ELSEVIER

Decision Support Systems 35 (2003) 129–147

Decision Support Systems

www.elsevier.com/locate/dsw

Automatic information extraction from semi-structured Web pages by pattern discovery

Chia-Hui Chang [a,*], Chun-Nan Hsu [b], Shao-Cheng Lui [c]

[a] *Department of Computer Science and Information Engineering, National Central University, Chungli, Tauyuan 320, Taiwan*
[b] *Institute of Information Science, Academia Sinica, Nankang, Taipei 115, Taiwan*
[c] *ChungHwa Telecommunication Laboratories, Yangmei, Tauyuan 326, Taiwan*

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 be generalized over unseen pages from the same Web data source. We empirically evaluate the performance of IEPAD on an information extraction task from 14 real Web data sources. Experimental results show that with the extraction rules discovered from a single page, IEPAD achieves 96% average retrieval rate, and with less than five example pages, IEPAD achieves 100% retrieval rate for 10 of the sample Web data sources.

Keywords: Information extraction; Semi-structured data; 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, and the problem of information extraction *from a Web page* is to apply information extraction to Web pages. 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 [26], which organize search results from multiple search engines for users, and shopping agents [11], which compare prices of the same product from multiple Web merchants.

Fig. 1 illustrates an example of this problem. In this example, we have a Web page containing the search results of the keyword "genome" from a search engine.

* Corresponding author.
E-mail addresses: chia@csie.ncu.edu.tw (C.-H. Chang), chunnan@iis.sinica.edu.tw (C.-N. Hsu), anyway@db.csie.ncu.edu.tw (S.-C. Lui).

0167-9236/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0167-9236(02)00100-8

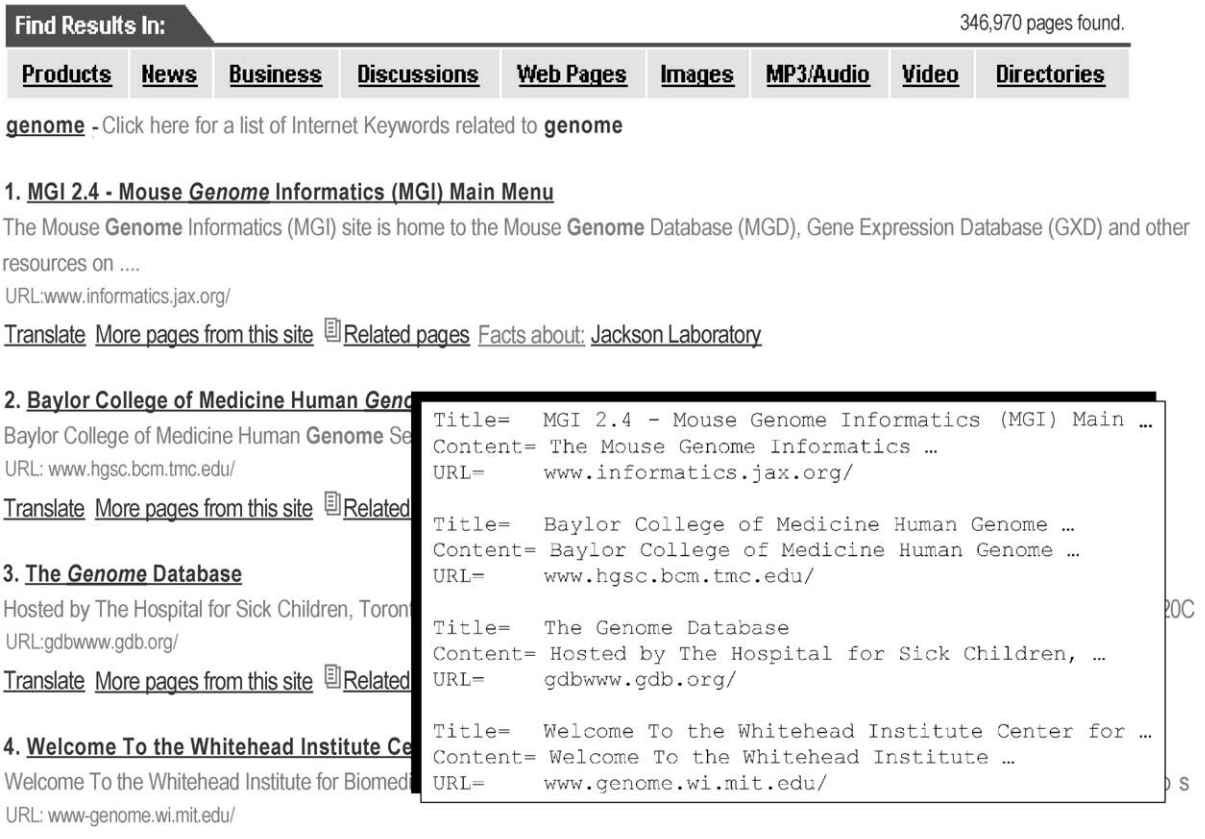

Fig. 1. Information extraction from a semi-structured Web page.

The goal is to extract the contents of this Web page into structured data records, as shown in the box in Fig. 1. In this example, there are four records in this Web page, each contains three data attributes (or information slots): Title, Content and URL. The structured data can then be fed into other applications as input, such as stored in a RDBMS database for data mining or translated into WML (Wireless Markup Language) for wireless Internet access.

Information extraction has been studied for years (see, e.g., Ref. [10]), but mostly concentrated on unstructured text (i.e., text expressed in natural language sentences). In that case, linguistic knowledge such as lexicons and grammars can be useful. However, a huge volume of information on the Web is rendered in a *semi-structured* manner, that is, in tables, itemized, enumerated lists, and the like, where linguistic knowledge provides limited hints. We call this type of Web pages as *semi-structured Web pages* [17]. An important difference of semi-structured Web pages and unstructured Web pages is that the layout formats of semi-structured Web pages are unique for different Web sites. Virtually no general grammar can describe all possible layout formats so that we can have one extractor for all semi-structured Web pages. As a result, each format may require a specific extractor, which makes it impractical to program extractors by hand.

Previously, researchers proposed several *wrapper induction* approaches for the rapid generation of extractors for Web pages [14,16,17,21,24,27]. 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.

A training example "teaches" a wrapper induction program how to extract data from a given Web page by providing examples of the following information:

1. how to partition a Web page (a long string of HTML tags and text) into meaningful substrings, and
2. how to group these substrings into attributes and data records.

Usually, wrapper induction programs are supplemented by a GUI for users to click and highlight strings on a rendered Web page to produce a training example. This procedure of clicking and highlighting is referred to as "labeling." Since generating a correct extractor may require many training examples, labeling can be tedious and labor intensive. As a result, wrapper induction only solves the problem partially. We need an approach that eliminates or minimizes the need of labeling.

Our work attempts to eliminate the need of user-labeled training examples. More precisely, users of IEPAD do not need to provide labeled examples as described above to tell IEPAD what information to extract, but simply choose among patterns to see if the pattern can extract the desired information. The idea is based on the fact that data on a semi-structured Web page is often rendered in some particular layout format with a regular and contiguous pattern. By discovering such patterns in target Web pages, an extractor can be generated. We design a pattern discovery algorithm that can be applied to any semi-structured 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.

1.2. The IEPAD architecture

Our approach to Web page information extraction has been implemented into a system called IEPAD (an acronym for Information Extraction based on PAttern Discovery). Fig. 2 shows the component diagram of IEPAD, which consists of three components:

• *Pattern discoverer* accepts an input Web page and discovers potential patterns that contain the target data to be extracted.

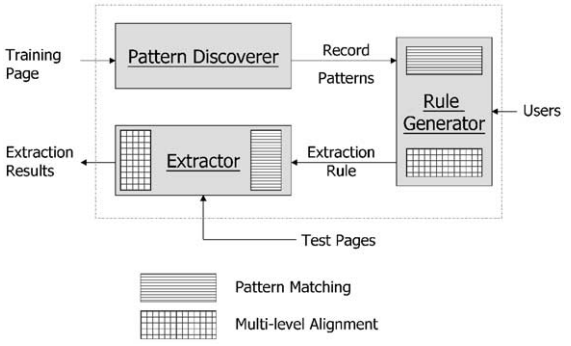

Fig. 2. The IEPAD architecture.

• *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.

The key idea of IEPAD is to discover and use patterns to extract data from target Web pages. A *pattern* is a subclass of regular expressions over an alphabet of tokens. Each pattern matches a set of strings. A pattern may contain options and alternatives. The simplest pattern is a string of tokens. An example of the pattern is given below:

$$\text{“} < \texttt{P} >< \texttt{A} >< \texttt{TEXT} >< /\texttt{A} >< \texttt{BR} > [< \texttt{TEXT} >] < \texttt{BR} >< \texttt{TEXT} >< \texttt{BR} >< \texttt{TEXT} > \text{”}, \quad (1)$$

where <P>,
, <TEXT>, etc., are tokens that match HTML tags <p>,
, and text strings, respectively. Options are denoted by [...]. In this example, the sixth token <TEXT> is optional. The following string matches this pattern:

$$< \texttt{p} >< \texttt{a href} = \text{“}\texttt{http} : //\texttt{www.csie.ncu.edu.tw} \text{”} > \texttt{NCU} < /\texttt{a} >< \texttt{br} > \texttt{National Central University} < \texttt{br} > \texttt{Chung} - \texttt{Li} < \texttt{br} > \texttt{Taiwan}. \quad (2)$$

The IEPAD extractor basically works as follows. Given a pattern and a Web page, the extractor scans the Web page to find all substrings that matches the pattern, and outputs the substrings as data records. In this case, the extractor will output (2) given the example pattern (1). Removing the HTML tags, we obtain a data record with the four text strings in (2) as the attributes:

⟨‘‘NCU’’, ‘‘National Central University’’, ‘‘Chung – Li’’, ‘‘Taiwan’’⟩.

The IEPAD pattern discoverer reverses the task of the extractor. The discovered patterns are usually sufficient to extract structured data from a Web page. The need of the rule generator is to enhance attribute extraction in each record and improve the speed of extraction by reusing the extraction rules for Web pages from the same Web site, which usually returns Web pages with the same pattern. There is no need to rediscover the pattern for each extraction.

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 rule generator and the extractor. Section 4 reports experimental results. Section 5 compares our work with related work. Finally, we draw the conclusion and discuss directions of future work in Section 6.

2. Pattern discoverer

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 [[13,23]] 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.

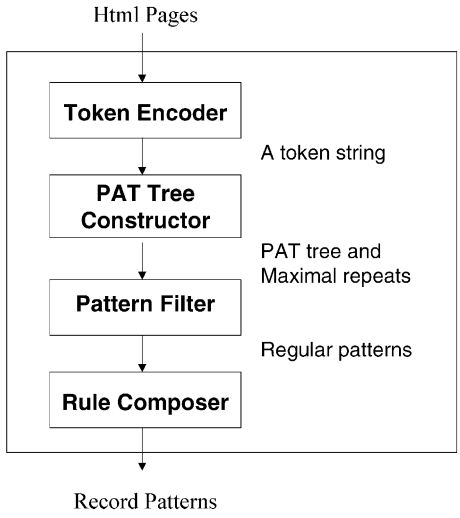

Fig. 3. Flow chart of the pattern discoverer.

Fig. 3 gives the flowchart of the pattern discovery process. Given a Web page, the token encoder will tokenize the page into a string of abstract representations, referred to as a *token string*. Each token is represented by a binary code of fixed length. The PAT tree constructor [13,23] 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.

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 Fig. 4 for a classification of block level tags and text level tags [30]). 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 text in a text block. The tag classifications allow different HTML translations to be generated. Users 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 keeping all tags. As shown in Fig. 5(a), the Congo code (an example used in Ref. [22]) can be translated into a string of 13 tokens using block-level encoding.

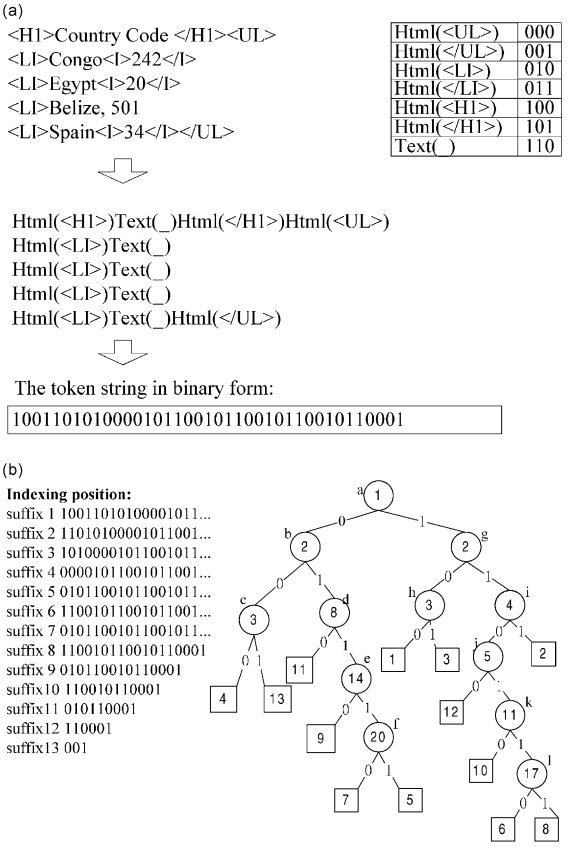

Fig. 5. The Congo code and its PAT tree.

2.2. Constructing PAT trees for maximal repeats

This section describes how to apply the PAT-tree technique to discover repeated patterns in a Web page. Structured data are usually organized as entries in a list or table in a Web page for clear presentation. These entries form repeated patterns and therefore, to extract the structured data from a Web page, an important step is to discover the repeated patterns. We define a *repeat* as any substring that occurs at least twice in the long string. Suppose a Web page contains k entries of data, we can extract these data entries by discovering the patterns that occurs k times in the target Web page.

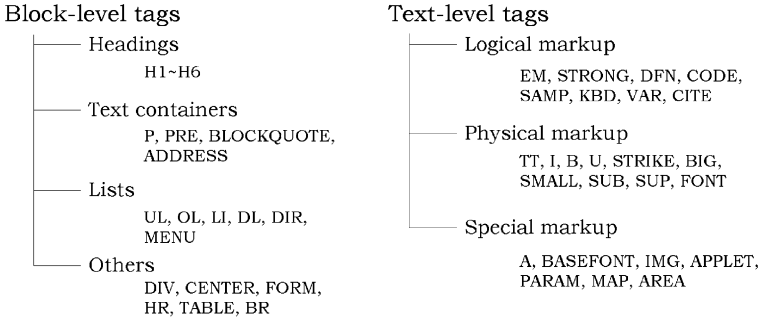

Fig. 4. Tag classification.

To better capture the idea of repeats and also reduce the number of candidate patterns, the concept of *maximal repeats* is used to refer to the longest patterns. We call a repeat *left maximal* (right maximal) if extending the repeat to the left (right) will not result in a new repeat for all repeat occurrences (see Ref. [4]). We say that 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 $p_1, p_2, \ldots, p_k$ 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, $1 \le i < j \le k$ (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, $1 \le x < y \le k$ (called *right maximal*).

Note that α is a string and $|\alpha|$ denotes its length. 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 [23]) 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 [15], 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 [20] have used suffix trees in bioinformatics for finding tandem repeats in genomes. It has also been used in Chinese keyword extraction [9] 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 Fig. 5(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 Fig. 5(b), a PAT tree is constructed from the encoded binary string of the sample example. The tree is constructed from 13 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. [15,23]. 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, in Fig. 5(b), 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 be 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:

$$\text{variance}(\alpha) = \frac{\sigma(\{d_i \mid 1 \le i < k, d_i = p_{i+1} - p_i\})}{(p_k - p_1)/(k-1)}. \tag{3}$$

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

$$\text{density}(\alpha) = \frac{(k-1) \times |\alpha|}{p_k - p_1}. \tag{4}$$

Generally speaking, machine-generated Web pages often render the data in templates with small variances and large densities. To sift candidate patterns, a simple approach is to apply different thresholds 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.

2.3.1. Occurrence clustering

The above approach can be implemented easily. However, it may 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, the search result pages of the search engine Lycos have advertisement banners inserted between the search result entries and divide the occurrences of the target pattern into several groups. As a result, the maximal repeats for Lycos' output pages may have large variance.

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 a regular block in which the pattern has a variance less than the threshold. The idea here is to cluster the occurrences into partitions so that the pattern discoverer can examine each partition. A simple loop can accomplish this one-dimension clustering. Let $C_{i,j}$ denote the list of occurrences $p_i, p_{i+1}, \ldots, p_j$ 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 and starts a new partition 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, the pattern as well as the occurrences in this partition are exported. To reduce the number of candidate patterns, the threshold is set to a small value close to zero than the variance threshold.

2.4. Composing extraction patterns

In addition to large variance, patterns with density less than 1 cause another problem. Since PAT trees compute only "exact match" patterns, templates with exceptions cannot be discovered by PAT trees. Therefore, we apply another technique called *multiple string alignment* to handle inexact or approximate matching.

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 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*":

a d c – b d

a d c x b –

a d c x b d

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 [17]. 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 $O(n \times m)$ in time by dynamic programming to obtain optimal edit distance, where n and m are string lengths. Extending dynamic programming to multiple string alignment yields an $O(n^k)$ algorithm. Alternatively, an approximation algorithm with much less time complexity is available such that the score of the multiple alignment is not greater than twice the score of the optimal multiple alignment [15]. 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 [15] 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 alignment results in extraction patterns with too many alternatives, such a pattern is unlikely to be interesting. Therefore, we set an upper bound of the most mismatches allowed.

2.4.1. Pattern rotation

An additional step after the pattern composition is pattern rotation. Let Σ be the alphabet of an encoding scheme. Suppose a generalized pattern is expressed as "$c_1c_2c_3\ldots c_n$", where each c_i is either a symbol or a subset of $\Sigma \bigcup \{-\}$ containing symbols that can appear at position i. Since c_1 might not be the beginning 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 c_j and generate a new pattern "$c_jc_{j+1}\ldots c_nc_1\ldots c_{j-1}$". If the left character of the first occurrence is the same as c_n, we rotate the pattern to "$c_nc_1c_2\ldots c_{n-1}$". The process will be continued until the last token has alternative options or 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 first token is a token with no option, and the right character of the last occurrence is the same as the first token, we rotate the pattern into "$c_2c_3\ldots c_nc_1$". The process will be continued until the first token has alternative options or 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 $\mathscr{T}$. 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 partitions 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 correspond 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 provide a graphic user interface so that the user can view the extracted contents by different patterns and select the desired pattern. 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 tell 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 testing Web pages by data extractor.

Fig. 6 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 that in Fig. 6(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 Fig. 6(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 such as <A> and <IMG> 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 Fig. 7. In summary, if there are m text tokens and special tag tokens in a record extraction pattern, all matched token strings can be divided into m blocks. For example, as shown in Fig. 6(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, a post-processing step must be conducted for the content in each block. Sometimes, by applying lower level encoding scheme, we can prevent the need of post-processing. However, it becomes much difficult to discover and compose the record pattern since the success of the pattern discovery approach depends on good abstraction of the Web pages.

To extract finer contents, the idea of *multiple string alignment* and block division are employed 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 Fig. 8. In the multilevel 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.

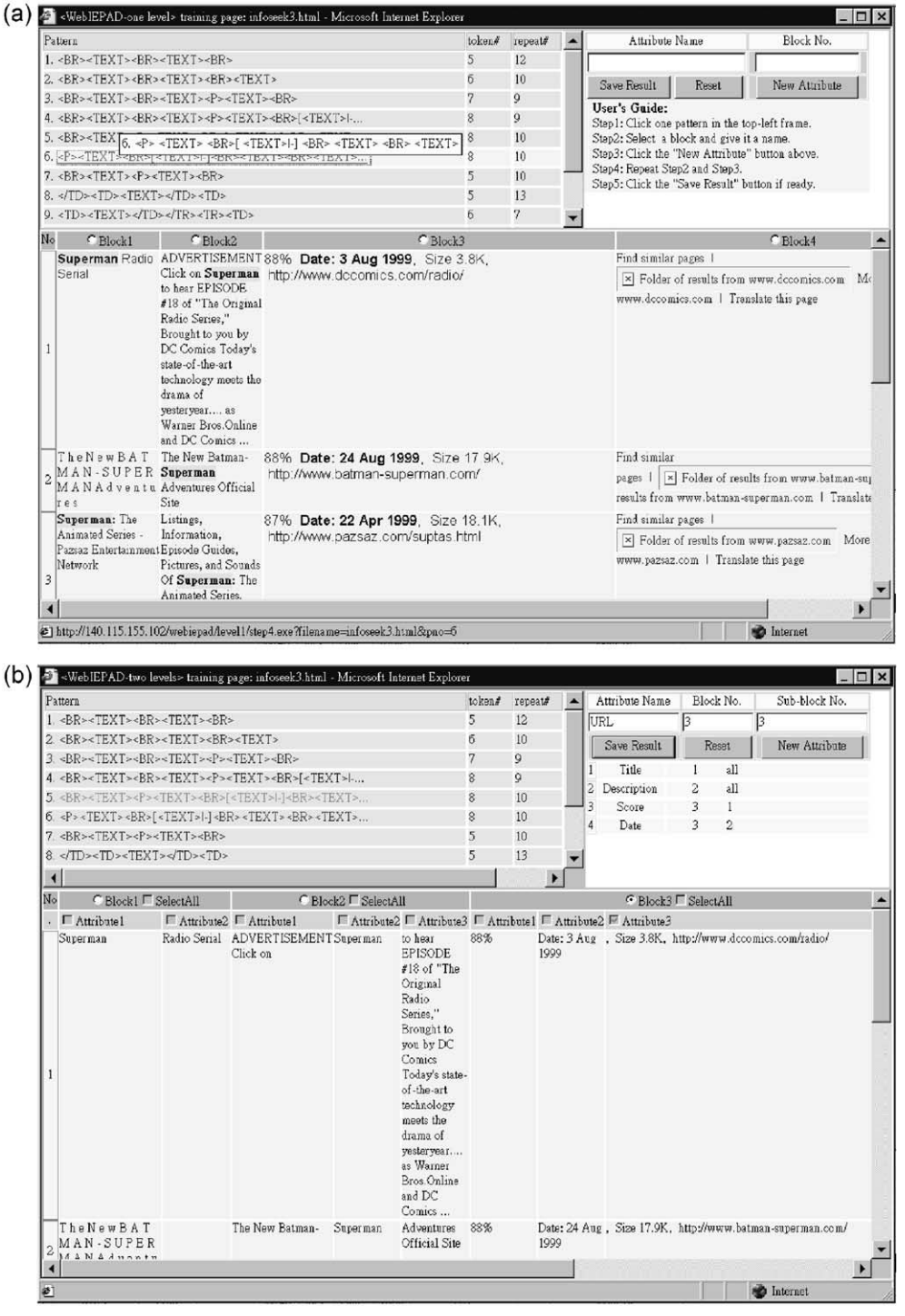

Fig. 6. The record patterns and the user interface (a) one-level alignment (b) two-level alignment.

Algorithm Block_division
Input: α = the extraction pattern containing k <TEXT> (or <A> or <IMG>) tokens;
R = the m records matched by pattern α;
Output: B = a $m \times k$ matrix which stores the divided records;
Method:
1. For each record r_i in R,
2. r_i'= Align(α, r_i);
3. For each <TEXT> in the extraction rule α,
4. Extract the corresponding data in r_i';
5. Store the divided record r_i' to $B[i]$;
6. Return matrix B;

Fig. 7. Block division procedure.

For example, the text contents that belong to the third block in Fig. 6(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 subblocks, 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.

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 Fig. 9. 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. Standard pattern matching algorithms, like the Knuth–Morris–Pratt's algorithm [19] or Boyer–Moore's algorithm [2] are sufficient. Note that each extraction rule composed by multiple string alignment actually represents several patterns since the patterns are expressed in regular expression with alternatives.

Algorithm MultiLevel_Alignment
Input: B = a $m \times k$ matrix;
$\prod$ = an encoding scheme;
Output: B' = a $m \times k'$ matrix;
Method:
1. For each column j in B,
2. $R = \{\}$;
3. For each row i in B,
3a. Encode $B[i][j]$ by $\prod$;
3b. Add the encoded token string to R;
4. Conduct Center_Star_Multiple_Alignment(R) to get the new pattern α_j;
5. Call Block_division(α_j, R);
6. Concatenate k blocks returned by step 5;
7. Return the concatenated matrix;

Fig. 8. Multilevel alignment.

Algorithm Extractor;
Input: P = an input Web page;
Λ = an extraction rule;
Method:
1. Encode the input page P by encoding scheme specified in $\Lambda.\Theta_1$;
2. R= Call Boyer_Moore algorithm to match all occurrences of the pattern $\Lambda.\alpha$ in the encoded token string of page P;
3. B= Call Block_division($\Lambda.\alpha$, R);
4. For each level j;
5. B= Call MultiLevel_Alignment(B, $\Lambda.\Theta_j$);
6. Extract designated information slots for each attribute;

Fig. 9. Extractor procedure.

In other words, there are alternative routes. Therefore, several can apply when matching the rule against the translated token sequence. 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

Fig. 10. Two-level attribute value extraction (test page: infoseek4.html).

call of multilevel 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. Fig. 10 shows that a test page (`infoseek4.html`) is uploaded and the extraction results are shown in the lower window.

4. Experimental results

The experiments here use two test data sets. The first one contains Web pages from 10 popular search engines. We collect 100 Web pages for each data source. The data set can be downloaded from http://www.csie.ncu.edu.tw/~chia/webiepad.html. The second data set are Okra, IAF, BigBook, and QuoteServer, taken from Kushmerick's work [21]. These four sources have also been used in Hsu and Chang [16] and Muslea [24] for the purposes of performance 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 28 K bytes and 11 K bytes for the above two data sets, respectively. The search results of the first data set typically contain at least 10 data records and advertisements, while the 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 (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.

Table 1
Data description

Data source	Number of records	Number of attributes	Missing	Unordered	Layout template	
					Variance	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

Table 2
Data size with different encoding schemes

Data set / Encoding	Search engines (Doc size = 28 K)		Okra, etc. (Doc size = 11 K)	
	Number of tokens	Percentage (%)	Number 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

4.1. Parameters setup

The input parameters to pattern discoverer include the encoding scheme, the minimum pattern length, the minimum occurrence count k, 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 [5]. The default value for the minimum length of maximal repeats and the minimum occurrence count are set to three and five, respectively.

In order to set the parameters used for variance and density, 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 Eq. (3). Similarly, we can compute the density of the common prefix of the layout template by Eq. (4). 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 output maximal repeats remained is about five 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 actual variance and density of its layout template, the number of output maximal repeats can be reduced to two (the "adaptive" column of Table 3). This shows that filtering candidate patterns based on variance and density thresholds is effective because without this filtering process, the number of maximal repeats discovered in a block-level encoded token string can be too large. These maximal repeats, after multiple string alignment and pattern rotation, constitute the record patterns for final output. Note that the number of record patterns may 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 multilevel division. Comparing the extracted to the correct number of attributes in Table 3 (the fifth column) and Table 1 (the second column), 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 `<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 finer text 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 (in IR, this corresponds to *recall*) is defined as the ratio of the number of desired data

Table 3
Performance

Data source	Number of maximal repeats		Performance		Number of attributes	
			Retrieval	Accuracy		
	Fixed	Adaptive	(%)	(%)	One level	Two 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

records correctly extracted and the number of desired data records contained in the input text. Likewise, accuracy rate (corresponds to *precision*) 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 Fig. 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 testing Web pages. Therefore, we design the following experiments in a way similar to those usually conducted in machine learning. For each Web site in the first data set, we randomly select 30 pages as the training pages (30%) and use the remaining (70 pages) as the testing set. The rule generated using the training set will be used to extract information from the testing 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 specify the information block for attribute value extraction. The extraction rule will then be applied to other training pages to validate the performance. If the rule cannot extract all the records for a page, this page will then be fed to the system for the discovery of better 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 testing phase, the extraction rule is applied to testing set for performance evaluation.

For most Web sites (12 of the 14), 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, 30 training pages are sufficient to present the diversity for the system to generate alternative patterns. Note that not all 30 training pages are used to generate patterns. Only pages that cannot be 100% extracted are used to generate patterns. Choosing a conservative pattern with 100% accuracy rate, Fig. 11 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 3 in average and 10 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 misextracted page) is used, the retrieval rate reaches 99.10% in the training set. With five training pages, IEPAD achieves 100% retrieval rate for 12 data sets except for HotBot and Northern-Light.

The learning curves are much steeper than those produced by machine learning based approaches since the learning is based on all records in a page instead of one training example. The retrieval rate is not only high in the training set but also in the testing set, which shows that 30% training pages have exhibited enough variety in the display format for the system to generalize. As we can see in Fig. 11, the retrieval rate for the testing set reaches 96.35% for one training page and 98.65% for two training pages. Of the 14 Web sites, IEPAD achieves 100% retrieval rate for 10 of them with less than five training pages in the testing set. 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.

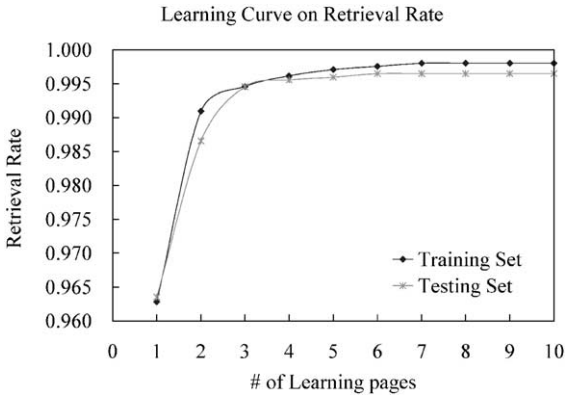

Fig. 11. Learning curve on training set and testing set.

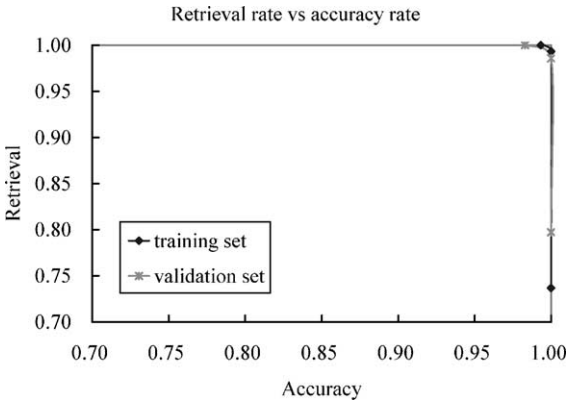

Fig. 12. Retrieval rate vs. accuracy rate for Northernlight.

Of course, such patterns may sometimes extract extra information because the patterns generalized from multiple string alignment may comprehend more patterns. There is a tradeoff. If retrieval rate is more important, accuracy rate will have to be sacrificed, just as the precision/recall trade-off in IR field.

We conduct another experiment to illustrate this point. When applying the rule over testing 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 Northern Light: one shorter pattern with no alternatives and one longer pattern with more alternatives. Sometimes, a compromised pattern can be found. As shown in Fig. 12, 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.

5. Related work

The research on information extraction from semi-structured Web pages can be traced to the research of *information agents* that integrate data sources on the World-Wide Web [6,7,11,18]. A 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) [6,7]. Since it is inappropriate to write extractors for all the Web data source, machine learning approaches are proposed to solve the problem.

Kushmerick, et al. [22] coin the term "*wrapper induction*" and describe a machine learning approach called WIEN [21,22]. Softmealy [16,17] is a wrapper induction algorithm that generates extraction rules expressed as finite-state transducers. Softmealy's extraction patterns are far more expressive than those generated by WIEN. 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 [25], which 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 the wrapper induction approaches with IEPAD, IEPAD is different from those approaches in many aspects. First, wrapper induction requires 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 [25] and multipass Softmealy [16]. Finally, a key difference between IEPAD and wrapper induction is that our extraction rules are based on context-based patterns while their approaches are based on *delimiters*. More precisely, their extraction rules use delimiters to determine which string on the Web page corresponds to a data attribute. For example, suppose we want to extract prices of digital cameras from a Web merchant. Each target data record to be extracted contains attributes `Brand`, `Model`, and `Price` of a digital camera model

(e.g., ⟨"Canon","S40",129.99⟩). A delimiter-based rule for extracting Price 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 the string " < blink>$129.99 < /blink>" in the input Web page will match this rule and "129.99" will be extracted as Price. In contrast, our extraction rule for the same task simply states that:

The strings containing attribute Price have this pattern : "< blink > $ < text >< /blink > ".

The reason we use context-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. job postings, flight schedules, query results from search engines, etc.). This naturally inspires the idea to discover such templates since the occurrences of these templates are usually aligned *regularly* and *contiguously* to allow for easy comprehension. These characteristics also allow for pattern discovery that automates the generation of the extraction rules 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 work in wrapper induction.

More recently, Chidlovskii, et al. [8] presented an approach to wrapper generation, which uses grammar induction based on string alignment. 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 aligned 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.

Fully automatic approach to information extraction is rare and often depends on some heuristics. For example, Embley et al. [12] 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 [12]. 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.

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 [28]. 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 [3,29] to refer to this huge 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.

6. Conclusion and future work

With the growing amount of online information, the availability of robust, flexible information extraction systems has become a stringent necessity. In this paper, we presented an unsupervised pattern discovery approach to semi-structured information extraction. This approach generates extraction rules for given Web pages. We also presented a multilevel alignment technique to extract finer data from discovered data records. Experimental results on real Web sites show

that IEPAD performs well for diverse structuring patterns.

The key features of this approach are as follows. First, the discovery of record boundaries can be completed automatically without user-labeled training examples. We consider this as the main contribution of this work, since labeling examples is the major bottleneck that deters commercial adoption of the wrapper induction approaches. Second, the discovered patterns can be generalized over unseen Web pages from the same Web data source regardless of various query topics. This is realized by two pattern discovery approaches: the PAT-tree technique that discovers repeated patterns from a Web page, and the multi-level alignment technique that partitions discovered patterns into data attributes. Comparing IEPAD to previous work, our approach 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 can tolerate layout exceptions such as missing attributes in the input.

This pattern discovery-based approach, however, still have several limitations. First, this approach cannot be applied to Web pages that contain only one data record. Second, the extraction rule for one Web data source generalized poorly to other Web data sources with different layout formats. Third, the number of pattern increases dramatically when there are too many layout exceptions since there will be too many alternative alignments. This case, however, is rare for script-generated Web pages.

The future work will follow two directions. The first one concerns the implementation of text level encoding to extract more delicate information. The second is to improve the filtering process of redundant patterns. We plan to incorporate semantic knowledge such as lexicons of application domains to filter discovered patterns. Finally, extending our work to help creating the Semantic Web will be an interesting topic.

Acknowledgements

The research reported here was supported in part by the National Science Council of Taiwan under Grant No.90-2213-E-008-042 and in part by DeepSpot Intelligent Systems, Taiwan under Contract No.05A-880527-03C with Academia Sinica, Taiwan. We wish to thank reviewers for their valuable comments.

References

[1] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific American 284 (5) (May 2001) 34–43.
[2] R.S. Boyer, J.S. Moore, A fast string searching algorithm, Communications of the ACM 20 (1977) 762–772.
[3] BrightPlanet.com LLC. The deep web: surfacing hidden value, http://www.completeplanet.com/tutorials/deepweb/index.asp, July, 2000.
[4] C.-H. Chang, S.-C. Lui, IEPAD: information extraction based on pattern discovery, Proceedings of the 10th International Conference on World Wide Web, Hong-Kong, Lecture Notes in Artificial Intelligence vol. 2336, Springer, 2001, pp. 223–231.
[5] C.-H. Chang, S.-C. Lui, Y.-C. Wu, Applying pattern mining to web information extraction, Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Hong Kong, China, ACM Press, 2001, pp. 4–15.
[6] S. Chawathe, H. Garcia-Molina, J. Hammer, The TSIMMIS project: integration of heterogeneous information sources, Proceedings of the 10th Meeting of the Information Society of Japan, Tokyo, Japan, 1994, pp. 7–18.
[7] B. Chidlovskii, U. Borghoff, P. Chevalier, Towards sophisticated wrapping of Web-based information repositories, Proceedings of the 5th International RIAO Conference, Montreal, Quebec, Canada, 1997, pp. 123–135.
[8] B. Chidlovskii, J. Ragetli, 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.
[9] 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, Philadelphia, PA, USA, ACM Press, 1997, pp. 50–58.
[10] Defense Advanced Research Projects Agency, Proceedings of the Sixth Message Understanding Conference (MUC-6), Morgan Kaufmann Publisher, San Francisco, CA, USA, 1995.
[11] R.B. Doorenbos, O. Etzioni, D.S. Weld, A scalable comparison-shopping agent for the world-wide web, Proceedings of the 1st International Conference on Autonomous Agents, New York, NY, USA, ACM Press, 1997, pp. 39–48.
[12] D. Embley, Y. Jiang, Y.-K. Ng, Record-boundary discovery in web documents, Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'99), Philadelphia, PA, USA, ACM Press, 1999, pp. 467–478.
[13] G.H. Gonnet, R.A. Baeza-Yates, T. Snider, New indices for text: PAT trees and PAT arrays, in: W.B. Frakes, R. Baeza-Yates (Eds.), Information Retrieval: Data Structures and Algorithms, Prentice Hall, New Jersey, 1992, pp. 66–82, Chapter 5.
[14] S. Grumbach, G. Mecca, In search of the lost schema, in: C. Beeri, P. Buneman (Eds.), Database Theory-ICDT '99, 7th International Conference, Proceedings, Lecture Notes in Com-

puter Science, vol. 1540, Springer, Jerusalem, Israel, 1999, pp. 314–331.
[15] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge Univ. Press, Cambridge, UK, 1997.
[16] C.-N. Hsu, C.-C. Chang, Finite-state transducers for semi-structured text mining, Proceedings of IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications, Stockholm, Sweden, Pergamon Press, NY, 1999, pp. 38–49.
[17] C.-N. Hsu, M.-T. Dung, Generating finite-state transducers for semi-structured data extraction from the web, Information Systems 23 (8) (1998) 521–538.
[18] 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, Madison, WI, USA, AAAI Press, Los Altos, CA, 1998, pp. 211–218.
[19] D.E. Knuth, J.H. Morris, V.B. Pratt, Fast pattern matching in strings, SIAM Journal on Computing 6 (1977) 323–350.
[20] S. Kurtz, C. Schleiermacher, REPuter: fast computation of maximal repeats in complete genomes, Bioinformatics 15 (5) (1999) 426–427.
[21] N. Kushmerick, Gleaning the web, IEEE Intelligent Systems 14 (2) (March/April 1999) 20–22.
[22] N. Kushmerick, D. Weld, R. Doorenbos, Wrapper induction for information extraction, Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI '97), Nagoya, Japan, Morgan Kaufmann, New Jersey, 1997, pp. 729–737.
[23] D.R. Morrison, PATRICIA—practical algorithm to retrieve information coded in alphanumeric, Journal of ACM 15 (4) (Jan 1968) 514–534.
[24] I. Muslea, Extraction patterns for information extraction tasks: a survey, Proceedings of AAAI'99: Workshop on Machine Learning for Information Extraction, AAAI Press, Menlo Park, CA, USA, 1999.
[25] I. Muslea, S. Minton, C. Knoblock, A hierarchical approach to wrapper induction, Proceedings of the 3rd International Conference on Autonomous Agents, ACM Press, New York, NY, USA, 1999, pp. 190–197.
[26] E. Selberg, O. Etzioni, Multi-Engine Search and Comparison Using the MetaCrawler, Proc. of the Fourth Intl. WWW Conference, Boston, USA, http://www.w3.org/Conferences/wwww4/, 1995.
[27] S. Soderland, Learning information extraction rules for semi-structured and free text, Machine Learning 34 (1–3) (1996) 233–272.
[28] The World-Wide Web Consortium (W3C), Extensible markup language (XML), http://www.w3.org/XML/, 1997.
[29] W.L. Warnick, A. Lederman, R.L. Scott, et al., Searching the deep web-directed Query engine applications at the department of energy, DLib Magazine 7 (1) (Jan 2000).
[30] Web Design Group, Wilbur-HTML 3.2, http://www.htmlhelp.com/reference/wilbur/, 1997.

Chia-Hui Chang is an Assistant Professor at the Department of Computer Science and Information Engineering, National Central University in Taiwan. She received her BS in Computer Science and Information Engineering from National Taiwan University, Taiwan in 1993 and got her PhD in the same department in January 1999. She worked postdoctoral in Chun-Nan Hsu's group after graduation, then joined National Central University from August 1999. Her research interests include information retrieval, knowledge discovery from databases, machine learning, and Web related research. She has one US patent pending. Her URL is http://www.csie.ncu.edu.tw/~chia/.

Chun-Nan Hsu is an Assistant Research Fellow at Institute of Information Science, Academia Sinica in Taiwan. He was Assistant Professor in the Department of Computer Science and Engineering at Arizona State University from 1996 to 1998, and Adjunct Assistant Professor in the Department of Computer Science and Information Engineering at National Chiao-Tung University from 1999 to 2000. He received his BS in Computer Engineering from National Chiao-Tung University, Taiwan in 1988, and both his MS and PhD in Computer Science from the University of Southern California, CA, USA in 1992 and 1996, respectively. His current research interests include machine learning, knowledge discovery and data mining, databases, intelligent Internet agents, and their applications in bioinformatics. He is a member of IEEE, ACM, and AAAI. He has served as the secretariat general of Taiwanese Association for Artificial Intelligence (TAAI) since 2001. He is on the Program Committee of AAAI-98, AAMAS-2002 and many other renowned academic conferences. He has two US patents pending. His URL is http://chunnan.iis.sinica.edu.tw/~chunnan/.

Shao-Cheng Lui is an Associate Researcher at Chunghwa Telecom in Taiwan. He received his BS in Computer Science and Information Engineering from Chung-Yuan Christian University, Taiwan in 1999 and MS in Computer Science and Information Engineering, National Central University, Taiwan in 2001. His current research interests include databases, intelligent systems on the Web. His URL is http://www.dbweb.csie.ncu.edu.tw/~anyway/.