IEPAD: Information Extraction Based on Pattern Discovery

Chia-Hui Chang
Dept. of Computer Science and Information Engineering
National Central University, Chung-Li, Taiwan 320
Tel: +886-3-4227151x4523
chia@csie.ncu.edu.tw

Shao-Chen Lui
Dept. of Computer Science and Information Engineering
National Central University, Chung-Li, Taiwan 320
anyway@db.csie.ncu.edu.tw

ABSTRACT

The research in information extraction (IE) regards the generation of wrappers that can extract particular information from semi-structured Web documents. Similar to compiler generation, the extractor is actually a driver program, which is accompanied with the generated extraction rule. Previous work in this field aims to learn extraction rules from users' training example. In this paper, we propose IEPAD, a system that automatically discovers extraction rules from Web pages. The system can automatically identify record boundary by repeated pattern mining and multiple sequence alignment. The discovery of repeated patterns are realized through a data structure call ***PAT trees***. Additionally, repeated patterns are further extended by pattern alignment to comprehend all record instances. This new track to IE involves no human effort and content-dependent heuristics. Experimental results show that the constructed extraction rules can achieve 97 percent extraction over fourteen popular search engines.

Keywords

Information extraction, extraction rule, PAT tree, multiple string alignment

1. INTRODUCTION

Current Web sites present information on various topics in various formats. A great amount of effort is often required for a user to manually locate and extract useful data from the Web sites. Therefore, there is a great need for value-added service that integrates information from multiple sources. For example, customizable Web information gathering robots/crawlers, comparison-shopping agents, meta-search engines and newsbots, etc. To facilitate the development of these information integration systems, we need good tools for information gathering and extraction. Suppose the data has been collected from different Web sites, a conventional approach for extracting data from various Web pages would have to write programs, called "*wrappers*" or "*extractors*", to extract the contents of the Web pages based on a priori knowledge of their format. In other words, we have to observe the extraction rules in person and write programs for each Web site. However, programming wrappers require manual coding which generally entails extensive debugging and is, therefore, labor-intensive. In addition, since the format of Web pages is often subject to change, maintaining the wrapper can be expensive and impractical.

Fortunately, researchers have built tools that can generate wrappers automatically. For example, WIEN [11], Softmealy [7], Stalker [13] etc. are three famous works in this field. Similar to scanner or parser generator for compilers where users provide the syntax grammar and get the transition tables for scanner or parser drivers, these wrapper construction systems actually output extraction rules from training examples provided by the designer of the wrapper. The common idea involved is the machine learning techniques to summarize extraction rules, while the difference is the extractor architectures presumed in each system. For example, the single-pass, LR structure in WIEN and the multi-pass, hierarchical structure in Stalker. Nevertheless, the designer must manually label the beginning and the end of the training examples for generating the rules. Manual labeling, in general, is time-consuming and not efficient enough.

Recently, researchers are exploring new approaches to fully automate wrapper construction. That is, without users' training examples. For example, Embley et al. describe a heuristic approach to discover record boundaries in Web documents by identifying candidate separator tags using five independent heuristics and selecting a consensus separator tag based on a heuristic combination [4]. However, one serious problem in this one-tag separator approach arises when the separator tag is used elsewhere among a record other than the boundary.

On the other hand, our work here attempts to eliminate human intervention by pattern mining [1]. The motivation is from the observation that useful information in a Web page is often placed in a structure having a particular alignment and order. Particularly, Web pages produced by search engines generally present search results in regular and repetitive patterns. Mining repetitive patterns, therefore, may discover the extraction rules for wrappers.

In this paper, we introduce IEPAD, an information extraction system applying pattern discovery techniques. In section 2, we present the system overview of IEPAD, including a pattern viewer, a rule generator and an extractor module. In section 3, we present the details of rule generator, followed by the embodiment of extractor in section 4. Finally, we present experimental results in section 5 and make the conclusion in section 6.

2. SYSTEM OVERVIEW

The system IEPAD includes three components, an **extraction rule generator** which accepts an input Web page, a graphical user interface, called **pattern viewer**, which shows repetitive patterns discovered, and an **extractor module** which extracts desired information from similar Web pages according to the extraction rule chosen by the user.

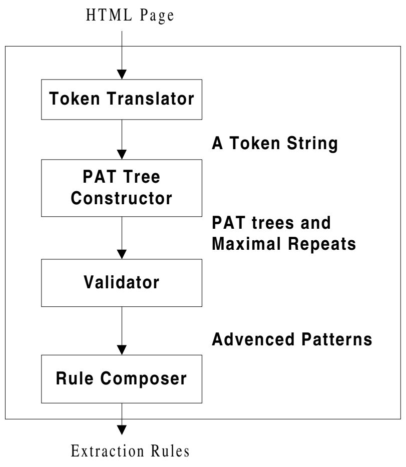


Figure 1. Extraction Rule Generator

The core techniques of pattern mining are implemented in the rule generator. The extraction rule generator includes a **translator**, a **PAT tree constructor**, a **pattern discoverer**, a pattern **validator**, and an extraction **rule composer**. The results of rule extractor are extraction rules discovered in a Web page. The graphical user interface can then enable users to view the information extracted by each extraction rule. Once the user selects a target extraction rule conforming to his information desire, the extractor module can use it to extract information from other pages having similar structure with the input page.

Referring to Figure 1, a flowchart of the rule extraction process is shown. The flowchart presents an overview of the mining process; more details will follow as to how a target pattern for extraction is generated. When a user submits an HTML page to IEPAD, the **translator** will receive the HTML page and translate it into a string of abstract representations, referred to here as tokens. Each token is represented by a binary code of fixed length *l*. The **PAT tree constructor** receives the binary file to construct a PAT tree. The pattern **discoverer** then uses the PAT tree to discover repetitive patterns, called maximal repeats. The maximal repeats are forwarded to **validator**, 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.

Once the extraction rules are discovered, the user may select from these candidate patterns his target pattern that contains desired information. The extractor receives a target pattern and a Web page as input and applies a pattern-matching algorithm to recognize and extract all occurrences of the target pattern in the token sequence of the encoded Web page. The extractor is a C++ module that can be linked to other information integration systems for information extraction. A typical pattern-matching algorithm, for example, is the Knuth-Morris-Pratt's algorithm. Each occurrence of the target pattern may represent a desired data record, and all of the desired data records form a useful *information block.*

3. EXTRACTION RULE GENERATOR

The motivation of IEPAD is based on the observation that desired information in a Web page is often placed in a structure having a particular alignment and forms repetitive patterns. The repetitive patterns, therefore, may constitute the extraction rules for wrappers. For a program to discover such repetitive patterns in a Web page, we need some abstraction mechanism to translate the HTML source to distinguish the display format (**translator**) and a pattern discovery algorithm to find out all repetitive patterns. By ***repetitive patterns***, we generally mean any substring that occurs at least twice in the encoded token string. However, such a definition will definitely include too many patterns fitting this requisite. Therefore, we define ***maximal repeats*** to uniquely identify the longest pattern as follows.

Definition: Given an input string S, we define **maximal repeat** α as a substring of S that occurs in k distinct positions $p_1, p_2, \dots, p_k$ 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 $(p_{i'}+|\alpha|)$th token is different from the $(p_{j'}+|\alpha|)$th token for at least one i', j' pair, $1 \le i' < j' \le k$ (called **right maximal**).

The definition of maximal repeats is necessary for identifying the well used and popular term repeats. Besides, it also captures all interesting repetitive structures in a clear way and avoids generating overwhelming outputs. However, not every maximal repeat represents a good one. Maximal repeats have to be further verified by the **validator** to filter interesting ones.

3.1 Translator

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

Tags tokens can be classified in many ways. The user can choose a classification depending on the desired level of information to be extracted. For example, tags in the BODY section of a document can be divided into two distinct groups: block-level tags and text-level tags. The former defines the structure of a document, and the latter defines the characteristics, such as format and style, of the contents of the text.

In Figure 2, a classification of block-level tags and text-level tags is shown. Block level tags are divided into different categories including headings, text containers, lists, and other classifications, such as tables and forms. Text-level tags are similarly divided into categories including logical markups, physical markups, and special markups for marking up texts in a text block.

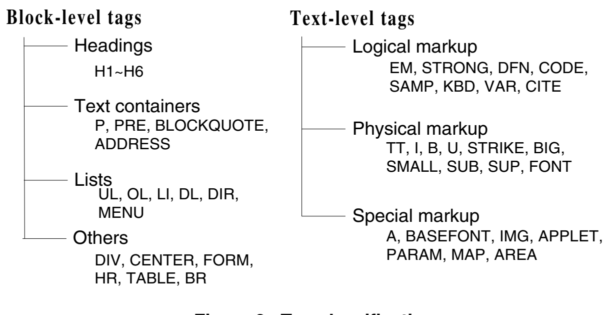

Block-level tags
Headings
H1~H6
Text containers
P, PRE, BLOCKQUOTE, ADDRESS
Lists
UL, OL, LI, DL, DIR, MENU
Others
DIV, CENTER, FORM, HR, TABLE, BR
Text-level tags
Logical markup
EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE
Physical markup
TT, I, B, U, STRIKE, BIG, SMALL, SUB, SUP, FONT
Special markup
A, BASEFONT, IMG, APPLET, PARAM, MAP, AREA

Figure 2. Tag classification

The many different tag classifications allow different HTML translations to be generated. With these different abstraction mechanisms, different patterns can be produced. For example, skipping all text-level tags will result in higher abstraction from the input Web page than when all tags are included. In addition, different patterns can be discovered and extracted when different encoding schemes are translated. The experiment results also indicate that block-level tags, while being a small percentage of the input size, contain a significant amount of useful information as shown in section 5.

For example, the translation process will be described using a simple Web page that contains only two lines of HTML source code (i.e., the Congo code):

<B>Congo</B><I>242</I><BR>
<B>Egypt</B><I>20</I><BR>$

The corresponding translation produced by the translator is a token string:

Html(<B>)Text(_)Html(</B>)Html(<I>)Text(_)Html(</I>) Html(<BR>)Html(<B>)Text(_)Html(</B>)Html(<I>)Text(_)Html(</I>) Html(<BR>)$, where each token is encoded as a binary string of "0"s and "1"s with length *l*. For example, suppose three bits encode the tokens in the Congo code:

Html(<B>)	*000*
Html(</B>)	*001*
Html(<I>)	*010*
Html(</I>)	*011*
Html(<BR>)	*100*
Text(_)	*110*

The encoded binary string for the token string of the Congo code is "000110001010110011100000110001010110011100$" of 3*14 bits, where "$" represents the ending of the encoded string.

3.2 PAT Tree Construction

Our approach for pattern discovery uses a PAT tree to discover repeated patterns in the encoded token string. A PAT tree is a Patricia tree (Practical Algorithm to Retrieve Information Coded in Alphanumeric [12]) constructed over all the possible suffix strings [5]. A Patricia tree is a particular implementation of a compressed binary (0,1) digital tree such that each internal node in the tree has two branches: zero goes to left and one goes to right. Like a ***suffix tree*** [6], the Patricia tree stores all its data at the external nodes and keeps one integer, the bit-index, in each internal node as an indication of which bit is to be used for branching. For a character string with *n* indexing point (or *n* suffix), there will be *n* external nodes in the PAT tree and *n–1* internal nodes. This makes the tree *O(n)* in size.

Referring to Figure 3, a PAT tree is constructed from the encoded binary string of the Congo example. The tree is constructed from fourteen sequences of bits, with each sequence of bits starting from each of the encoded tokens and extending to the end of the token string. Each sequence is called a "semi-infinite string" or "sistring" in short. Each leaf, or external node, is represented by a square labeled by a number that indicates the starting position of a sistring. For example, leaf 2 corresponds to sistring 2 that starts from the second token in the token string. Each internal node is represented by a circle, which is labeled by a bit position in the encoded bit string. The bit position is used when locating a given

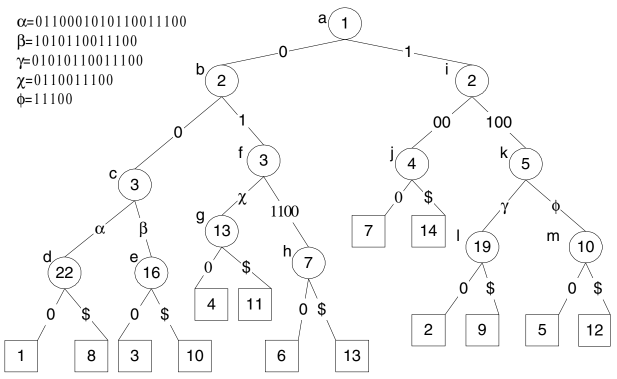


Figure 3. The PAT tree for the Congo Code

sistring in PAT tree. As shown in Figure 3, all suffix strings with the same prefix will be located in the same subtree.

Each edge in the PAT tree has a virtual edge label, which is the substring between two bit positions of the two nodes. For example, the edge label between node *k* and *m* is ϕ =11100, i.e. the substring from the fifth bit to the ninth bit of sistring (external node) 5 and 12. Edges that are visited when traversing downward from root to a leaf form a path that leads to a sistring corresponding to the leaf. In fact, we can uniquely identify a sistring by a prefix, which is formed by concatenating all the edge labels along the path from root to the corresponding leaf. The concatenated edge labels along the path form a virtual path label. For example, the edge labels "1", "00", and "0…" on the path that leads from root to leaf 7 form a path label "1000…", which is a unique prefix for sistring 7.

The PAT tree shown here is actually depicted for understanding. In implementation, there are only two pointers at each internal node. The index bit alone can distinguish the branches, zero goes left and one goes right. The edges labeled with a dollar sign are an indication of the end of the string. This edge uses the right link and pointed to the internal node itself. At the construction phase, sistrings are inserted one by one. To insert a sistring, we first search the sistring in the PAT tree and find the proper node that can distinguish the sistrings with others. Such an implementation requires more time than the optimal algorithm described in Gusfield's book, chapter 3 to 5 (linear), since we are using binary tree to store character string and only the character suffixes need to be indexed instead of all bit suffixes. However, binary trees have the advantage of simpler implementation.

3.3 Pattern Discoverer

All the leaves in a subtree share a common prefix, which is the path label for the path that leads from root to the root of the subtree. Each path label of an internal node represents a repeated sequence in the input. Therefore, to discover repetitive patterns, the discoverer only needs to examine path labels to determine whether or not they are maximal repeats. Since every internal node in a PAT tree indicates a branch, it implies a different bit following immediately the common prefix between two suffixes. Hence, every path label is right maximal. For a path label of an internal node *v* to be left maximal, at least two leaves in the *v*'s subtree should have different left characters. Let's call such a node ***left diverse***. Followed by definition, the property of being left diverse propagates upward in *T*. Therefore all maximal repeats in *S* can be found in linear time based on the following lemma [6].

Lemma The path labels of an internal node *v* in a PAT tree *T* is a *maximal repeat* if and only if *v* is left diverse.

The essence of a PAT tree is a binary suffix tree, which has also been applied in several research fields for pattern discovery. For example, Kurtz and Schleiermacher have used suffix trees in bioinformatics for finding repeated substring in genome [9]. As for PAT trees, they have been applied for indexing in the field of information retrieval since a long time ago [5]. It has also been used in Chinese keyword extraction [2] for its simpler implementation than suffix trees and its great power for pattern discovery. However, in the application of information extraction, we are not only interested in repeats but also repeats that appear regularly in vicinity.

By recording the occurrence counts and the reference positions in the leaf nodes of a PAT tree, we can easily know how many times a pattern is repeated and where it occurs. Thus, given the pattern length, occurrence count, we can traverse the PAT tree to enumerate all maximal repeats. For example, to find all patterns with the length greater than three tokens. Since each token is encoded with three bits, discoverer only needs to consider the internal nodes with index bit greater than 3*3. Therefore, only node "d", "e", "g", "l", and "m" are qualified.

The path labels and their respective token representations are candidates for useful repetitive patterns. Applying the definition of maximal repeat to the Congo example, only node "d" qualifies for a left diverse, since the left tokens for other leaves rooted at the same node all have the same token. For example, sistrings 3 and 10 which are rooted at node "e" have the same token Text(_). The token representation for the corresponding maximal repeat is, therefore, Html(<B>)Text(_)Html(</B>)Html(<I>)Text(_)Html(</I>) Html(<BR>).

The use of maximal repeats reduces the amount of unnecessary output produced by the discoverer. Besides, during the process of discovering the maximal repeats, the number of occurrences of the repeat and their respective positions in the input sequence can be easily derived for the **validator** to use.

In the above example, where the path label of node "d" is a maximal repeat, the subtree having node d as its root contains two leaves: leaf 1 and leaf 8. Because leaves 1 and 8 represent the only two sistrings that share the maximal repeat as their common prefix, the number of occurrences is two, and the positions of the maximal repeat in the input are one and eight in the input token sequence.

3.4 Pattern Validator

The above Congo source code is a simple example that shows how a maximal repeat can be discovered for a given input. However, a typical web page usually contains a large number of maximal repeats, not all of which contain useful information. To eliminate undesired maximal repeats, IEPAD uses the validator to determine whether or not the maximal repeats contain useful information. In addition to the occurrence frequency and pattern length of a maximal repeat, the validator employs a number of criteria, including regularity, compactness, and coverage. A user of IEPAD may choose to use only one of the criteria, or to use multiples of them in combination. Each of the criteria has a threshold that can either have a default value, or can be determined by the user. Suppose a maximal repeat α are ordered by its position such that suffix $p_1 < p_2 < p_3 \ldots < p_k$, where p_i denotes the position of each suffix in the encoded token sequence, we define regularity, compactness, and coverage as follows.

Regularity of a pattern is measured by computing the standard deviation of the interval between two adjacent occurrences ($p_{i+1}-p_i$). That is, the sequence of spacing between two adjacent occurrences is (p_2-p_1), (p_3-p_2), …, (p_k-p_{k-1}). Regularity of the maximal repeat α is equal to the standard derivation of the sequence divided by the mean of the sequence. A commonly used bound for regularity is 0.5.

Compactness is a measure of the density of a maximal repeat. It can be used to eliminate maximal repeats that are scattered far apart beyond a given bound. Using the example of maximal repeat α in the previous paragraph, density, is defined as $(k-1)*|\alpha|/\{p_k-p_1\}$, where $|\alpha|$ is the length of α in number of tokens. The criterion of compactness requires that only maximal repeats with a density greater than a given bound (default 0.5) be qualified for extraction.

Coverage measures the volume of content in a maximal repeat. Suppose the function $\mathcal{P}(i)$ returns the position of the *i*th sistring in the input Web page in number of bytes. Coverage is defined as $[\mathcal{P}(p_k+|\alpha|)-\mathcal{P}(p_1)]/|\text{Webpage}|$, where |Webpage| is the number of bytes of the input Web page.

These three measures are proposed because most information we would like to extract is presented in regular and contiguous format. Ideally, the extraction pattern should have regularity equal to zero, density equal to one and large coverage. To filter potentially good patterns, a simple approach will be to use a threshold for each of these measures above. Only patterns with regularity less than the regularity threshold and density between the density thresholds are considered candidate extraction patterns. Additionally, we can see that not every candidate maximal repeat is useful. For example, patterns that do not contain any text tokens are of no avail to users.

In this paper, we have utilized regularity and compactness to filter the discovered patterns. These two measures, together with coverage, can further be used to measure patterns' fitness for pattern ranking. The validation proceeds as follows: all discovered maximal repeats with regularity less than the regularity threshold γ (= 0.5) are considered potential and are forwarded to rule composer if they have density greater than δ (= 0.5), while others (patterns with density less than δ) are discarded. As for patterns with regularity greater than γ, special care is taken by the partition module described below.

3.5 Occurrence Partition

The regularity threshold 0.5 can extract information that is placed in a structure having a particular alignment and order. However, in examples such as ***Lycos*** (as will be discussed later), where the pattern of the target information occurs as three blocks in the Web pages (and forms three information blocks), the regularity can be large since it is measured over all instances of the pattern. Nonetheless, the regularity of the occurrences in each single block is still small. Therefore, the idea here is to partition the occurrences into segments so that we can analyze each partition respectively.

To handle such Web pages, these occurrences of such a pattern are carefully segmented to see if any partition of the pattern's occurrences satisfies the requirement for regularity. Generally, the regularity of a pattern is computed through all occurrences of the pattern. For example, if there are k occurrences, the $k-1$ intervals (between two adjacent instances) are the statistics we use to compute the standard deviation and the mean. To partition those occurrences, the occurrences are first sorted by their position. Let $G_{i,j}$ denote the ordered occurrences p_i, p_{i+1}, ..., p_j and initialize variables $s=1$, $j=1$. For each instance p_{j+1}, if the regularity of $G_{s,j+1}$ is greater than γ then we further consider whether $C_{s,j}$ is a good partition and assign $j+1$ to s. The pseudo code is as follows:

```
$S=\phi$, $s=1$;
For $j=1$ to $k-1$ do
        If Regularity($G_{s,j+1}$) > $\gamma$ then
            If Regularity($G_{s,j}$) > $\gamma_m$ then
                    $S= S \cup \{G_{s,j}\}$;
            endif
            s= j+1;
        endif
endf
```

If a partition includes occurrences more than the minimum count and has regularity less than threshold γ_m, the pattern as well as the occurrences in this partition are outputted. Note that the threshold γ_m is set to a smaller value close to zero to control the number of outputted patterns. With this modification, more Web pages such as *Lycos* can easily be handled.

3.6 Rule Composer

The main application of PAT tree is on the domain of exact match. To allow inexact, or approximate, matching (matching with some errors permitted), the technique for multiple string alignment is borrowed to extend the discovered exact repeats. The idea is to find a good presentation of the critical common features of multiple strings. For example, suppose "*adc*" is the discovered pattern for token string "*adcwbdadcxbadcxbdadcb*". If we have the following multiple alignment for strings "*adcwbd*," "*adcxb*" and "*adcxbd*":

a d c w b d
a d c x b -
a d c x b d

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

Multiple string comparison is a natural generalization of ***alignment*** for two strings which can be solved in $O(n*m)$ by ***dynamic programming*** where n and m are string lengths. Extending dynamic programming to multiple string alignment yields an $O(n^k)$ algorithm. Instead, an approximation algorithm is used such that the score of the multiple alignment is no greater than twice the score of optimal multiple alignment (see [5], Chapter 14). The approximation algorithm starts by computing the ***center string*** S_c in k strings $\boldsymbol{S}$ that minimizes *consensus error* $\sum_{P_i \in S} D(S_c, P_i)$. Once the center string is found, each string is then iteratively aligned to the center string to construct multiple alignment, which is in turn used to construct the extraction pattern.

For each pattern with density less than 1, the ***center star*** approximation algorithm for multiple string alignment is applied to generalize the extraction pattern. Note that the success of this approach lies in the assumption of contiguous repeats. If alignment of substrings between two adjacent occurrences results in extraction rules with alternatives at more than ten positions, such an alignment is ignored. In such cases, the original maximal repeats are kept instead of the aligned patterns.

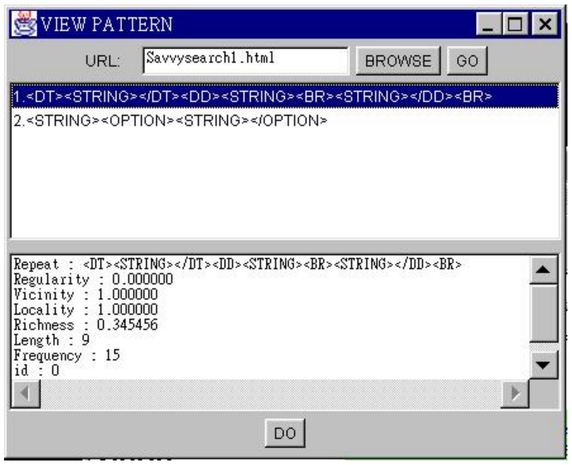
Another problem regarding the aligned extraction pattern is that the pattern is not necessarily located at the boundary of the information record. Suppose the generalized extraction pattern is expressed as "$c_1c_2c_3...c_n$", where each c_i is either a symbol $x \in \Sigma$ or a symbol set indicating more than one symbols can appear at position i (e.g. subset of $\Sigma \cup \{-\}$). If the actual starting position of a record is at c_j then the correct extraction pattern should be "$c_jc_{j+1}c_{j+2}...c_nc_1c_2...c_{j-1}$". Therefore, an additional step is taken to consider whether "$c_jc_{j+1}c_{j+2}... c_nc_1c_2...c_{j-1}$" is the pattern we need. Generally, such position j often has only one symbol. Besides, since extraction patterns often begin with tags like <DL>, <DT>, <TR> etc. and end up with tags such as <P>, <BR>, <HR> etc., we therefore use this guide[1] to generate possible extraction rules. However, the disadvantage here is that a large number of patterns can be produced during this rule generation step.

4. THE EXTRACTOR

After the extraction rules are constructed, the user may select from the ***pattern viewer*** one or more target patterns that contain desired information. Figure 4 is a demonstration of the *pattern viewer* where two patterns are discovered for *SavvySearch*. The upper frame shows the patterns discovered and the lower frame shows the detail measures of the selected pattern. The selected pattern is then forwarded to the extractor for pattern recognition and extraction. A screen shot of the extracted data when double click the pattern is shown in Figure 5.

The extractor is implemented as a module that can be linked into other information integration systems. There are two ways to complete the extraction work depending on whether a PAT tree is available for the input Web page. The extractor can search the PAT tree to find all occurrences of the extraction pattern. Or, the extractor can also apply a pattern-matching algorithm to recognize and extract all occurrences of the target pattern in the translated token string of unseen Web pages. Searching in a PAT is fast, since every subtree of a PAT tree has all its sistrings with a common prefix. Therefore, it allows efficient, linear-time solutions to complex string search problem. For example, string prefix searching, proximity searching, range searching, regular expression search etc.

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

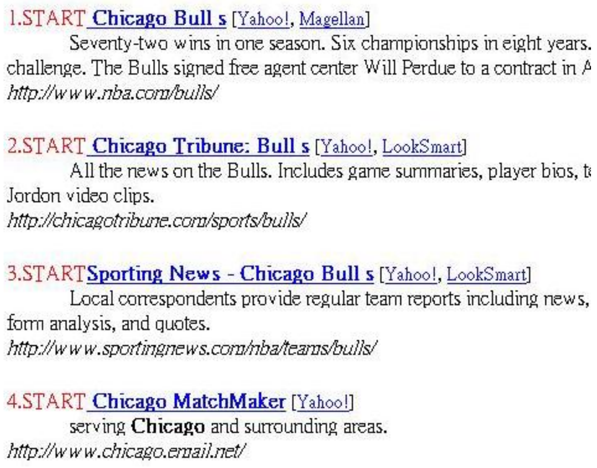


Savvysearch1.html

1.<DT><STRING></DT><DD><STRING><BR><STRING></DD><BR>
2.<STRING><OPTION><STRING></OPTION>

```
Repeat : <DT><STRING></DT><DD><STRING><BR><STRING></DD><BR>
Regularity : 0.000000
Vicinity : 1.000000
Locality : 1.000000
Richness : 0.345456
Length : 9
Frequency : 15
id : 0
```

Figure 4. Pattern Viewer



1.START Chicago Bull s [Yahoo!, Magellan]
Seventy-two wins in one season. Six championships in eight years. challenge. The Bulls signed free agent center Will Perdue to a contract in A
http://www.nba.com/bulls/

2.START Chicago Tribune: Bull s [Yahoo!, LookSmart]
All the news on the Bulls. Includes game summaries, player bios, t Jordon video clips.
http://chicagotribune.com/sports/bulls/

3.STARTSporting News - Chicago Bull s [Yahoo!, LookSmart]
Local correspondents provide regular team reports including news, form analysis, and quotes.
http://www.sportingnews.com/nba/teams/bulls/

4.START Chicago MatchMaker [Yahoo!]
serving Chicago and surrounding areas.
http://www.chicago.email.net/

Figure 5. Screen shot of the data extracted

On the other hand, if the pattern is used to extract unseen Web pages where no PAT tree has been constructed, pattern-matching algorithms can be applied. Typical pattern matching algorithms, for example, are the Knuth-Morris-Pratt's algorithm or Boyer-Moore's algorithm [15]. Alternatively, since the extraction rule is expressed as a regular expression (with concatenation and alternative only), it is easier to construct a finite-state machine for such an extraction rule.

Note that each extraction rule composed by multiple string alignment actually represents several patterns. That is, there are alternatives at each state of the non-deterministic finite-state machine. Therefore, several patterns can apply when matching the rule against the translated token sequence. In such cases, the longest match is considered. For example, extracting occurrences of the rule "ab[a|-][b|-]" in the string "abababab" will find only two occurrences of "abab".

The input to the extractor module is the Web pages to be extracted and the selected rule together with the encoding scheme used to discover the extraction rule. The extractor module first translates the Web pages using the encoding scheme specified. In addition to the encoded token string, we also record the starting positions in the source Web page for each token. Once the occurrences of the pattern in the encoded token string are found, the corresponding positions in the Web pages can be counted.

5. EXPERIMENTS

Extraction results of a rule are often evaluated by retrieval rate and accuracy rate. Retrieval rate is defined as the ratio of the number of desired data records enumerated by a maximal repeat to the number of desired data records contained in the input text. Accuracy rate is defined as the ratio of the number of desired data records enumerated by a maximal repeat to the number of occurrences of the maximal repeat. A data record is said to be enumerated by a maximal repeat if the matching percentage is greater than a bound determined by the user. The matching percentage is used because the pattern may contain only a portion of the data record. For example, suppose an input text contains 20 desired data records, and a maximal repeat that occurs 25 times enumerates 18 of them. Then the retrieval rate is 18/20 and the accuracy rate is 18/25. A higher retrieval rate indicates that more records are extracted.

We first show the number of maximal repeats validated by our system using fourteen state-of-the-art search engines, each with ten Web pages. The number of validated maximal repeats is sometimes an indicator of the extraction result. If the number of validated maximal repeats is 1, the maximal repeat is very likely the target pattern. When the number of the maximal repeats is increased, it implies that there is a great variance in the records and each discovered maximal repeat only captures part of the target information block.

Table 1. Number of maximal repeats validated

Search Engine	Maximal Repeats	Regularity < 0.5	Density	
			>0.25	<1.5
AltaVista	213.1	49.4	43.5	6.8
Cora	100.5	38.4	14.6	3.5
Excite	120.9	11.7	8.0	1.0
Galaxy	72.5	26.6	11.7	5.0
Hotbot	74.8	20.4	14.2	9.4
Infoseek	143.9	17.1	7.5	7.2
Lycos	210.0	18.0	5.6	3.9
Magellan	42.8	12.7	6.0	1.0
Metacrawler	190.7	34.5	14.6	6.8
NorthernLight	119.3	21.2	15.3	3.8
Openfind	76.2	36.9	9.5	2.4
Savvysearch	119.2	44.3	35.1	8.2
Stpt.com	72.5	39.5	33.2	3.7
Webcrawler	82.7	20.5	14.3	1.3
Average	117.1	27.9	16.7	4.7

There are several control parameters which can affect the number of maximal repeats validated, including the regularity threshold, the density thresholds, the minimum length and frequency of a pattern, and the encoding scheme, etc. The effect of different encoding schemes will be discussed in the next section. We first experiment with the all-tag-encoding scheme. Table 1 shows the number of maximal repeats validated with regularity smaller than 0.5, density between 0.25 and 1.5. The thresholds are decided empirically to include as many good patterns as possible. The effect of various regularity and density thresholds is shown in

Figure 6 with fixed minimum length and minimum frequency requirements set to 3 and 5, respectively.

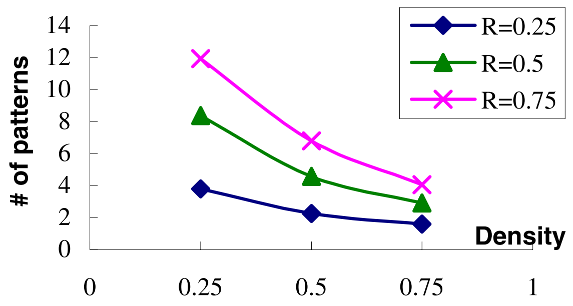


Figure 6. Number of Patterns validated

5.1 Encoding Scheme

The experiments in the previous section present the encoding scheme when all tag classes are involved in the translation (each tag is translated to their corresponding token class). In this section, four additional encoding schemes are conducted, which skip logical, physical, special and all text-level tags respectively (Figure 2). For example, the second encoding scheme skips physical markups, including <TT>, <I>, <B>, and <U>, etc. Because of space limitation, we show only the average performance for these encoding schemes. In Table 2, we show the comparison on the length of translated token string. Basically, the higher the abstraction level, the shorter the length. Table 3 shows the performance comparison of the last three encoding schemes. Note that the patterns evaluated here are maximal repeats past the validation criteria. The effects of occurrence partition and multiple string alignment are discussed later. The results of the first encoding scheme are not shown because logical markups are less used in HTML files (only 0.4%) and the difference is not obvious from all-tag encoding schemes. For the second encoding scheme, the performance is increased for *Cora*, *Metacrawler* and *Savvysearch*; while for the third encoding scheme, the matching percentage is increased for *Cora*, *Excite*, and *Metacrawler*. That is, though the encoding scheme of skipping some markups/tags enables the display of extraction pattern for some search engines, it may disable the patterns of other Web sites, especially when the pattern gets shorter than three tokens. However, high-level abstraction has better performance on average.

The experimental results indicate that block-level tags (skipping all text-level tags), while being a small percentage of the input size, contain a significant amount of useful information. In our experiment, the size of the token string having only block-level tags is 2 percent of the input size (Table 2); however, the extraction result obtained from the token string can extract 86 percent of the desired information block (Table 3). Note that for the block-level encoding scheme, the performance is increased for almost every search engine except for *Lycos*. *Lycos* has a decreased performance because the maximal repeats occur in three blocks and has regularity greater than 0.5 in block-level encoding scheme. We conclude that the block-level encoding scheme performs best among others. In addition, the token string for the block-level encoding scheme is only two percent (514 Bytes) of the original HTML file (22.7KB in average).

Table 2. Size of translated sequences and number of patterns

Encoding Scheme	Size of translated sequences	No. of Patterns discovered
All Tag	1128	7.9
No Physical	873	6.5
No Special	796	5.7
Block-Level	514	3.8

Table 3. Comparison of different encoding schemes

Encoding Scheme	Retrieval Rate	Accuracy Rate	Matching Percentage
All Tag	0.73	0.82	0.60
No Physical	0.82	0.89	0.68
No Special	0.84	0.88	0.70
Block-Level	0.86	0.86	0.78

5.2 Occurrence Partition and Multiple String Alignment

Since the block-level encoding scheme has better abstraction of the input Web page, the following experiments use this encoding to show the effect of multiple string alignment and segmentation. As discussed above, the block-level encoding scheme discovers no patterns for *Lycos* because its regularity is greater than the default threshold 0.5. Though larger threshold can keep such patterns, it may also include too many irregular patterns. Occurrence partition, on the other hand, provides the opportunity to resume those patterns with larger threshold while reserve only information blocks with really small threshold. Indeed, with the additional step of occurrence segmentation, we can successfully partition instances into several blocks and discover such patterns. Therefore, the performance is greatly increased to 92% retrieval rate and accuracy rate (Table 4). However, the number of patterns increased a lot for Web sites like *Lycos*.

Table 4 also shows that with the help of multiple string alignment the performance is improved to 97% retrieval rate, 94% accuracy rate and 0.90 matching percentage. The high percentage of retrieval rate is pretty encouraging. The 90% of matching percentage is actually higher in terms of the text content retrieved. For those Web sites with matching percentage greater than 85%, the text contents are all successfully extracted. What bothers is the accuracy rate. Since the extraction pattern generalized from multiple alignment may comprehend more than the information we need, the extractor may extract more than the desired information. The detail performance of each Web site is shown in Table 5 for reference.

Table 4. Effect of segmentation

Method	Retrieval Rate	Accuracy Rate	Matching Percentage
Block-level Encoding	0.86	0.86	0.78
Occurrence Segmentation	0.92	0.91	0.85
Multiple String Alignment	0.97	0.94	0.90

Table 5. The performance of multiple string alignment

Search Engine	Retrieval Rate	Accuracy Rate	Matching Percentage
AltaVista	1.00	1.00	0.91
Cora	1.00	1.00	0.97
Excite	1.00	0.97	1.00
Galaxy	1.00	0.95	0.99
Hotbot	0.97	0.86	0.88
Infoseek	0.98	0.94	0.87
Lycos	0.94	0.63	0.94
Magellan	1.00	1.00	0.76
Metacrawler	0.90	0.96	0.78
NorthernLight	0.95	0.96	0.90
Openfind	0.83	0.90	0.66
Savvysearch	1.00	0.95	0.97
Stpt.com	0.99	1.00	0.95
Webcrawler	0.98	0.98	0.98
Average	0.97	0.94	0.90

6. SUMMARY AND FUTURE WORK

The core technique of information extraction is the discovery of extraction rules. In earlier work, extraction rules are learned from training examples. In this paper, we presented an unsupervised approach to pattern discovery. We propose the application of PAT trees for pattern discovery in the encoded token string of Web pages. The discovered maximal repeats are further filtered by the measures: regularity and compactness. The basic pattern discovery module (without occurrence partition and multiple string alignment) can extract 86% records with block-level encoding scheme. The parameters of the thresholds for regularity and compactness have been controlled to keep as many good patterns while to filter out useless patterns.

For patterns with regularity greater than the default threshold, occurrence partition is applied to segment the occurrences into blocks. This additional step help to solve cases like *Lycos* whose pattern scatters among several blocks and has large regularity. Though higher threshold can reserve such patterns, it may also include too many irregular patterns. Occurrence partition, on the other hand, provides the opportunity to resume those patterns with greater threshold than default while reserve only information blocks with really small threshold.

Despite PAT trees' efficiency in "exact" string problems, "inexact" string problems are ubiquitous. Therefore, multiple string alignment is applied to patterns to generalize multiple records and express the extraction rule in regular expressions. In our experiments, the extraction rule generalized from multiple string alignment can achieve 97% retrieval rate and 94% accuracy rate with high matching percentage. The whole process requires no human intervention and training example. It takes only three minutes to extract 140 Web pages. Comparing our algorithm to others, our approach is quick and effective.

7. ACKNOWLEDGEMENT

This work is sponsored by National Science Council, Taiwan, under grant NSC 89-2213-E-008-056. Also, we would like to thank Lee-Feng Chien, Ming-Jer Lee and Jung-Liang Chen for providing their PAT tree code for us.

8. REFERENCES

[1] Chang, C.H.; Lui, S.C.; and Wu, Y.C. Applying pattern mining to Web information extraction. In Proceedings of the Fifth Pacific Asia Conference on Knowledge Discovery and Data Mining, Apr. 2001, Hong Kong.

[2] Chien, L.F. PAT-tree-based keyword extraction for Chinese information retrieval. In Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 50–58. 1997.

[3] Doorenbos, R.B.; Etzioni, O.; and Weld, D. S. A scalable comparison-shopping agent for the World Wide Web. In Proceedings of the first international conference on Autonomous Agents. pp. 39–48, NewYork, NY, 1997, ACM Press.

[4] Embley, D.; Jiang, Y.; and Ng, Y. -K. 1999. Record-boundary discovery in Web documents. In Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)}. pp. 467–478, Philadelphia, Pennsylvania.

[5] Gonnet, G.H.; Baeza-yates, R.A.; and Snider, T. 1992. New Indices for Text: Pat trees and Pat Arrays. Information Retrieval: Data Structures and Algorithms, Prentice Hall.

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

[7] Hsu, C.-N., and Dung, M.-T. 1998. Generating finite-state transducers for semi-structured data extraction from the Web. Information Systems. 23(8): 521–538.

[8] Knoblock, A. et al., Eds. 1998. In Proceedings of the 1998 Workshop on AI and Information Integration, Menlo Park, California. AAAI Press.

[9] Kurtz, S., and Schleiermacher, C. 1999. REPuter: fast computation of maximal repeats in complete genomes. Bioinformatics 15(5): 426–427.

[10] Kushmerick, N. 1999. Gleaning the Web. IEEE Intelligent Systems 14(2): 20–22.

[11] Kushmerick, N.; Weld, D.; and Doorenbos, R. 1997. Wrapper induction for information extraction. In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI).

[12] Morrison, D. R. Journal of ACM, 15, pp. 514–534, 1968.

[13] Muslea, I.; Minton, S.; and Knoblock, C. 1999. A hierarchical approach to wrapper induction. In Proceedings of the 3rd International Conference on Autonomous Agents (Agents '99), Seattle, WA.

[14] Muslea, I. 1999. Extraction patterns for information extraction tasks: a survey. In Proceedings of AAAI '99: Workshop on Machine Learning for Information Extraction

[15] Sedgewick, R. Algorithms in C, Addison Wesley, 1990.