

A NOVEL APPROACH OF TABLE DETECTION AND ANALYSIS FOR SEMANTIC ANNOTATION

ENHONG CHEN and SHU WANG

*Department of Computer Science, University of Science and Technology of China
Hefei Anhui 230027 P.R. China*

PHILLIP C.-Y. SHEU

Department of EECS, University of California, Irvine, CA 92697

Received 31 January 2005

Accepted 13 July 2005

Semantic web mining is getting more attention in intelligent web applications. Many web sites, especially those dynamically generate HTML pages to display the results of user queries, present information in the form of lists or tables. It is very useful to extract concept instances from these tables for many web applications such as intelligent agent systems for on-line product recommendations. This paper describes a technique for extracting data from tables in two steps, namely table detection and table analysis. The table detection step identifies the existence of a table and extracts its contents, and the table analysis step discovers the semantic meanings embedded in the table and associates them with the concepts described in the domain ontology that are used for semantic annotation on these tables. Our algorithm has been tested based on real-life web documents and the experimental results are encouraging.

Keywords: Semantic annotation; table detection; ontology.

1. Introduction

Capturing knowledge based on semantic annotations has been a major technique for creating metadata. It is beneficial for a wide range of content-oriented applications. For example, current research about the Internet has been striving to augment the syntactic information already present with semantic metadata in order to achieve a semantic web that human and software agents can understand. To this end one of the most urgent challenges is knowledge capturing, i.e., turning existing syntactic resources into knowledge structures. One possible solution is semantic annotation, i.e., to mark up the existing web documents. It is getting more and more attention in the semantic web community^{1,2,3,4}.

Most existing approaches have attempted to provide an annotation editor to facilitate the annotation process manually or semi-automatically. Some efforts were made to solve the problem automatically^{5,6}, with a main focus placed on the annotation of free texts. However, a large percentage of web documents contain data-rich tables or lists. Consider for example a web document shown in Figure 1. It is desired to annotate these tables when a search agent searches for the flights that will leave the JFK airport at 7:00am on

Friday. This is unfortunately not an easy task due to the complicated layout involved. The existing automatic annotation approaches^{5,6} cannot be used in this case. Reference 8 proposed a deep annotation framework that combines the presentation layer with the data description layer. But this approach involves certain requirements on the database and the sever side which may hardly be satisfied in reality. Reference 7 proposed an automatic approach by exploiting the structure of the corresponding web document, but it requires that some instances of the table already exist in the ontology instance base.



Scooby-Doo 2: Monsters Unleashed
[WS](2004)
Enhanced Widescreen (Soft Matte) DVD
[All Editions](#)

Member Rating **Rated: PG**
★★★★ **Release Date:**
 09/14/2004

America's favorite teenage canine-led crime fighters earn a second shot at the big screen in this sequel to the hit comedy Scooby-Doo. The reunited Mystery Inc. team -- Fred (Freddie Prinze Jr.),... [More](#)

Starring: [Freddie Prinze, Jr.](#), [Sarah Michelle Gellar](#), [Matthew Lillard](#), [More](#)





Man on Fire(2004)
Enhanced Widescreen Letterbox DVD
[All Editions](#)

Member Rating **Rated: R**
★★★★ **Release Date:**
 09/14/2004

A man whose ideals have been shattered for the last time is out for violent justice in this thriller. Creasy (Denzel Washington) is a former United State intelligence agent-mercenary who has... [More](#)

Starring: [Denzel Washington](#), [Dakota Fanning](#), [Christopher Walken](#), [More](#)



Fig. 1. A web document containing data-rich tables and lists

This paper proposes an automatic approach to extract table instances including their values and semantic meanings (ontology concepts). The semantic annotation process can be performed automatically by creating a set of semantic metadata from the cells of these tables using the related ontology concepts. Our approach includes table detection and table analysis techniques, and no specific restriction is required.

The table detection process identifies a table and extracts any instances from the table. Most existing table detection techniques have relied on machine learning^{9,10,15}. There are some problems associated with them however. For example, the approach proposed in Ref. 10 uses a page template to indicate where a table is located and uses a *separator* for cell extraction. But sometimes such page template does not exist and a user-defined *separator* can hardly fit different tables. In addition, column identification using *AutoClass* is not stable enough for practical applications.

Our table detection algorithm was developed based on the regularity of table structures. The algorithm uses a JDOM parser to parse a web document, and extracts the rows and columns of a table from the corresponding DomTree. We explore the DomTree in a bottom up fashion. A pattern comparison technique based on hashing is used to improve the efficiency. We use a set of *Consecutive Tree Patterns* to record the regularity of table structures and to extract the instances of a table by a set of *Maximum Consecutive Patterns*. Our algorithm employs a subtree comparison technique similar to the one used in Ref. 15 to mine a data region and extract its data records. However unlike the technique discussed in Ref. 15 which traverses a tag tree from the root downward in a depth-first fashion and executes the procedure CombComp is at each internal node, our algorithm only needs to explore the DomTree once and compare each pattern once.

For table analysis, the instances of a table are associated with the related concepts in the domain ontology (given) to accomplish the task of annotation. In this step, we distinguish two kinds of tables, namely self-described tables and non-self-described tables (defined in Section 3.2). For a self-described table, we extract a *Semantic Indicator* for each column by finding the maximum prefix for all the values in that column. We then find an appropriate ontology concept using the synonym chain available in the WordNet¹⁴ based on that semantic indicator. For a non-self-described table, three cases are considered, namely the head of the table, the linguistic pattern of the value in each cell and the link page of the value in the cell. A different strategy is provided in each case.

The rest of this paper is organized as follows: Section 2 first defines the table detection problem and describes our table detection algorithm. Section 3 discusses table analysis algorithm and shows the strategies for all possible situations. Section 4 presents our experiments and results. Section 5 concludes this paper.

2. Table Detection

In this section, we will introduce a table detection algorithm for identifying the structure of a table. Furthermore, it prepares the table analysis step to extract the contents from the table. Section 2.1 will introduce some basic definitions. Section 2.2 will give a comprehensive description of the algorithm. An example is presented in Section 2.3 to explain the algorithm.

2.1. Preliminaries

Our algorithm is designed based on the DOM tree derived from a web document in HTML or XML. The tree structure can be treated as an ordered tree $T = \{V, E, R\}$, where V designates a finite set of nodes, E designates the edges between the nodes, where $E \subseteq V^2$, and R designates the root node of T , where $R \bullet V$.

We distinguish two kinds of nodes: *Simple Nodes* and *Complex Nodes*. A Simple Node (S_node) is a leaf node in T that may be a text node (denoted as Text_Node) or a single tag node (denoted as Single_Tag_Node). A Complex Node (C_node) is an interior node in T . For example, in Figure 2, the nodes "Fender", "USA", etc. are text nodes. {"
"} is single tag node. The sub_tree rooted at "<TR>" is a C_node . Also, each sub_tree rooted at a "<TD>" is a C_node . For each node $v \bullet V$, we denote $paT(v)$ as the

parent node of v and $chT(v)$ as the set of children of v . For example, in Figure 2, $paT(<TD>)$ is $<TR>$.

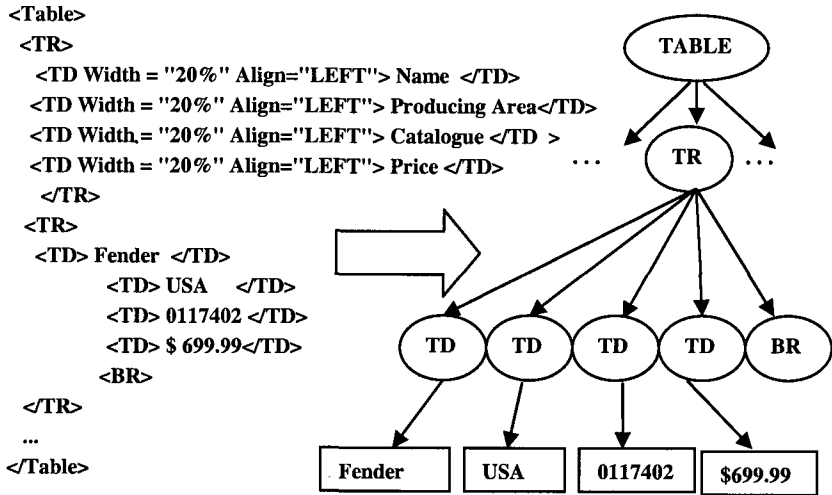


Fig. 2. An HTML document and its tree structure

Definition 1. A *Sub_Tree Pattern* is a kind of C_node in T , which occurs repeatedly. A *Sub_Tree Pattern Instance* is an occurrence of one *Sub_Tree Pattern*.

For example, a pattern “ $<TD> \rightarrow Text_Node$ ” in Figure 2 is a *Sub_Tree Pattern*, while “ $<TD> \rightarrow Fender$ ” is a *Sub_Tree Pattern Instance*. We can easily see that each cell of a table must be a *Sub_Tree Pattern Instance*.

Definition 2. Consecutive Tree Patterns (Consecutive Patterns for short) are a set of *Sub_Tree Pattern Instances* with the same *Sub_Tree Pattern* occurred in T consecutively.

In Figure 2, {“ $<TD> \rightarrow Fender$ ”, “ $<TD> \rightarrow USA$ ”...} is a set of *Consecutive Patterns*. We can easily see that each row of a table is a set of *Consecutive Patterns*.

Definition 3. $Size(n)$ denotes the number of tags and S_nodes in n , where n could be a S_node or a C_node . So $Size(n) = 1$ if n is a S_node , $Size(n) > 1$ if n is a C_node .

In Figure 2, the *size* of the complex node rooted at “ $<TR>$ ” is 10. We can see that the *Sub_Tree Pattern Instances* with the same *Sub_Tree Pattern* have the same *size*.

Definition 4. $Distance(n)$ denotes the distance from the root node R to the start tag of n , where n could be a S_node or a C_node . If the web document is treated as a text stream, $Distance(n)$ is equal to the number of tags and S_nodes in the stream before the occurrence of the start tag of n .

For example, in Figure 2, if “ $<Table>$ ” is the root of the tree, then $Distance(S_node(“Name”)) = 3$, $Distance(S_node(“Fender”)) = 13$. In our table detection algo-

rithm, we will calculate the *Distance* for every node in T and all these can be done in the HTML parsing phase using our extended JDOM parser.

Definition 5. Two Consecutive Patterns X and Y are called Maximum Consecutive Tree Patterns (Maximum Consecutive Patterns in short) if there are no other Consecutive Patterns X' and Y' with $\text{Size}(X') > \text{Size}(X)$.

Definition 6. $\text{Depth}(n)$ indicates the depth of the start tag of n in T , where n is a S_node or C_node .

For example, $\text{Depth}(R) = 0$ while $\text{Depth}(R_ch) = 1$, where R_ch is any child node of the root node R . So $\text{Depth}(n_ch) = \text{Depth}(n) + 1$, where n_ch is any child node of n .

Lemma 1. For any two nodes X and Y which are *Sub_Tree Pattern_Instances* with the same *Sub_Tree Pattern*, if $\text{Distance}(Y) - \text{Distance}(X)$ is equal to $\text{Size}(X)$, then X and Y are *Consecutive Patterns*.

This lemma is based on the continuity of HTML texts and can be easily proofed.

2.2. The table detection algorithm

Figure 3 shows our table detection algorithm. In Ref. 9 the authors classified two kinds of tables, genuine tables and non-genuine tables. Genuine tables are document entities where a two dimensional grid is semantically significant in conveying the logical relations among the cells. On the other hand non-genuine tables are document entities where the `<TABLE>` tags are used as a mechanism for grouping contents into clusters only for the ease of viewing. We can clearly see that only the genuine tables need to be annotated. We will try to extract only the genuine tables in this step.

We can detect a genuine table structure by finding the *Maximum Consecutive Patterns* between a "`<Table>`" and "`</Table>`" pair where the *Size* of each pattern is above a certain threshold σ

In a non-genuine table, *Maximum Consecutive Patterns* certainly exist. But these patterns contain fewer structures and less information than those in a genuine table. Thus the size of such a pattern is always less than σ . Using a threshold $\sigma = 3$ or 4 will prevent the extraction of most non-genuine tables in real world documents. However, some non-genuine tables may still be extracted. We will drop those meaningless non-genuine tables in the table analysis step. Sometimes a web document may miss a "`</Table>`" tag or other end tags of a node; they will be added during the parsing phase with our extended JDOM parser.

In the algorithm we use an extended JDOM parser to parse an input web document. For every node three attributes are considered: $\text{node.depth}(=\text{Depth}(\text{node}))$, $\text{node.size}(=\text{Size}(\text{node}))$, and $\text{node.dist}(=\text{Distance}(\text{node}))$. In line 2, we find the sub_tree T with the maximum size among all sub_trees (C_nodes) rooted at the tag "`<Table>`" of the Dom-Tree. Usually many sub_trees rooted at the "`<Table>`" tag can be found in a DomTree, but the rows of the table we want to find are always hidden in the sub_tree rooted at the "`<Table>`" tag with the maximum size.

Algorithm Table Detection**Input** : a web document D **Output**: *Max Consecutive Patterns* on the DomTree corresponding to the rows of the table we hope to find

Method:

```

1  Parsing on  $D$  using an extended JDOM and get its DomTree // Parsing Phase
2  Finding sub_tree  $T$  with maximum Size in all sub_trees rooted at the "<Table>"
   tag of the DomTree. // Initial Phase
3   $F_1$  = all  $S\_nodes$  in  $T$ , listed in their occurrence order
4  int deepest = maximum Depth for all the node in  $F_1$ 
5  int limit =  $T.depth$ 
6  int  $i$  = 2
7  while (deepest > limit) do begin
8    foreach node  $N \in F_{i-1}$  do begin
9      if ( $N.depth == deepest$ )
10        $F_i.add(Up\_Extension(N))$ 
11     else
12        $F_i.add(N)$ 
13     NodeMerge( $F_i$ )
14     for (int  $j=0; j < F_i.size - 1; j++$ ) do begin
15        $N_j = F_i.get(j)$  //  $N_j$  is the  $j$ th element in  $F_i$ 
16        $N_{j+1} = F_i.get(j+1)$  //  $N_{j+1}$  is the  $j+1$ th element in  $F_i$ 
17       if ( $N_j.pHash == N_{j+1}.pHash$ ) //  $N_j$  and  $N_{j+1}$  are of the same pattern
18         {
19           if ( $(N_{j+1}.dist - N_j.dist) == N_j.size$ ) // Consecutive Patterns
20             {
21                $F_i.conseqPattern.add(N_j)$ ;
22                $F_i.conseqPattern.add(N_{j+1})$ ;
23             }
24         }
25       deepest--
26        $i++$ 
27     end //end loop while
28 List maxCP = maxCPFinding( $i-1$ )
29 return maxCP

```

Fig. 3. Table detection algorithm

All the leaf nodes (S_nodes) are stored in F_1 with their $pHash$ values. Reference 11 proposed a hash function to calculate the hash value for a node. Here the same function is used to calculate $pHash$ for every node, but all Text_nodes are assigned the same $pHash$ value. Therefore if two nodes have the same pattern, they will have the same $pHash$ value. In the initial phase, we define some integer value based on the depth of the node, which will be used to terminate the loop later. After that, we adopt a *Consecutive Patterns* mining method by exploiting T from the leaf nodes to the root node. In lines 8-12, we

use an *Up_Extension* technique to extend the nodes in F_i . Only those nodes with the maximum depth need to be extended, and the others will be extended later. In line 13 we use the function $NodeMerge(F_i)$ as shown in Figure 4 to merge the nodes with the same root. In lines 14-24, we find all the *Consecutive Patterns* in F_i based on Lemma 1, and all these *Consecutive Patterns* are stored in $F_i.conseqPattern$. In line 28, $maxCPFinding(i-1)$ finds the *Max Consecutive Patterns* in $\{F_1.conseqPattern, F_2.conseqPattern, \dots, F_{i-1}.conseqPattern\}$.

The *Up_Extension* technique is used to extend node n with its $paT(n)$ to get a new node as shown in Figure 5 and to calculate the $pHash$ value for the new node. The function $NodeMerge(F_i)$ is used to merge the nodes in F_i . The merged nodes should share the same start tag as shown in Figure 5.

Algorithm *NodeMerge*

Input : NodeList F

Output : Merged Nodes in List F

Method:

```

1  int k=0;
2  while (F.hasNext( )) do begin
3       $N_k = F.get(k)$  //  $N_k$  is the kth element in  $F$ 
4       $N_{k+1} = F.get(k+1)$  //  $N_{k+1}$  is the (k+1)th element in  $F$ 
5      if ( $N_k.rootNode == N_{k+1}.rootNode$ )
6          merge  $N_k$  and  $N_{k+1}$  to get new node  $N_{new}$  and calculate
            $pHash$  value for  $N_{new}$ 
7           $F.remove(N_k)$  // remove  $N_k$  in  $F$ 
8           $F.replace(N_{k+1}, N_{new})$  // replace  $N_{k+1}$  with  $N_{new}$  in  $F$ 
9           $k++$ 
10 end // end while
11 return  $F$ 

```

Fig. 4. *NodeMerge* function

2.3. An example

Figure 5 gives an example to illustrate our algorithm. We transform a real world HTML page into a DomTree rooted at node 101. The tag for node A is “<Table>”, the tag for node B is “<TR>”, the tag for node C is “<TD>”, and the dark nodes are leaf nodes with a text value. We can see that F_1 is the *Up_Extension* of F_0 , and F_2 is the *Up_Extension* of F_1 . $\{(301, 401), (302, 402)\}$ are *Consecutive Patterns*, so are $\{(303, 403), (304, 404)\}$, $\{(201, 301, 401), (201, 302, 402)\}$ and $\{(201, 301, 302, 401, 402), (201, 303, 304, 403, 404), \text{ and } (201, 305, 306, 405, 406)\}$. The *Max Consecutive Patterns* are $\{(201, 301, 302, 401, 402), (201, 303, 304, 403, 404), (201, 305, 306, 405, 406)\}$, where each stands for an instance of the row in the table.

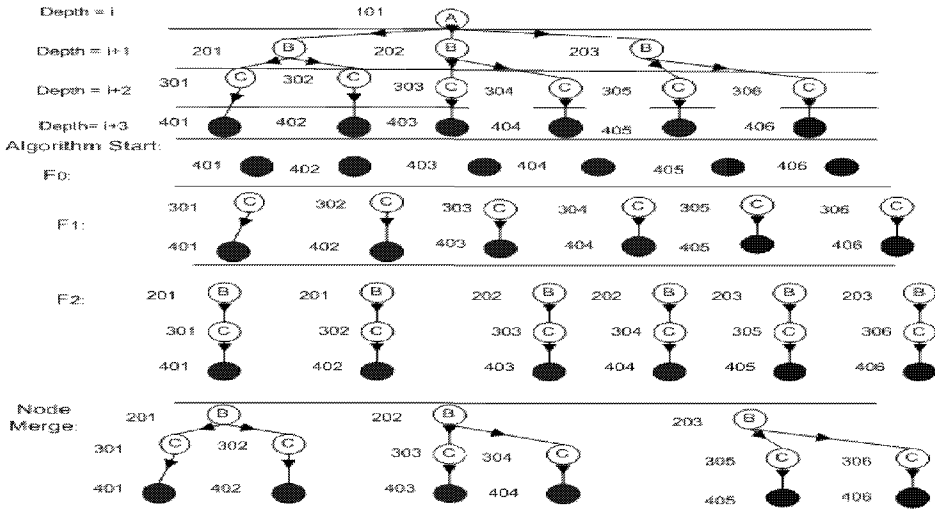


Fig. 5. Table detection process.

3. Table Analysis

3.1. Overview

In this section, we will analyze the table instances found in the table detection step, and explore the semantic meanings behind them with a given ontology. After the table detection step, the table structure in the web document is converted into a relational data set called *Table_Matrix* that shares the same rows and columns appeared in the web browser. However, some noisy data may exist (e.g., non-genuine tables) and most of these noisy data will be dropped in this phase.

The ontology describing the site is assumed to be available in our annotation system. For example, the table shown in the left hand side of Figure 1 describes “movies”, while the table shown in the right hand side describes “flights”. A real-world ontology called *Onto* contains a set of concepts or properties to describe movies. As usually a small subset of the concepts are used to annotate the corresponding values in a table, our task therefore is to identify the related concepts in *Onto* for every column of the table; subsequently these related concepts are used to annotate the cells of the column.

3.2. Strategies for table analysis

First we will separate two kinds of genuine tables (self-described and non-self-described) appeared in a web document. We will then use different strategies to associate some semantic meanings with them.

- **Self-described tables:** Every cell in a self-described table has specific semantic meanings with a text string, like the table shown in the left hand side of Figure 1.

- **Non-self-described tables:** Every cell in a non-self-described table has no specific semantic meaning but only a value, like the table shown in the right hand side of Figure 1.

For a self-described table, every cell in *Table-Matrix* has some semantic meaning by itself, that is to say, there exist both a *Semantic Indicator* and a value in the cell. For example, in the self-described table about “Movie”, the following value may exist in a cell: “[name] - Matrix : ReLoad”, where “[name]” is a *Semantic Indicator* indicating that the cell is used to describe the name of a film, “Matrix : ReLoad” is the value of the name, and “-” is the separator between the *Semantic Indicator* and its value.

Algorithm *Semantic_Finding* for self-described table

Input: *Onto* and *Table-Matrix* of a self-described table

Output: *Concept_Column_Set* in which Concepts in *Onto* that describe the semantic meanings for each column in *Table-Matrix*

Method:

```

1 Concept_Column_Set = null;
2 Foreach column Col in Table-Matrix do Begin
3     Semantic Indicator = maximum prefix for all cells in Col
4     Delete all the nonsense characters in Semantic Indicator and only leave meaningful words.
5     Foreach Concept C • Onto do Begin
6         if (Semantic Indicator has the same meaning with C in WorldNet)
7             // C can describe the semantic meaning of Col
8             Concept_Column_Set.add(pair(Col, C));
9             break;
10        End_if
11    End
12 return Concept_Column_Set
    
```

Fig. 6. *Semantic_Finding* for self-described table.

However, we cannot depend on the separator sometimes because they do not always exist. Fortunately in a self-described table, all the cells in a column always have the same *Semantic Indicator*. Based on this observation we propose the algorithm *Semantic_Finding* as shown in Figure 6. In the algorithm, we find the maximum prefix for all the cells in a column. The prefix would contain the *Semantic Indicator* for that column. After the *Semantic Indicator* is extracted, we decide whether some concept in *Onto* is in a synonym chain of the *Semantic Indicator* in the WordNet. If so, this concept can be used to describe the semantic meaning of the column, and can be used to annotate the column.

For a non-self-described table, the task is more difficult. We have developed three strategies for this task: (1) Exploiting table head information; (2) Exploiting the linguistic pattern of a cell and associating it with a concept; and (3) Exploiting the link page for the value in each cell.

(1) *Exploiting table head information*

If a table has a table head, then the semantic meanings for each column are hidden in the table head. In this situation, we need to find every value in the table head that can be used to describe the semantic meaning of a column such that some appropriate ontology concepts can be used to associate with the column. Our case study shows that table heads are always presented in either of the following two forms:

- (i) The table head appearing on the top of the row instances has the same pattern with those of the row instances. In this case, the table head is extracted together with the row instances in *Table_Matrix*.
- (ii) The table head appearing on the top of the row instances has a different but similar pattern with those of the row instances. If there are n columns in the row instances, there might be a node *HeadNode* containing n children appearing right before the row instances and the *pHash* value of *HeadNode* is very close to those of the row instances. In this case, we extract the table head from *HeadNode*.

After the table head is found, its value includes the *Semantic Indicators* for the related columns. So we can use lines 4-9 of the algorithm shown in Figure 6 to find the correct ontology concepts.

(2) *Exploiting the linguistic pattern of each cell*

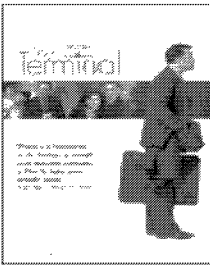
This strategy is popular in some wrapper systems (e.g., Refs. 12 and 13). The basic idea lies in that we can learn the semantic concept for some typical values in each cell. For example, consider “10018 Fourth Ave Brooklyn” and “120 Orchard Street New York”. These values share a common linguistic pattern “NUMBER + STREET + CITY” which actually corresponds to the concept of “Address”. Thus we can associate them with the concept “Address”. As another example, the values “\$17.95” and “\$25.99” can be associated with the concept “Price”.

(3) *Exploiting the link page for the value in each cell*


Link analysis is heavily used in web information retrieval. The link(s) in a row is often used as a pointer to a page that contains the details of a cell. In most cases, all values in the same row of a table are repeated in the corresponding detail page. We can use strategy (2) to find the corresponding concepts using the linguistic patterns identified in the detail page. These concepts designate the semantic meanings of the row.

3.3. *An example*

In real world web pages, some part of a table may be self-described and other part may be non-self-described. As an example, Figure 7 is such a sample table describing movie from www.blockbuster.com. So in our experiment, both kinds of table analysis strategies are integrated. This section will illustrate the whole table analysis process. Figure 8 is a part of movie ontology *Movie_Onto* that we built consisting of a concept and its related properties. In this ontology, linguistic patterns for some properties have been built. For example, “Name” has linguistic pattern “Name_String [String] (Year)”, “Release Date”



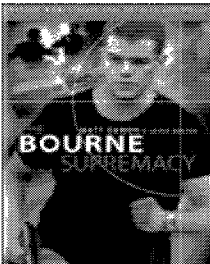
Terminal [WS](2004)
Enhanced Widescreen Letterbox DVD
[All Editions](#)

Member Rating:  **Rated: PG13**
Release Date: 11/23/2004


[Add to Queue](#)

Shot almost entirely on a two-and-a-half-story recreation of a full-size operating airport terminal, this romantic comedy from director Steven Spielberg revolves around an Eastern European man by the... [More](#)

Starring: [Tom Hanks](#), [Catherine Zeta-Jones](#), [Catherine Zeta-Jones](#), [More](#)



Bourne Supremacy [WS](2004)
Enhanced Widescreen Letterbox DVD
[All Editions](#)

Member Rating:  **Rated: PG13**
Release Date: 12/07/2004

[Add to Queue](#)

The second chapter in the "Bourne Trilogy," based on Robert Ludlum's best-selling espionage novels, reaches the screen in this sequel to the 2002 thriller The Bourne Identity. Jason Bourne (Matt... [More](#)

Starring: [Matt Damon](#), [Franka Potente](#), [Franka Potente](#), [More](#)

Fig. 7. Sample Table describing movie.

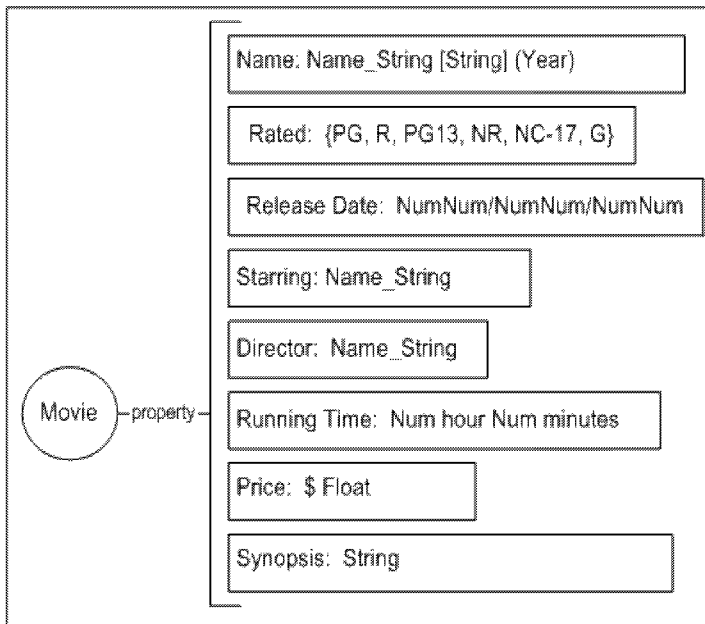


Fig. 8. Concept and related properties describing movie.

has linguistic pattern “Num Num/ Num Num/ Num Num”, and “Running Time” has linguistic pattern “Nun hour Nun minutes”. Table 1 shows the *Table_Matrix* constructed by the table detection phase. From Table 1, we can see 9 columns are detected and each of them is assigned a virtual name “A, B, C ...” Note, here we record the link page of each column (if available) for further use. Table analysis will identify the related concepts in *Movie_Onto* (see Figure 7) for each column of the *Table_Matrix*.

Table 1. *Table_Matrix*

A [link]	B [link]	C	D [link]	E [link]	F [link]	G	H [link]	I [link]
IMG	Terminal [WS](2004)	Enhanced Widescreen Letterbox DVD	All Editions	IMG	Rated: PG13	Release Date: 11/23/2004	Text	Starring: Tom Hanks
IMG	Bourne Supremacy [WS](2004)	Enhanced Widescreen Letterbox DVD	All Editions	IMG	Rated: PG13	Release Date: 12/07/2004	Text	Starring: Matt Damon
IMG	Collateral [WS] (2004)	Enhanced Widescreen Letterbox DVD	All Editions	IMG	Rated: R	Release Date: 12/14/2004	Text	Starring: Tom Cruise
IMG	I,Robot [WS](2004)	Letterbox DVD	All Editions	IMG	Rated: PG13	Release Date: 12/14/2004	Text	Starring: Will Smith
IMG	Stepford Wives [WS](2004)	Enhanced Widescreen Letterbox DVD	All Editions	IMG	Rated: PG13	Release Date: 11/09/2004	Text	Starring: Nicole Kidman
IMG	Manchurian Candidate [WS](2004)	Enhanced Widescreen Letterbox DVD	All Editions	IMG	Rated: R	Release Date: 12/21/2004	Text	Starring: Denzel Washington

Firstly, using algorithm *Semantic_Finding* on each column, we can find that column B has common suffix “[WS] (2004)” and column D contains the same value on each rows, so as column C. But from this information, we can find no relationship with the *Movie_Onto*. However, we can find *Semantic Indicator* “Rated” on column F, and “Rated” is a property of *Movie_Onto*. So we successfully identify an ontology property for column F, so as column G and I. Note, column A and E only contains images, so we ignore them in this step.

Secondly, exploring the linguistic pattern information on column B, we find that each value in it could match with the linguistic pattern in the ontology property “Name”. So we successfully identify an ontology property for column B.

Thirdly, building wrappers for the link pages of the values in column A, B, D, E, F, H and I, we find the wrapper building process succeeded in column A, B, H and I, but failed in column D and E. From the wrappers, we find more property instances of *Movie_Onto* like “Director”, “Running Time”, “Price” and “Synopsis”. Here we can also find that the instances of “Synopsis” have nearly the same values with those in column I. So column I can be used to describe the ontology property “Synopsis”.

So, we have correctly identified the 5 ontology properties for column B, F, G, H and I, and these properties can be used to add semantic annotation for these cells. As a sup-

plemental result, we also find the property instances of “Director”, “Running Time” and “Price”. They can also be used for semantic annotation.

4. Experiments

To evaluate the effectiveness of our approach, we designed experiments based on a set of real-life web sources including 11 sites. These sites describe 7 concepts {faculty, airport, movie, book, restaurant, stocks, hotel}. We used SWOOGLE to get the corresponding ontologies and made some improvements manually to each concept in order to make them usable for our experiments. As an example, Figure 9 shows the faculty ontology.

In each of these 11 sites, we selected 5 pages containing genuine tables, and some pages containing non-genuine tables. The table detection algorithm successfully extracted all the table structures from all the HTML pages in the data set as shown in Table 2.

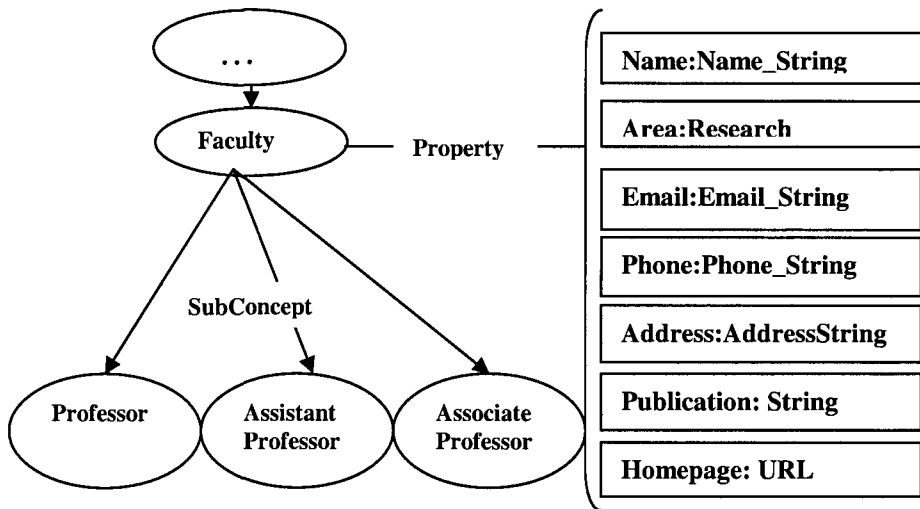


Fig. 9. Faculty ontology

Table 2. Results of table detection.

source	No. of genuine tables given	No. of non-genuine tables given	No. of extracted genuine tables + No. of non-genuine tables	accuracy
www.uci.edu	5	3	5 + 0	100%
www.nationsonline.org	5	3	5 + 1	87.5%
www.blockbuster.com	5	2	5 + 0	100%
www.bordersstores.com	5	5	5 + 2	80%
www.restaurantrow.com	5	4	5 + 1	88.8%
finance.yahoo.com	5	3	5 + 3	62.5%
www.hotel.com	5	5	5 + 0	100%
canberra.citysearch.com.au	5	4	5 + 1	88.8%
boston.citysearch.com	5	4	5 + 3	66.7%
boston.citysearch.com	5	2	5 + 0	100%
www.amazon.com	5	5	5 + 5	50%

Table 2 also shows the number of non-genuine tables extracted. The accuracy of the algorithm was calculated by the following formula:

$$\text{Accuracy} = \frac{GT_{ext} + nonGT_{ori} - nonGT_{ext}}{GT_{ori} + nonGT_{ori}} \quad (1)$$

In the formula, GT_{ori} and $nonGT_{ext}$ denote respectively the total number of genuine tables and non-genuine tables given originally, and GT_{ori} and $nonGT_{ext}$ designate the number of genuine tables and non-genuine tables that have been extracted by our table detection algorithm.

Note from the results of the table detection algorithm, we found most pages of a site containing genuine tables share the same structure because they were produced by the same template. Therefore we only need to choose one page from each site for semantic analysis. In Table 3 we list the number of columns appeared in the tables, the number of columns that should be annotated and the number of columns that were successfully annotated.

Table 3. Table analysis results for semantic annotation.

Source	Domain	Number of columns	Number of columns needed to be annotate	Number of the columns annotated	accuracy
www.uci.edu	Faculty	8	4	3	75%
www.nationonline.org	Airport	3	3	2	66.7%
www.blockbuster.com	movie	7	5	4	80%
www.bordersstores.com	book	4	4	4	100%
www.restaurantrow.com	restaurant	2	2	2	100%
finance.yahoo.com	stock	7	7	5	71.4%
www.hotel.com	hotel	4	2	2	100%
canberra.citysearch.com.au	restaurant	7	4	4	100%
boston.citysearch.com	hotel	4	3	2	66.7%
boston.citysearch.com	restaurant	5	3	3	100%
www.amazon.com	book	8	5	4	80%

From the results we can see that both algorithms are of quite high accuracy. The inaccuracy of table analysis was mainly resulted from certain specific features of a column. For example, at “www.amazon.com”, an important column “Customer Rating” is reported with an image. As another example, at “boston.citysearch.com”, a non-self-described table with a “brief introduction” column could not be correctly analyzed because the concept corresponding to the column is not contained in the ontology.

5. Conclusions

This paper presents an approach to automatically extracting instances from the tables in existing web documents. The core of our approach consists of two processes: table detection and table analysis. Based on the regularity of table structures, we have proposed an efficient table detection algorithm that uses an extended JDOM parser to parse a web

page and explore the DomTree in a bottom up fashion. We use *Consecutive Tree Patterns* to record the structure regularity of tables, and extract the instances of a table by the *Max Consecutive Patterns*. In table analysis, we associate the instances of a table with their related concepts in a given ontology. In this step, we distinguish two kinds of tables: self-described tables and non-self-described tables. For a self-described table, we extract its *Semantic Indicators* by finding the maximum prefix from all the values in the same column. We then find the appropriate ontology concepts using the synonym chains in the WordNet from the *Semantic Indicators*. For a non-self-described table, we consider information that may be derived from the head of a table, the linguistic pattern of a cell and the link page from a cell. A different strategy is designed for each situation and integrated into our system.

The performance of the proposed algorithms has been verified by a set of experiments based on real world web documents and the results are encouraging.

We are currently improving the table analysis by considering context information of the web document and building intelligent system on deep web by integrating our semantic annotation technique with the interface query technique discussed in Ref. 16.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No.60573077), Program for New Century Excellent Talents in University, Microsoft Research Asia and 973 Program of China (No. 2003CB317002).

References

1. J. Heflin and J. Hendler. Searching the web with SHOE. In: *Proceedings of AAAI-2000 Workshop on AI for web Search*, 2000, pp. 35–40.
2. S. Handschuh, S. Staab, and A. Maedche. CREAM – Creating Relational Metadata with a Component-based, Ontology-driven Annotation Framework. In: *Proceedings of First International Conference on Knowledge Capture*, 2001, pp. 76–83.
3. S. Handschuh, S. Staab, F. Ciravegna. S-CREAM: Semi-automatic CREATION of Metadata. In: *Proceedings of 13th International Conference on Knowledge Engineering and Knowledge Management*, 2002, pp. 358–372.
4. J. Kahan and M.-R. Koivunen. Annotate: an open RDF infrastructure for shared web annotations. In: *Proceedings of WWW 2001*, pp. 623–632.
5. S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, et al. SemTag and Seeker: Bootstrapping the Semantic web via Automated Semantic Annotation. In: *Proceedings of WWW 2003*, pp. 178–186.
6. A. Dingli, F. Ciravegna, and Y. Wilks. Automatic Semantic Annotation using Unsupervised Information Extraction and Integration. In: *Proceedings of the K-CAP 2003 Workshop on Knowledge Markup and Semantic Annotation*, 2003.
7. S. Wang, E. Chen. An Instance Learning Approach for Automatic Semantic Annotation. In: *International Conference on Computational and Information Sciences*, Lecture Notes in Computer Science 3314, 2004, pp. 962–968.
8. S. Handschuh, S. Staab, and R. Volz. On Deep Annotation. In: *Proceedings of WWW 2003*, pp. 431–438.
9. Y. Wang, J. Hu. A Machine Learning Based Approach for Table Detection on the web. In: *Proceedings of WWW 2002*, pp. 242–250.

10. K. Lerman, C. Knoblock, and S. Minton. Automatic Data Extraction from Lists and Tables in web Sources. In: *Proceedings of the workshop on Advances in Text Extraction and Mining*, IJCAI-2001.
11. Y. Wang, D. J. DeWitt, and J. Cai, X-Diff: A Fast Change Detection Algorithm for XML Documents. In: *Proceedings of ICDE 2003*.
12. W. Cohen and L. Jensen. A Structured Wrapper Induction System for Extracting Information from Semi-structured Documents. In: *Proceedings of the Workshop on Adaptive Text Extraction and Mining*, IJCAI 2001.
13. I. Muslea and S. Minton and C. Knoblock: Active Learning with Strong and Weak Views: A Case Study on Wrapper Induction. In: *Proceedings of 18th International Joint Conference on Artificial Intelligence*, IJCAI 2003.
14. G. Miller. WordNet: A lexical database for english. *CACM*, 38(11), 1995, pp. 39–41.
15. B. Liu, R. Grossman, Y. Zhai. Mining Data Records in Web Pages. In: *Proceedings of ACM SIGKDD 2003*, pp. 601–606.
16. Z. Zhang, B. He, and K. C.-C. Chang. Understanding Web Query Interfaces: Best Effort Parsing with Hidden Syntax. In: *Proceedings of ACM SIGMOD 2004*.