Finding Structure and Characteristics of Web Documents for Classification

Wai-ching Wong and Ada Wai-chee Fu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
{wcwong, adafu}@cse.cuhk.edu.hk

Abstract

Many Web documents containing the same type of information, would have similar structure. In this paper, we examine the problem of finding the structure of web documents and present a *hierarchical structure* to represent the relation among text data in the web documents. Due to the loose standard of web page publishing, different authors can use different wordings (labels) to label the same information. We introduced a *labels discovery algorithm* that uses the hierarchical structure extracted from the web pages. The algorithm discovers similar labels which describe the same kind of information. Such labels would help us find the structure of the web documents. Experiments have shown that the algorithm can successfully discover similar labels and the structure obtained by our method can distinguish web pages accurately.

1 Introduction

The World Wide Web (WWW) provides a vast resource for information of almost all types. There are over half a billion homepages on the WWW and a thousand more are appearing each hour. Users commonly retrieve this data by browsing and keyword searching. However, browsing is not suitable for locating particular items of data because following links is tedious, and it is easy to get lost. Furthermore, browsing is not cost-effective as users have to read the documents to find the desired data. Keyword searching is sometimes more efficient than browsing but often returns vast amounts of data, much more than the user can handle. Some researchers have resorted to ideas taken from databases techniques in order to retrieve data more efficiently from the Web. Databases, however, contain structured data and most Web data is semistructured in nature and cannot be retrieved easily by using traditional techniques. Therefore, quite a lot of recent work has been targeted to handle this problem [2, 7, 11, 13, 16].

Semistructured data [1, 5] is characterized by the lack of any fixed and rigid schema, although, unlike unstructured raw data, typically the data has some implicit structure. Querying such data is greatly difficult. Due to this irregularity, each web page contains its own schema or structure. However, for web pages that describe the same type of information, it is common that they have similar ***structure***. We call a set of web pages which stores the same type of information a ***class***. We shall introduce a definition for the typical structure of a class, and derive mechanisms to extract such structure from web pages. The structure can be used for identification of web pages that belong to the class.

The paper is organized as follows: Section 2 defines the problem. Section 3 describes some related work. Section 4 and Section 5 present the method for the extraction of hierarchical structure and the labels discovery algorithm. Section 6 describes the web document classification method. Section 7 shows the experimental results. Section 8 is a conclusion.

2 Problem Definition

We say that web pages that describe the same type of information belong to the same ***class***. For example, we may have a class of web pages that describe job openings. Web pages of the same class would have similar ***structure*** which describes the format of information in the pages. For users to find a particular data in a class of web pages more efficiently, the typical structure of the class should be retrieved. In this paper, the objective is to discover the typical structure of a class of web pages. The typical *structure* of a class of web pages describes the types of data that could be stored in the web pages.

Definition 1 *A* ***class*** *of web pages is a set of web pages that store the same type of information. Each class of web pages has a number of* ***attributes***.

Attributes for a class is similar to the attributes in a relational database tables. For example, in a class of job opportunities web pages, there are three attributes: *job name*, *job description*, and *job requirement*. For our considerations, a web page mainly consists of two categories of data, ***labels*** and ***values***.

Definition 2 *A **label** of a class C labels a data stored in a web page of class C, it corresponds to one of the attributes of the class C.*

For each label, there is a corresponding *value* in the page. For example, in a class of job opportunities web pages, we assume that there is an attribute called *job name*. In a segment of HTML codes:

```
<TD>Position Title</TD>
 <TD>IT Specialist</TD>,
```

Position Title is the label, which is used to label the attribute *job name*, and the value of the label or attribute is *IT Specialist*. A label can be used by multiple web pages, we say that each of such web pages contains an instance of the label.

Definition 3 *A **value** is the value of an instance of the corresponding **label** in a web page.*

There are two major differences between the attributes here and the attributes in a relational database: (1) In a relational database, the attributes may not uniquely identify a table. We assume the set of attributes can uniquely determine the class of a web page. I.e. different classes have different sets of attributes. (2) In a relational database table, an attribute has a unique reference name. However, the standards are quite loose for publishing web pages, so the label that is used for the same data attribute can be different in different pages. That is, more than one label may be found the same attribute. For example, for the attribute *job name*, a web page may use *job title*, *job position*, or *position title*, etc., as the label. So, *a typical structure* consists of a set of labels that would appear in the class for each attribute.

Definition 4 *The **structure** of a class consists of a set of **labels** for each attribute for the class.*

We shall make use of the properties of HTML tags, which provide display format of the data stored in the pages, to extract the structure. Based on the properties of HTML tags, they can be divided into three groups: ***rigid structure tags***, ***loose structure tags*** and ***non-structure tags***. Firstly, we construct a ***tag-tree*** for each web page by referring to the first two groups of tags. A heuristic approach is introduced to extract the ***hierarchical structure*** from the web pages with the aid of the tag-tree. The hierarchical structure could then be used to find the typical structure of the web pages.

In our collection of job opportunities pages, almost all pages store only one job description instead of a list of jobs. Therefore, in this paper, we assume that each web page contains a single instance of a class. In the future, we can extend our works to pages with multiple instances. The multiple instances could be extracted by discovering the boundaries between instances in the web documents [10].

Assumption 1 *In a web page, only a single instance of a class is stored.*

Observation 1 *Labels for two different attributes of a class are always different within a single web page instance.*

3 Related Works

The TSIMMIS project [7] uses a configurable tool for extracting semistructured data from a set of HTML pages and for converting the extracted information into database objects [12]. The output data is in the form of the OEM (Object Exchange Model) [7]. A problem of this approach is that the extraction mechanism depends on user input for describing the structure of HTML pages. Also, this approach can only extract information from a set of web pages with exactly the same structure.

Another research work [14] extracts schema from semistructured data. It discovers the structure implicit in semistructured data and, subsequently, the recasting of the raw data in terms of this structure. It considers a general form of semistructured data based on labeled, directed graphs. It proposes an algorithm for approximating typing of semistructured data. They establish that the general problem of finding an optimal such typing is NP-hard, but present some heuristics and techniques based on clustering that allow efficient and near-optimal treatment of the problem. However, the typing of the schema is the same in the data set. For example, each record in the data set will use 'Birthday' to represent the same attribute. We assume here that different words can be used to represent the same attribute. Other works that also use a labeled directed graph for the modeling can be found in [15, 6].

Another related work that discovers the typical structures of documents is [17]. The structure of a document is determined by the role and hierarchy of subdocument references. They propose that a document contains subdocument references, which can be hierarchical, labeled, ordered, and cyclic. They adopt OEM for representing semistructured documents. An algorithm similar to that for mining association rules [3, 4] is used to find the typical structure

```
<HTML>
<HEAD>
<TITLE>IBM Hong Kong - Employment</TITLE>
</HEAD>
<BODY>
<TABLE>
<TR>
  <TD>Position Title</TD>
  <TD>Engagement Manager, Cross Industry Offerings</TD>
</TR>
<TR>
   <TD>Ref. no:</TD>
   <TD>Shanghai 1</TD>
</TR>
</TABLE>
<P>Work Location</P>
<UL>
    <LI>IBM China Company Limited, Shanghai Branch, Shanghai</LI>
</UL>
<P>Responsibilities</P>
<UL>
    <LI>Qualify service opportunities</LI>
    <LI>Conduct business requirement analysis for customers</LI>
      :
</UL>
<P>Requirements</P>
<UL>
     <LI>5 to 6 years engagement related work experience</LI>
      :
</UL>
  :
</BODY>
</HTML>
```

Figure 1: Example 1: a section of the HTML source

of a collection of structures. However, the typical structure found in the above refers to the hierarchy of subdocuments and they do not make use of the structure to distinguish web pages. In our approach, we find the labels that describe the data stored in the web pages in the class. Also, the typical structure obtained would be used to classify web pages.

4 Hierarchical Structure

Given a set of training web pages for a particular class, we want to identify the characteristics of the class. Then given any web page, we can determine if it belongs to the class by measuring its similarity to the characteristics.

Web pages are constructed by the HTML tags which provide the format of displaying data stored in the web page. By using the properties of the tags and the relations among the tags, we may be able to find the structure of web pages for a given class. According to the properties of tags, a ***tag-tree*** is constructed, which could be used to construct the hierarchical structure efficiently.

Definition 5 *A* **text data** *is the text enclosed by a pair of HTML tags in the web pages.*

For example, in the following HTML segment, Ability to work with senior management 60% of working time is in China, 'Ability to work with senior management' and '60% of working time is in China' are two text data. The ***hierarchical structure*** of a web page describes the relations among text data in a web page.

4.1 Tag-tree Construction

HTML document has some linguistic conventions in its page layout. These conventions are determined by the HTML tags. HTML tags divide HTML document into regions. The start-tag and the corresponding end-tag define a discrete region. Also, there are nested tags within the start-tag and end-tag to further divide the region into sub-regions. By using this nested property of HTML tags, a ***tag-tree*** [9] can be constructed. Two *rules* are used to construct the tag-tree:

1. If the tag t_1 is directly nested in another tag t_2, then t_1 is the **parent** of t_2.

2. If two tags, t_1 and t_2, enclosed by the same tag are not nested to each other and the position of t_1 in the web page is above t_2, then t_1 is the **left sibling** of t_2.

The tag-tree is useful in the extraction of data from web documents [9]. From the *tag-tree*, the relation among discrete regions is represented by the tags in the internal node of the tree. In Figure 1, we show a segment of an HTML document for an example, Example 1. The snapshot of the corresponding web page is shown in Figure 2. The tag-tree of the document is shown in Figure 3.

Single tags, such as `IMG`, `HR`, are eliminated from the *tag-tree*, since such tags do not enclose any ***text data*** in the web page and some of them are used for decoration only.

As text data and single tags do not enclose other tags or text data, there are no children under them in the tree. In the *tag-tree* the leaf nodes are either single tags or text data. If a leaf node is a single tag, then it has to be removed. After removing these nodes, their parent node may become leaf node, and then it also has to be removed. This step is propagated upwards the tag-tree until there is no leaf node containing tag.

4.2 Types of Tags

We divide the HTML tags into three groups: ***rigid structure tags***, ***loose structure tags*** and ***non-structure tags***.

(1) ***Rigid Structure Tags.*** They provide structure of the enclosed text data. For example, the `table` tag provides a tabular structure of text data within it. The *list* tag, `ul`, provides a hierarchical structure of text data within it.

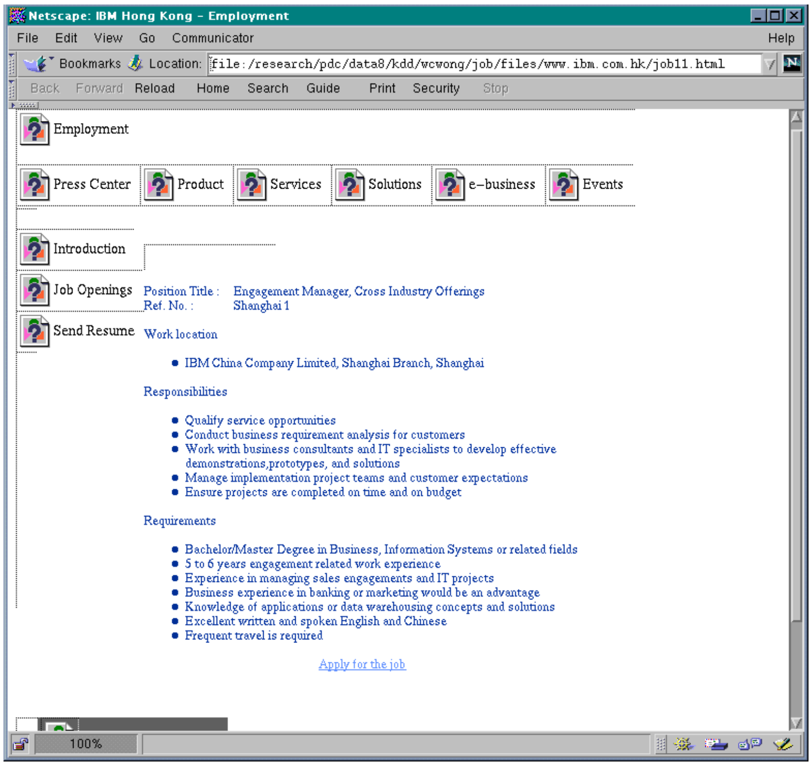

Figure 2: The web page for Example 1

(2) ***Loose Structure Tags.*** They do not provide any structure to the data they enclose but they give structural information among the tags. For example, the heading tags, `H1, H2, ..., H6`, provide information of the structure among the tags but do not structure the data within them.

(3) ***Non-structure Tags.*** They do not provide any information to the structure of the web pages. For example, the image tag, `img`, and some decoration tags, e.g. `font`, `hr`, etc.

As we want to extract the structure of a web page, *non-structure tags* will be eliminated from the tag-tree. Rigid structure and loose structure tags are then used to construct the hierarchical structure.

4.3 Heuristics in Hierarchical Structure Extraction

We shall construct one hierarchical structure for each web page. A hierarchical structure is a tree, in which each **node** corresponds to a text data in the web page. The root of the tree is at **level** 0, we say that it is at the highest level. A child of a node at level i is at level $i+1$. We say that level i is higher than level $i+1$. The tree structure will represent some relationship among the text data in the web page.

To construct the hierarchical structure, we use the relationship among the tags. Each tag has some meaning to the page layout. They are used to display the text data in the format that is more convenient for the users to locate the information in the web page. Based on the properties of the tags and the relations among them, we propose five main heuristics to construct the hierarchical structure from a web page.

(1) ***Heading loose structure tags.*** In the loose structure tags, there are heading tags (`H1, H2, ..., H6`) that explicitly show the hierarchical relation. The number in the tags can be mapped to the levels in the hierarchy. In addition, `title` tag is used to indicate the title of the page, therefore the text data enclosed is at the highest hierarchical level. So, we assign the hierarchical level to each data enclosed by loose structure tags as follows: title > H1 > H2 > ... > H6.

(2) ***Non-heading loose structure tags.*** In an article, heading is usually used to describe its following paragraph. We observe that text data of a short length is likely the heading of its following text data with a longer length. Therefore, for the text data enclosed by loose structure tags other than heading tags, a short text data is set higher in the hierarchy than its following longer text data.

(3) ***Rigid structure tags and loose structure tags.*** The heading tags are often used to divide the

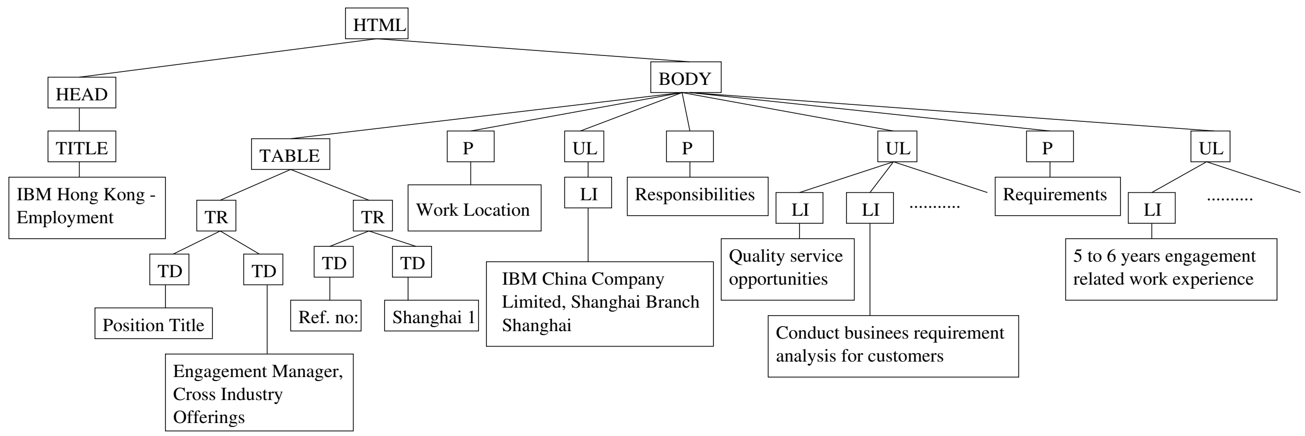

Figure 3: A tag-tree for Example 1

web pages into several regions and the text data enclosed by the heading tags are used to described the text data in the corresponding region. Therefore the text data extracted from some rigid structure tags has lower hierarchical level than the text enclosed by heading tags. Also, a short length text data above some rigid structure tags in the web page has a higher hierarchical level than the text data extracted from that rigid structure tags.

(4) ***Hierarchical structure tags.*** The rigid structure tags can be divided into two types: hierarchical structure and tabular structure. The hierarchical structure include ul, ol, dl, div, dir. These tags are used to group similar data together and to arrange the text data into different hierarchical level. For example, if there is a HTML segment "
item1
item2
", then the text data "item1" and "item2" are under the same parent and in the same level. These tags can also be nested. For example, if there is a HTML segment " item1 item3 item2 ", then the text data "item3" is the child of "item1".

(5) ***Tabular structure tags.*** The tabular structure is the structure extracted from the table tag. There are two types of table, one has headings and another does not. If the table has headings, then the headings are the labels of the corresponding column. The text data in each column are the values of the corresponding headings. Each row of the table is an individual record. For a table without headings, in the same row, text data that appear to the left of the table will be given higher hierarchical level than those to the right. This is because the leftmost entry will have a higher possibility to be the label. An example can be seen in Example 1, where "Position Title" and "Ref no:" appear at the left of the table and they are the headings.

There are other minor heuristics which we do not list for interest of space. Applying these heuristics for the tag-tree, the hierarchical structure can be constructed. Each node in the structure contains a text data in the web page. Figure 4 shows the resulting hierarchical structure for Example 1. From this structure, we can discover four possible neighbour nodes for each text data: *parent*, *children*, *left sibling* and *right sibling*. In the hierarchical structure of web pages, we have an observation that most labels have their corresponding values as their child or right sibling text data. Based on this observation, we introduce an algorithm to find all labels of a set of web pages.

5 Labels Discovery Algorithm (LDA)

The structure of a web page consists of *labels* by Definition 4. To find the typical structure of a class of web pages, we have to find the set of labels that represents that class. There are two main tasks: (1) Not all text data will be labels. The first job is to identify those text data that may be labels. (2) More than one labels may be used to describe the same attribute. We need to find the set of labels that represent the same attribute. In this section, we introduced the ***labels discovery algorithm (LDA)*** which can achieve these goals.

5.1 Expression of Hierarchical Structure

The hierarchical structure is a tree in which each node corresponds to some *text data* in a web page. In the structure, there are four possible **neighbour nodes** for a node: parent, left sibling, right sibling and children. In our algorithm, we use these relations to find the set of labels, so the hierarchical structure of each page is represented by a set of nodes $\{n_1, n_2, \ldots\}$. Each ***node***,

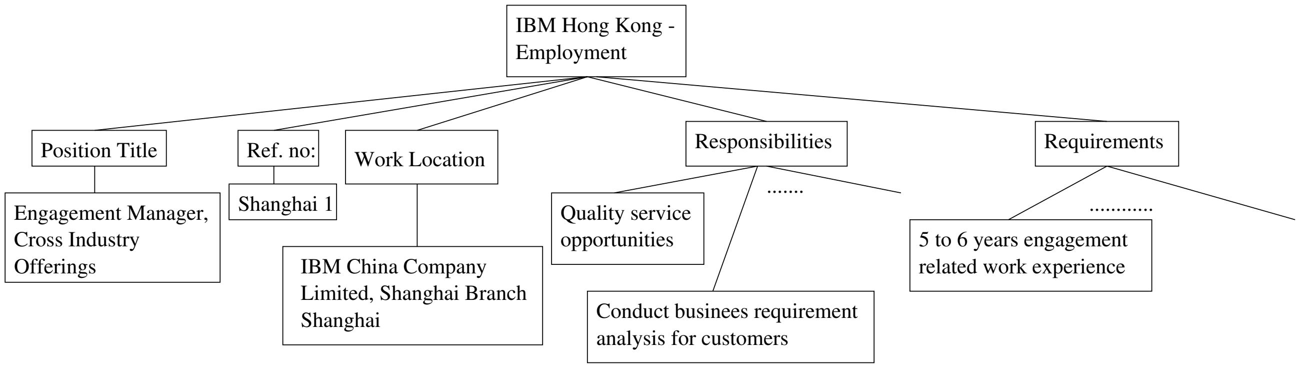

Figure 4: The hierarchical structure for Example 1

n_i	node i
d_o	own data set
d_p	parent data set
d_l	left siblings data set
d_r	right siblings data set
d_c	children data set
w_i	word i

Table 1: Notations

n_i, is associated with five sets of text data $\{d_o, d_p, d_l, d_r, d_c\}$. They are the sets of text data from the node itself, from the parent, sibling and child nodes. We call these the **data sets**. Note that the left (right) text data sets may correspond to more than one nodes. Some of these sets may be empty as some nodes may not have parent, left sibling, etc. Each data set consists of words, $\{w_1, w_2, \ldots\}$. These notations are summarized in Table 1.

For the example in Figure 4, for the node "Position Title"
d_o = (Position, Title)
d_p = (IBM, Hong, Kong, Employment)
d_r = (Ref, no, Work, Location, Responsibilities, Requirements)
d_c = (Engagement, Manager, Cross, Industry, Offerings)
$d_1 = \phi$

We also eliminate some stop words from the above data sets.

5.2 Phase 1: remove non-label nodes

In a web page, we are interested in the labels and values for its class. In Phase 1, we identify those nodes that would not contain a label in their own data set. We make use of the following rule.

Rule 1 *A node n in the hierarchical structure does not contain a label in its own data set if it satisfies either one of the following conditions.*

1. *The right sibling data set and the children data set of n are empty.*
2. *There is a node in the same web page that have the same own data set as n.*
3. *There is a node in another web page that have the same own data set, children data set, left sibling data set and right sibling data set as node n.*

Arguments: (1) For each label, the value is located either at the right sibling or children node in the hierarchical structure. Therefore, if the two neighbour nodes are empty, the own data set of the node n does not contain a label.

(2) From observation 1, labels are always different for different attributes. A label would not occur more than once in a web page. If more than one node have the same own data set, then it would not contain a label.

(3) Some data may be repeated in multiple web pages. For example, in the class of job opportunities pages the company name, the company address, the company email, the company telephone number would always appear. These data would be put in the similar format within the same company's pages. For example, company name is put before company address and company address is put before company email, etc. Therefore, the nodes containing these data would have the same relations in the hierarchical structure of each web page. Then these nodes have the same own data set, left sibling data set, right sibling data set and children data set in another page. □

5.3 Phase 2: Identify label nodes

In Phase 2, we shall try to identify nodes that are likely labels. We shall make use of the following definition:

$$\textit{support of a text data} \\ = \frac{\textit{number of pages containing this text data}}{\textit{total number of pages}}$$

We have the following steps:

(1) After removing all those nodes that do not have a label, we count the number of occurrences of nodes with the same text data in the set of training web pages. We calculate the support of the text data. If a text data has a high support, then it is possibly a label for an attribute in the class. We set a **min_support** threshold so that if the support of a text data is above this threshold, it is considered as label.

(2) Merge the nodes with the same label. The right siblings and children data sets of the merged nodes are union. The merged node is called a **label node**.

Definition 6 *The union of the child and right sibling data sets of a node N are called the **feature** of the node N.*

(3) For each resulting label node, The frequency of each word in the feature data set is determined.

5.4 Phase 3: Merge similar label nodes

In definition 3, a *value* is an instance of the corresponding label. In the hierarchical structure described in previous section, the values of a label are mainly located at the children nodes of the node containing that label. Also, sometimes the values may be located at the right sibling nodes. By the comparison of some characteristics related to the features of two nodes $N1$ and $N2$, we can measure the similarity between $N1$ and $N2$. In Phase 3, we discover nodes with similar features, even if their labels may be different. This will help us to identify the set of labels for an attribute. Also the sets of labels will form the structure of the class.

Next we shall discuss how to measure the similarity of features. For the same label, there will usually be some common words that appear in the corresponding values. For example, in the label *requirements* of job opportunities pages, the word *required* will usually appear in the corresponding value in different pages. We use the ***confidence*** to measure the informative level of a word in a data set. Let us call the nodes merged to form a label node n the *nodes in n*. The confidence of a word, w_i, in the data set d of a label node n is calculated as **confidence** of w_i =

$$\frac{\text{number of nodes in } n \text{ that contain } w_i}{\text{number of nodes in } n}$$

For example, if there are 100 web pages containing the label in label node n and so they are merged to form n, and there are 50 nodes in n having the word w_i in the feature data set, then the confidence of w_i in the feature data set of n is $50/100 = 0.5$.

5.4.1 Similarity function

For information retrieval, a document is typically represented by a feature vector $X = (x_1, x_2, ..., x_n)$ where the element x_i is the frequency of the word i in that document. A similarity function that is used frequently for feature vectors X and Y is the cosine function: $Cosine(X, Y) =$

$$\frac{X \cdot Y}{\sqrt{(X \cdot X) \bullet (Y \cdot Y)}} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} {x_i}^2 \sum_{i=1}^{n} {y_i}^2}}$$

where the big dot ($\bullet$) denotes the scalar multiplication.

In our problem, we want to determine nodes with similar labels. We measure the similarity between their features given by the feature data set of the two nodes, d_{f1} and d_{f2}. We use the cosine function above. feature data sets The feature vectors used in calculation is different when it compares different nodes. The average size of feature vectors is then quite small as many nodes contain few words in the children data set.

The values in the two feature vectors are the *confidence* of the corresponding word in the two nodes respectively. So, if a node does not contain a word w_i in its children data set, then the value of this word's position x_i in the feature vector is zero. So, the product $x_i y_i$ is zero. We can consider the words in the children data set of nodes, $d_{f1} \cap d_{f2}$. Therefore the similarity function used to determine the closeness of two nodes, n_1 and n_2, in phase 2 can be written as follows. $similarity(n_1, n_2) =$

$$\frac{\sum_{w_i \in d_{f1} \cap d_{f2}} con_1(w_i) \times con_2(w_i)}{\sqrt{\sum_{w_j \in d_{f1}} con_1(w_j)^2 \sum_{w_j \in d_{f2}} con_2(w_j)^2}}$$

where $con_1(w_i)$ and $con_2(w_i)$ are the confidence of word w_i in d_{f1} and in d_{f2} respectively.

5.4.2 Finding and Merging Similar Nodes

In Phase 2, we obtained a set of nodes which may contain a label in its own data set. From this set of nodes, we find the similar nodes. We merge the similar nodes and combine the features of that nodes. This process is called ***MergeSimilarNode***.

There is a pre-defined threshold, ε_s, such that if $similarity(n_1, n_2) < \varepsilon_s$, then n_1 and n_2 are said to be different. There is another condition that two nodes will not be similar. From Observation 1, if two nodes occur in the same page, then they must not correspond to the same attribute. So they should not be similar to each other and the similarity between these two nodes is 0.

For each node, n_i, we find the node that have the highest similarity to n_i. *Two nodes are merged if and only if they have the highest similarity to each others.* The two similar nodes are then remove from the set and the resulted node from merging these two nodes is added to the set.

```
Algorithm 1 MergeSimilarNode(K)
Q := φ;
  ▷ Find similarity of all nodes in K
for each nodes n_i ∈ K do
    max_sim(n_i) ← 0;
    sim_node(n_i) ← φ;
    for each nodes n_j ∈ K and n_i ≠ n_j do
        if similarity(n_i, n_j) > ε_s and similarity(n_i, n_j)
        > max_sim(n_i) then
            max_sim(n_i) ← similarity(n_i, n_j);
            sim_node(n_i) ← n_j;
        end if
    end for
end for
  ▷ Find all similar node pair and merge them
for each nodes n_i ∈ K do
    if sim_node(n_i) = sim_node(sim_node(n_i))
    then
        n_s = merge(n_i, n_j);
        add n_s to Q;
        remove n_i and n_j from K;
    else
        add n_i to Q;
    end if
end for
K := Q;
```

Figure 5: Algorithm MergeSimilarNodes

$merge(n_i, n_j)$: when two nodes n_i, n_j are merges, the corresponding data sets, excluding the own data set, of two nodes are union. The confidences of the words in the feature data set are then updated. The own data sets are merged together by grouping the phrases or words in the two nodes. For example, if two nodes have the own data set (responsibilities) and (job responsibilities) respectively, then the resulted nodes have the own data set ((responsibilities) (job responsibilities)).

MergeSimilarNode is repeated until no similar nodes remain in the set.

5.5 Results of the Algorithm

The results of the algorithm are (1) structure of the given class: a set of labels for each attribute in the class of web pages. We also refer to each set of labels as a **node** in the structure. (2) feature text data for each set of labels.

6 Classifying a web page

Given any web page, we are interested to know if it belongs to a certain class. For example we may want to locate web pages that contain job descriptions. So we scan web pages and determine if each belongs to the class of pages for jobs. Note that this is different from the more common classification problem: given a number of classes, classify each given object as one of the classes. Here we are given only one class at a time, and we want to see if an object belongs to the class. If there are multiple classes, the same object can be discovered to belong to more than one class.

By the method presented in the previous section, we can compute the structure of the given class. The feature of the resulting labels are also discovered. Now, given a web page to be classified, the hierarchical structure of the web page can be extracted by the mechanisms in Section 4. A set of nodes in the hierarchical structure would be found. Each of these nodes is then compared with the nodes in the structure of the given class.

First, if the own data set of a node appears in a label set in the class structure, then the similarity between this node and the class structure node would be 1. Otherwise, as the value may occur in the right sibling or children nodes. The union of the two corresponding data sets, *d*, of the node will be compared with the feature, *f*, of the class structure by the following similarity function

$$sim(d,f) = \frac{\sum_{w \in d \cap f} con(w)}{\sum_{w \in d} con(w)} \times \frac{m}{N_d}$$

where *con(w)* is the confidence of word w in the feature *f*, m is the number of matched words, and N is the number of words in the data set *d*.

The similarity of a node is the highest similarity among all the labels. To distinguish the web pages, we have two pre-defined thresholds, ε_n and ε_m. If a web page has enough number of nodes in its hierarchical structure (greater than ε_n) that has a large similarity value (greater than ε_m) then it is classified as the given class. These two threshold could be obtained by using a training set of positive examples and a training set of negative examples. From the results, we use the two values that can provide the optimal result as the threshold.

7 Experiments

The proposed methodology was applied to a set of job opportunities web pages. We extract the structural knowledge from 4000 web pages of 34 different companies. Most companies have about 100 web pages. These companies are chosen arbitrarily from the company listings of yahoo search engine[1].

7.1 Results of LD Algorithm

Some results in the structure was shown in Figure 6. In Figure 6, the upper box contains those words or phrases (labels) representing the same attribute and the

[1] http://www.yahoo.com/

lower box contains the feature of the label node. From the feature, we can even discover some seldomly used phrases, e.g. in Figure 6, the phrase *minimum education experience required* only occurs in 2 web pages. The results have a few errors. For example, in figure 7, *scope* and *job responsibilities* appear in the same node. However, the correct extraction outweighs the errors and the end result in classification is highly accurate.

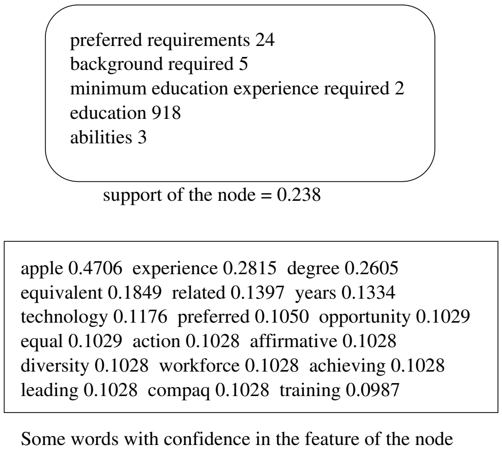

Figure 6: One node of the knowledge extracted.

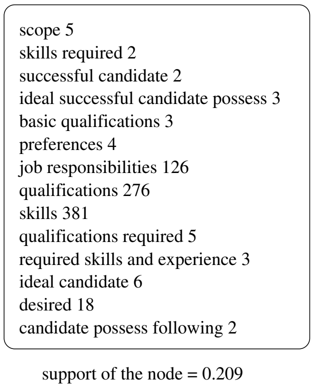

Figure 7: One node of the knowledge extracted.

7.2 Results of Web Pages Classification

The structural knowledge of the class of web pages could be used to distinguish web pages so as to extracted data from correct web pages. We use the knowledge obtained by the 4000 web pages to distinguish 10897 web pages in which 3651 pages are job opportunities pages of 23 different companies and 7246 pages are other classes. These 23 companies are also chosen arbitrarily from the company listings of yahoo search engine. They are different to those companies of the 4000 web pages used in the extraction of knowledge. The 7246 negative web pages are retrieved from yahoo search engine. By using the yahoo search engine, web

ε_n	ε_m				
	0.5	0.6	0.7	0.8	0.9
1	99.92%	99.92%	99.92%	99.92%	99.64%
2	99.10%	98.74%	98.74%	95.59%	95.59%
3	89.43%	88.88%	88.88%	88.50%	88.50%
4	72.20%	72.20%	72.20%	72.06%	72.06%
5	36.65%	36.65%	36.65%	36.65%	36.64%

Table 2: Correctness for classification of positive web pages

ε_n	ε_m				
	0.5	0.6	0.7	0.8	0.9
1	96.41%	96.41%	96.41%	96.42%	96.42%
2	99.77%	99.77%	99.77%	99.77%	99.78%
3	99.99%	99.99%	99.99%	99.99%	99.99%
4	100.00%	100.00%	100.00%	100.00%	100.00%
5	100.00%	100.00%	100.00%	100.00%	100.00%

Table 3: Correctness for classification of negative web pages

pages are iteratively retrieved by following the links in the retrieved web pages. Then we obtained 7246 pages that are not related to job opportunities web pages by removing those web pages containing the keyword *job*.

For different threshold, ε_m and ε_n, we would first measure the accuracy of distinguishing positive web pages, which refer to the correct pages, and the accuracy of distinguishing negative web pages, which refer to the wrong pages. The result is shown in table 2 and 3.

Table 2 shows the accuracy of correctly identifying the web pages in the class. Table 3 shows the accuracy of correctly identifying the web pages that do not belong to the class. From the result, we could observe that the effect of ε_m is lower than that of ε_n on the accuracy of distinguishing web pages. The result shows that the structural knowledge could distinguish the web pages accurately at a precision of about 99%.

8 Conclusion and Future Works

In this paper, we have proposed an approach to discover the typical *structure* of a class of web pages. We propose a definition of *structure* and show that discovering such structures can lead to successful web page classification. Experiments have been conducted on using the structural knowledge to distinguish web pages. By using the knowledge obtained, we can distinguish the web pages accurately. The structural knowledge could be further applied to extract the useful data stored in web pages by locating the labels in the web pages.

Our approach is also applicable to XML [8][18] files. XML is typically more structured than HTML. The result of our approach may be more accurately. Also, if label could be discovered more accurately and similar labels could be discovered with fewer errors, then

HTML files could be converted to XML files by the labels discovered.

References

[1] S. Abiteboul. Querying semi-structured data. In *Proceedings of ICDT*, pages 1–18, Delphi, greece, January 1997.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semistructured data. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, 1997.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference*, pages 207–216, Washington, DC, 1993.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of VLDB*, pages 487–499, 1994.

[5] P. Buneman. Semistructured data. In *Proceedings of PODS*, pages 117–121, Tucson, Arizona, May 1997.

[6] P. Buneman, S. davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference*, pages 505–516, Montreak, Canada, June 1996.

[7] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The tsimmis project: Integration of heterogeous information sources. In *Proceedings of Tenth Anniversary Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.

[8] The World Wide Web Consortium. http://www.w3.org/XML/.

[9] D.W. Embley, D.M. Campbell, Y.S. Jiang, Y.K. Ng, R.D. Smith, S.W. Liddle, and D.W. Quass. A conceptual-modeling approach to extracting dta from the web. In *Proceedings of the 17th International Conference on Conceptual Modeling, ER'98, Lecture Notes in Computer Science, 1507*, pages 78–91, Singapore, November 1998.

[10] D.W. Embley, Y. Jiang, and Y.K. Ng. Record-boundary discovery in web documents. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 467–478, Philadelphia, PA USA, 1999.

[11] M. J. Carey et. al. Towards heterogenous multimedia information systems: the garlic approach. In *Proceedings of Fifth International Workshop on Research Issues in Data Engineering (RIDE): Distributed Object Management*, pages 123–130, Los Angeles, California, 1995.

[12] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the web. In *Workshop on Management of Semistructured Data*, Tucson, Arizona, 1997.

[13] T. Kirk, A. Levy, J. Sagiv, and D. Srivastava. The information manifold. Technical report, ATT Bell Laboratories, 1995.

[14] S. Nestorov, S. Abiteboul, and R. Motwan. Extracting schema from semistructured data. In *Proceedings of ACM SIGMOD International Conference*, pages 295–306, Seattle, Washington, USA, 1998.

[15] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Deductive and Object-Oriented Databases(DOOD)*, pages 319–344, Singapore, December 1995.

[16] K. Shoens, A. Luniewski, P. Schwarz, J. Stamos, and J. Thomas. The rufus system: Information organization for semi-structured data. In *Proceedings of Nineteenth International Conference on Very Large Databases*, pages 97–107, Dublin, Ireland, 1993.

[17] Ke Wang and Huiqing Liu. Discovering typical structures of documents: A road map approach. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and retrieval*, pages 146–154, Melbourne, Australia, 1998.

[18] XML.com. http://www.xml.com/.