0%
No-Special	835	3.0%	447	3.9%
Block-level	639	2.0%	333	3.0%

(see Table 2). The number of tokens is even small when block-level encoding is used. Therefore, the effort to build PAT trees and the tree size can be kept small.

4.1 Parameters Setup

The input parameters to pattern discoverer include the encoding scheme, the minimum pattern length, the occurrence count, and the thresholds for variance and density. We choose block-level encoding scheme to discover record pattern since it is the highest level abstraction and perform the best in our previous work [3]. The default value for the minimum length of maximal repeats and the minimum occurrence count are set to 3 and 5, respectively. In order to set the parameters properly, we have computed the variance of the layout template for each data source. That is, the positions of all records in the block-level encoded token string are used to compute variance using Equation (1). Similarly, we can compute the density of the common prefix of the layout template by Equation (2). These two values are shown in the last two columns of Table 1. The variance coefficients are small for most data sources as expected and the maximal value is 0.36. As for density values, they vary from 0.20 to 1, which indicates that the missing tags can occur far front in the layout template.

The number of discovered record patterns depends on the variance and density threshold. If the variance threshold is set as the maximal variance (0.36) and the density threshold is set as the minimal density (0.20), the number of maximal repeats validated is about 5 as shown in the "fixed" column of Table 3. If on the other hand, the variance and density thresholds for each data source are set by the variance and density of its layout template, the number of maximal repeats can be reduced to 2 (the "adaptive" column of Table 3). This filtering process based on variance and density thresholds is very useful because without this filtering process, the number of maximal repeats discovered in a block-level encoded token string can be close to a hundred. With the filtering using the variance and density thresholds, the number of maximal repeats remained can be kept small for users to select. These maximal repeats, after multiple string alignment and pattern rotation, make up the record patterns for final output. Note that the number of record patterns can increase due to pattern rotation procedure of the aligned pattern.

The extraction rule we used here extracts not only record boundary, but also attribute values through multi-level division. By comparing the fifth column in Table 3 with the second column of Table 1 (i.e.,

Table 3: Performance

	# of maximal repeats		performance		# of attributes	
Data source	fixed	adaptive	retrieval	accuracy	1 level	2 levels
AltaVista	6	2	100%	100%	4	4
Direct Hit	1	1	100%	100%	3	4
Excite	3	1	100%	100%	4	4
HotBot	5	1	100%	100%	4	4
Infoseek	6	1	100%	100%	3	3
MSN	4	1	100%	100%	3	3
NorthernLight	8	3	100%	100%	3	4
Sprinks	3	1	100%	100%	3	4
Webcrawler	5	2	100%	100%	3	3
Yahoo	9	3	100%	100%	4	4
Okra	3	2	100%	100%	4	4
Bigbook	3	1	100%	100%	5	6
IAF	2	2	100%	100%	1	3
QuoteServer	5	2	100%	100%	3	18

the correct number of attributes), we see that one-level division along can extract the desired information slots for eight data sources; while two-level extraction, where the All-tag encoding scheme is used in the second-level, can extract all attributes for all Web sites except for IAF. This is because IAF uses a special <pre> tag to separate the detailed information. The second-level extraction here using All-tag encoding can only divide the record into three slots, which needs text-level encoding for further extraction. Since most attribute values are tag-separable, attribute extraction can become much easier.

To evaluate the performance of a pattern, two measures: retrieval rate and accuracy rate are evaluated. Retrieval rate is defined as the ratio of the number of desired data records correctly extracted and 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 correctly extracted and the number of records extracted by the rule. The performance of the generated pattern is shown in the performance column of Figure 3. In either "fixed" or "adaptive" setting, the best discovered pattern can achieve 100% retrieval and accuracy. This implies that pattern discoverer can always generate a record pattern to extract all records in these pages.

4.2 Generalizing over Unseen Pages

Although the process for rule generation is not the same as a typical machine learning process, the goal is the same. That is, to generalize the extraction over unseen Web pages. Therefore, we design the following experiments in a way similar to those usually conducted in machine learning. For each Web site, we randomly select 30 pages as the training pages and use the remaining as the validation set. The rule generated using the training set will be used to extract information from the validation set.

In the training phase, one page from the training set is fed to the system for the discovery of extraction pattern. The user then choose one (that extracts only correct records) as the record extraction rule and

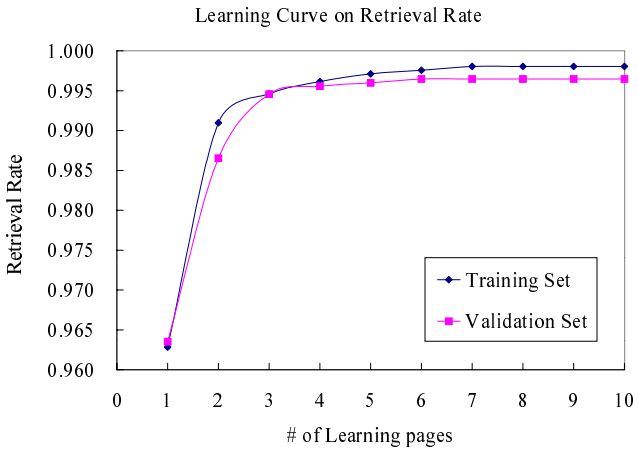


Figure 10: Learning curve on training set and validation set

specify the information block for attribute value extraction. The extraction rule will then be applied to other training pages to check the performance. If the rule can not extract all the records for a page, the page will then be fed to the system for the discovery of proper extraction rules. Existing extraction rule will then be appended with the extraction rule for the second page and form a set of new extraction rules and applied to extract other pages in the training set. The process goes on to extract as many records as possible. Finally, in the validation phase, the extraction rule is applied to validation set for performance evaluation.

For most Web sites (twelve of the fourteen), IEPAD can generate a best rule that can achieve both 100% retrieval and accuracy. This might not be achieved for only one training page. However, thirty training pages are sufficient to present the diversity for the system to generate alternative patterns. Note that not all thirty training pages are used to generate patterns. Only pages that can not be 100% extracted are used to generate patterns. Choosing a conservative pattern with 100% accuracy rate, Figure 10 shows the retrieval rate of the extraction patterns joined by different numbers of training pages. The number of training pages that are used to discover record patterns is three in average and ten at most. The retrieval rate reaches 96.28% on average using the first rule (discovered from one training page). When the second rule (discovered from the first mis-extracted page) is used, the retrieval rate reaches 99.10% in the training set. With five training pages, IEPAD achieves 100% retrieval rate for twelve data sets except for HotBot and NorthernLight.

The learning curves are different from that produced by machine learning-based approaches since the learning is based on all records in a page instead of one training example. We do encounter errors in the data, such as breaking tags, etc. But the percentage is low. The retrieval rate is not only high in the training set but also in the validation set, which shows that thirty training pages have exhibited enough variety in the display format for the system to generalize. As we can see in Figure 10, the retrieval rate

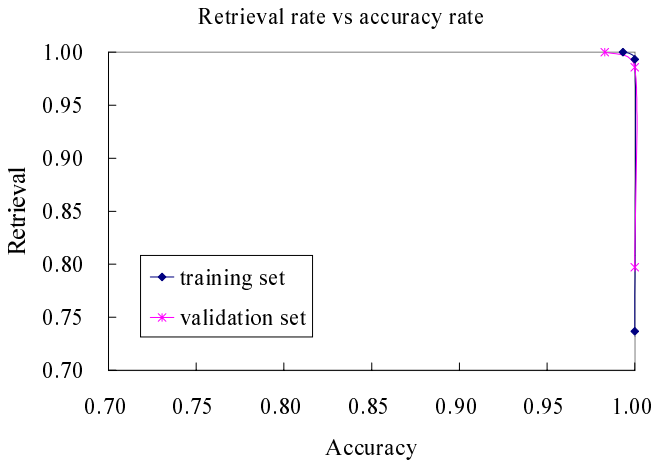


Figure 11: Retrieval rate vs accuracy rate for Northernlight

for the validation set reaches 96.35% for 1 training page and 98.65% for 2 training pages. Of the 14 Web sites, IEPAD achieves 100% retrieval rate for ten of them with less than five training pages in the validation phase. We can also plot learning curves with higher retrieval rate if different pattern selection strategy is used. For example, the retrieval rate can be tuned to nearly 100% with the accuracy rate measured around 98.9%. This can be achieved if we choose an aligned pattern that comprehend more variety in the records. Of course, such patterns may sometimes extract extra information because the patterns generalized from multiple string alignment may comprehends more patterns. There is a tradeoff. If retrieval rate is more important, accuracy rate will have to be sacrificed.

We conduct another experiment to illustrate this point. When applying the rule over unseen pages, some of these patterns can achieve higher retrieval rate with less accuracy rate, while some can achieve higher accuracy rate with less retrieval rate. Depending on the control of variance and density threshold, the IEPAD may generate several record patterns for users to choose from. While some patterns can achieve higher accuracy rate with less retrieval rate, others (more comprehensive patterns) can achieve higher retrieval rate with less accuracy. For example, two typical patterns can be found in a training page for Northernlight: one shorter pattern with no alternatives and one longer pattern with more alternatives. Sometimes, a compromised pattern can be found. The shorter pattern can extract 74.7% of the records with 100% accuracy, while the longer pattern can extract 99% records with 99% accuracy. Therefore, there is a tradeoff between retrieval rate and accuracy rate as shown in Figure 11.

5 Related Work

Building Information agents for integrating data sources on the World-Wide Web is an active research topic in recent years [4, 5, 8, 15]. An critical problem that must be addressed for these agents is how to extract structured data from the Web. Some of the projects rely on hand-coded extractors, others provide script languages to express extraction rules (written by a human expert) [4, 5]. Since it is inappropriate to write an extractor for each Web data source, machine learning approaches are proposed to solve the problem.

Kushmerick, et al. coin the term "wrapper induction" and describes a machine learning approach called "WIEN" [19]. Softmealy is a wrapper induction algorithm that generates extraction rules expressed as finite-state transducers [14]. Softmealy's extraction patterns are far more expressive than the WIEN's. The main limitation of both approaches is their inability to use delimiters that do not immediately precede and follow the data to be extracted. Stalker is a wrapper induction system that performs hierarchical information extraction [21]. It introduces the Embedded Catalog Tree (ECT) to describe the output schema for the extraction task. With the ECT, STALKER is said to extract data from documents that contain arbitrarily complex combinations of embedded lists and records.

Note that "wrapper induction" systems actually induce extraction rules and generate rules that depends on syntactic similarities instead of linguistic constrains. Comparing their work with IEPAD, they are different in many aspects. First, their work require user-labeled examples to learn the extraction rules, while IEPAD only requires users to select record pattern and information slots. Second, in terms of expressive power, IEPAD can achieve the same performance for the test data used by Stalker [21] and multi-pass Softmealy [13]. Finally, these systems generate delimiter-based extraction rules while IEPAD generates pattern-based extraction rules.

More recently, Chidlovskii, et al. presented an approach to wrapper generation, which uses grammar induction based on string alignment [6]. The authors claim that their system requires a small amount of labeling by the user: labeling only one record suffices. Other records are found by iteratively align adjacent records. However, this approach only achieve 73% accuracy since it considers only two adjacent records at a time while IEPAD takes all records into consideration a time.

Fully automatic approach to IE is rare and often depends on some heuristics. For example, Embley et. al. [9] describe a heuristic approach that discovers record boundaries in Web documents by identifying *candidate separator tags* using five independent heuristics and choosing a consensus separator tag based on a heuristic combination [9]. However, one serious problem in this one-tag separator approach is that their system cannot be applied to Web pages where two or more separator tags are the same within a record, because their system cannot distinguish them and will fail to extract data correctly. As a result, the applicability of their system is seriously limited. Moreover, their system only discovers record boundaries while IEPAD can correctly extract attribute values from a record.

6 Conclusion

With the growth of the amount of online information, the availability of robust, flexible information extraction systems has become a stringent necessity. In this paper, we presented an unsupervised approach to semi-structured information extraction. We propose a pattern discovery approach that generates extraction rules for any input Web pages. The process of record pattern discovery requires no human intervention or user-labeled training examples.

The key features of this approach are as follows. First, by applying the PAT-tree pattern discovery algorithm, the discovery of record boundaries can be completed automatically without user-labeled training examples. Second, by applying the multiple alignment algorithm, discovered patterns can be generalized over unseen pages. Third, the extraction rule is pattern-based instead of delimiter-based. Finally, by allowing alternative expressions in the extraction rules, IEPAD can handle exceptions such as missing attributes, multiple attribute values and variant attribute permutations.

Comparing IEPAD to previous work, IEPAD performs equally well in terms of extraction accuracy but requires much less human intervention to produce an extractor. In terms of the tolerance of layout irregularity, the extraction rules generated by IEPAD allow alternative tokens and hence can tolerate exceptions and variance such as missing attributes in the input. We also show that multi-level alignment can be applied to extract finer information by extracting attributes from a discovered data record. We have successfully applied our approach to extract data from real Web sites with diverse structuring patterns.

Directions of future work include implementing text-level encoding to extract more delicate information. An enhanced presentation of the generated patterns can be helpful to generalize over unseen pages. We also plan to incooperate semantic knowledge such as lexicons of the application domains to filter discovered patterns. Finally, extending our work to help creating the Semantic Web will be another interesting topic.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila, The Semantic Web, *Scientific American* (May. 2001).

[2] R.S. Boyer, and J.S. Moore, A Fast String Searching Algorithm, Communication of ACM, Vol 20, pp.762–772 (1977).

[3] C.-H. Chang, S.-C. Lui, and Y.-C. Wu, Applying Pattern Mining to Web Information Extraction, *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 4-15 (Hong-Kong, 2001).

[4] S. Chawathe, H. Garcia-Molina, J. Hammer, et al. The TSIMMIS Project: Integration of Heterogeneous Information Sources. *Proceedings of the 10th Meeting of the Information Society of Japan*, pp. 7–18 (Tokyo, 1994)

[5] B. Chidlovskii, U. Borghoff, and P. Chevalier, Towards Sophisticated Wrapping of Web-based Information Repositories. *Proceedings of the 5th International RIAO Conference*, pp.123–135 (1997).

[6] B. Chidlovskii, J. Ragetli, and M. Rijke, Automatic Wrapper Generation for Web Search Engines. *Proceedings of the 1st International Conference on Web-Age Information Management*, WAIM'2000, LNCS Series (Shanghai, China, 2000).

[7] L.F. Chien, PAT-tree-based Keyword Extraction for Chinese Information Retrieval. *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval.* pp.50–58 (Philadelphia, PA, 1997).

[8] R.B. Doorenbos, O. Etzioni, and D.S. Weld, A Scalable Comparison-shopping Agent for the World-Wide Web. *Proceedings of the 1st International Conference on Autonomous Agents.* pp. 39–48 (NewYork, USA, 1997).

[9] D. Embley, Y. Jiang, and Y.-K. Ng, Record-boundary Discovery in Web Documents. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'99).* pp. 467–478 (Philadelphia, PA, 1999).

[10] G.H. Gonnet, R.A. Baeza-yates, and T. Snider, New Indices for Text: Pat Trees and Pat Arrays. *Information Retrieval: Data Structures and Algorithms* (Prentice Hall, 1992).

[11] D. Gusfield, *Algorithms on Strings, Trees, and Sequences* (Cambridge, 1997).

[12] S. Grumbach, and G. Mecca, In Search of The Lost Schema, *Database Theory — ICDT '99, the seventh International Conference*, pp. 314–331 (1999).

[13] C.-N. Hsu, and C.-C. Chang, Finite-State Transducers for Semi-Structured Text Mining. *Proceedings of IJCAI-99 Workshop on Text mining: Foundations, Techniques and Applications*, pp. 38–49 (Stockholm, Sweden, 1999)

[14] C.-N. Hsu, and M.-T. Dung, Generating Finite-state Transducers for Semi-structured Data Extraction From the Web. *Information Systems.* Vol. 23, No. 8, pp.521–538 (1998).

[15] C. Knoblock, S. Minton, J. Ambite, et al. Modeling Web sources for Information Integration. *Proceedings of the 15th National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference*, pp. 211-218 (Wisconsin, USA, 1998)

[16] D.E. Knuth, J.H. Morris, and V.B. Pratt, Fast Pattern Matching in Strings, SIAM J. Comput., Vol. 6, pp.323–350 (1977)

[17] S. Kurtz, and C. Schleiermacher, REPuter: fast computation of maximal repeats in complete genomes. *Bioinformatics* Vol. 15, No.5, pp.426–427 (1999).

[18] N. Kushmerick, Gleaning the Web. *IEEE Intelligent Systems*, Vol. 14, No. 2, pp. 20–22 (Mar/Apr., 1999).

[19] N. Kushmerick, D. Weld, and R. Doorenbos, Wrapper Induction for Information Extraction, *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (Japan, 1997).

[20] D.R. Morrison, PATRICIA–Practical algorithm to retrieve information coded in alphanumeric. *Journal of ACM*, Vol. 15, No. 4, pp.514–534 (Jan. 1968).

[21] I. Muslea, S. Minton, and C. Knoblock, A Hierarchical Approach to Wrapper Induction,*Proceedings of the 3rd International Conference on Autonomous Agents* (Seattle, WA, 1999).

[22] I. Muslea, Extraction Patterns for Information Extraction Tasks: A Survey, *Proceedings of AAAI'99: Workshop on Machine Learning for Information Extraction* (1999).

[23] E. Selberg, O. Etzioni, Multi-engine Search and Comparison Using the MetaCrawler, *Proc. of the Fourth Intl. WWW Conference* (Boston, USA, 1995).

[24] S. Soderland, Learning information extraction rules for semi-structured and free text, *Machine Learning*, Vol. 34, No. 1-3, pp. 233-272 (Jan. 1996).

[25] S. Soderland, Learning to Extract Text-based Information From the World Wide Web, *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (CA, 1997).

[26] The Deep Web: Surfacing Hidden Value, BrightPlanet.com LLC, `http://www.completeplanet.com/tutorials/deepweb/index.asp` (July, 2000).

[27] The World-Wide Web Consortium (W3C), Extensible Markup Language (XML), `http://www.w3.org/XML/` (1997)

[28] Web Design Group. Wilbur – HTML 3.2 `http://www.htmlhelp.com/reference/wilbur/`

[29] W.L. Warnick, A. Lederman, and R.L. Scott et al. Searching the Deep Web — Directed Query Engine Applications at the Department of Energy, *DLib Magazine*, Vol. 7, No. 1 (Jan. 2001).