Finding Structure and Characteristic of Web Documents for Classification

By

WONG, Wai Ching

Supervised By :

Prof. Wai-Chee FU Ada

A Thesis Submitted in Parital Fulfilment

of the requirements for the degree of

Master of Philosophy

in

Computer Science & Engineering

JUNE, 2000

Finding Structure and Characteristic of Web Documents for Classification

submitted by

WONG, Wai Ching

for the degree of Master of Philosophy

at the Chinese University of Hong Kong

Abstract

Large amount of data is available on the World Wide Web (WWW). If web pages could be converted to a database, then the users could retrieve data more efficiently. However, problems arise due to the semistructured nature of web data and many researchers are working on these problems recently. In this thesis, we address one of these problems: different wordings are used to represent the same attribute in different web pages.

Web data, like most semistructured data, is self-describing; i.e. the schema is contained within the data in a web page. Attributes of the schema would be highlighted explicitly by wordings, called **labels**, in some web pages, e.g. the wordings or label, "*job position*" is used to represent an attribute in a web page. In fact, different web pages may use different labels to represent the same attribute, e.g. "*job title*" is used instead of "*job position*" in other pages. This is a critical problem for retrieving the data of the attribute represented by "*job title*" and "*job position*" if we do not know that these two labels represent the same attribute. In this thesis, we propose a **labels discovery algorithm** to discover labels that represent the same attribute.

The major contributions of this thesis are: (1) we introduce a **hierarchical**

structure constructed by five heuristic methods which represents the hierarchical relations of each data in a web page, (2) we propose a labels discovery algorithm to discover labels representing the same attribute by using the relations among data in the hierarchical structure of web pages, and (3) we then propose a web pages classification method by using the knowledge obtained in the labels discovery algorithm. The labels discovery algorithm could discover labels of the same attribute as shown in the experiments. Also, by using the knowledge obtained in our algorithm, we obtain a high precision of classification of web pages in the experimental results.

論文題目： 尋找網頁的結構與特徵以應用於網頁分類

作者： 王威青

學校： 香港中文大學

學系： 計算機科學與工程

修讀學位： 哲學碩士

日期： 二零零零年六月三十日

摘要

萬維網存在著龐大數量的數據，如能將數據由網頁形式轉爲數據庫形式，用者便能有效率地由萬維網提取數據。但是，萬維網數據（web data）是半結構的（semistructured），而這種特徵導致提取數據存在難題。最近，許多研究員正在不斷尋找解決方法。其中一個難題就是在不同的網頁裏不同的字串代表同一屬性（attributes）。我們將會在本論文裏探討這個難題。

萬維網數據和它的結構（schema）一同儲於網頁內，而每個屬性則由字串代表，我們稱呼這些字串爲標簽（labels）。但是有時不同標簽會用來代表同一屬性，所以我們提出了標簽搜尋演算法（labels discovery algorithm）用來搜尋代表同一屬性的標簽。

本論文主要貢獻包括：（1）提出五個試探性方法（heuristic methods），從一個網頁建築一個分級結構（hierarchical structure）來代表網頁內的數據的關係；（2）利用分級結構顯示的關係，提出標簽搜尋演算法來搜尋代表同一屬性的標簽；（3）利用標簽搜尋演算法的結果進行網頁分類。實驗演示標簽搜尋演算法能夠在同一種類的網頁內，發現代表同一屬性的標簽。另外，由這些網頁得來的結果應用於網頁分類，實驗演示準確度很高。

Acknowledgments

First of all, I would like to thank Prof. Ada Fu, my supervisor, for her guidance and patience. My research could not have been done reasonably without her insightful advice. For the past two years, she gave me encouragement, support, and guidance on my paper (DMKD2000) and my thesis. It would not be possible to get my paper published without her, and not even to USA for my presentation.

My great gratitude goes to Prof. Man-Hon Wong and Prof. John Chi-Shing Lui, who marked my term paper and gave me valuable suggestions.

My gratitude goes to the Department of Computer Science & Engineering, CUHK. It provides the best equipment and office environment required for high quality research to our students.

Finally, I thank to my fellow colleagues, who helped me in solving the programming, computer and official problems, and enlightened me to the new research ideas. They are (not in order), Chun-Hing Cai, Kin-Pong Chan, Chun-Hung Cheng, Wang-Wai Kwong, Po-Shan Kam, Wai-Chiu Wong, Yin-Ling Cheung, Wai-To Chan, and Men-Hin Yan (database research group), Yuk-Chung Wong (AI tutor), Ka-Po Ma, Tsui-Ying Law, Kwong-Wai Chen, Yin-Hung Kuo, Wing-Kai Lam, Ka-Lung Chong, Hing-Wing Chan, and Ting-On Lee(discussion group). They gave me a happy and wonderful university life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

There are over half a billion homepages on the World Wide Web (WWW) and a thousand more are appearing each hour. WWW provides a vast resource for information of almost all types, ranging from DNA databases to resumes to lists of favorite restaurants. However, the semi-structured nature of information on the web leads to a critical problem that users cannot directly sift through and interpret all the information.

Users commonly retrieve web data by browsing and keyword searching. In fact, browsing is not suitable for locating particular items of data [5] because following links is tedious, and it is easy to get lost. Besides, browsing is not cost-effective as users have to read the documents to find the desired data. For example, if the users want to find the requirements of a particular job in the web page, they have to locate the information by reading the whole web page. On the contrary, keyword searching is sometimes more efficient than browsing. But it often returns vast amounts of data, much more than the user can handle. Sometimes the number of web pages returned by a search engine is in the order of ten thousands. A tiresome exhaustive manual browsing is often needed to accurately pinpoint the data that users require even though the returned web pages are ranked.

Different types of data are available on the web: text, image, sound, movie, etc. In particular, the amount of textual data available on the web is estimated to be in the order of one terabyte. The web can then be seen as a very large, unstructured but ubiquitous database [7]. For that reason some researchers have resorted to ideas taken from database techniques in order to retrieve data more efficiently from the web. Databases, however, contain structured data and most web data is semistructured in nature [9]. This means that web data does not have a rigid structure and cannot be retrieved easily by using traditional techniques. The main reason of this problem is that web pages publication usually does not involve any publishing authority to critique, edit, or verify the accuracy of the material. Due to this loose standard, web data will be stored in different formats.

1.1 Semistructured Data

Semistructured data [1, 9] is characterized by the lack of any fixed and rigid schema, although, unlike unstructured raw data, typically the data has some implicit structure, which is neither regular nor known a-priori to the system. Information of semistructured data is normally associated with a schema and is contained within the data. In other words, attribute names are stored with the data, hence most semistructured data model are self-describing. In some forms of semistructured data, there is no separate schema. In others, it exists but loose constraints are placed on the data.

The lack of external schema information makes browsing and querying these data sources inefficient and even impossible at worst. For instance, a user finding a job instance in a traditional relational database would know the structure of the job. The user would know all the attributes of this instance. As an example, the schema of that database would tell us that each job has job name, description and requirement. However, in a semistructured world, some jobs may have job

name and description only. Other jobs may have job name, description and job location, instead of requirement.

The web data is an example of semistructured data because the standard on web page publication is loose. As stated in [20], data found over the web is generally fairly irregular. For example, the majority of home-pages of job openings may contain some similar pieces of information (e.g. position, responsibilities, qualifications) but some of these may be missing in some particular pages, and extra information may be present in others. In [20], the authors said that irregularities are often the norm in data found over the web. They arise naturally when one integrates data originating from several distinctly structured sources that provide information about a common set of entities but which represent these entities differently.

There are several ways to deal with the lack of fixed schema. If the semistructured data is somewhat regular but incomplete, then an object-oriented or relational schema can be used to represent the data along with null values. This approach fails if the semistructured data is very irregular. As a consequence, trying to fit the data into a traditional database form will either introduce too many nulls or discard most of the information [29].

A lot of recent work has been targeted to handle this problem [2, 6, 11, 14, 18, 28]. However, the previous approaches require users to specify a particular structure for different set of web documents. This process mainly depends on human input. There is an enormous amount of web documents and they contain different types of information. It is an issue to specify the data model for each type of information. These approaches are then suitable to some specific data. On the whole, people are still trying to discover the mapping of web pages to database.

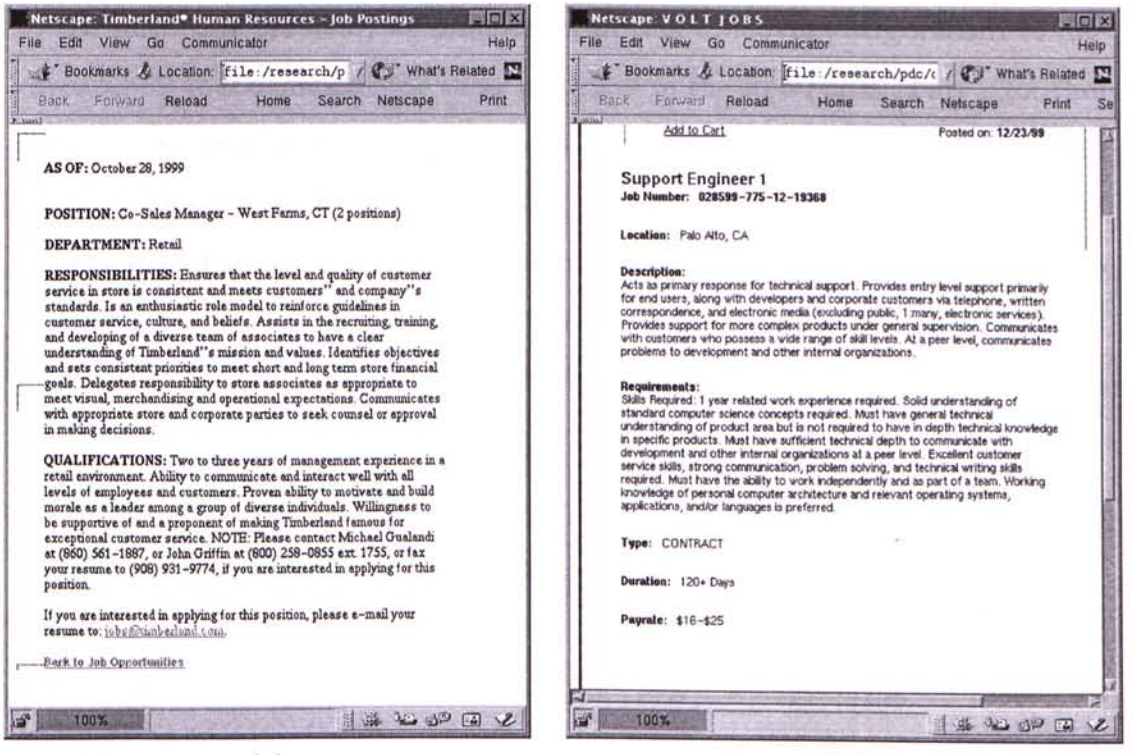
1.2 Problem Addressed in the Thesis

Many problems on web data retrieval arise because of the semistructured nature of web data. Many researchers are working on these problems, e.g. modeling web data as database objects [21, 24, 19], finding structure of web page [20, 31], extracting data on the web [12, 8, 15], etc. In this thesis, we focus on one problem of the loose standard of web page publication: different wordings are used to represent the same attribute. As a result, the objective of the thesis is to develop a methodology that discover a set of wordings that represent the same attribute in a collection of web pages.

1.2.1 Labels and Values

In the thesis, a web page is assumed to contain two main categories of information: **label** and **value**. Web data, like most of semistructured data, is self-describing. In a manner of speaking, the schema is described within the data in a web page. Attributes of schema would be stored explicitly with their names by wordings in the web pages and these wordings are called labels. Each label represents an attribute. Besides, the data of attributes in the web pages are called value. To illustrate, there are two web pages in Figure 1.1. In Figure 1.1(a), "*POSITION*" is a label used to represent an attribute in that web page. Then the value of this label is "*Co-Sale Manager - West Farms, CT(2 positions)*". Details will be described in Chapter 3.

To determine the labels in web pages, a **hierarchical structure** is then introduced to reveal the concept hierarchical relation of the data in the web page. Based on the syntax and properties of HTML tags, we proposed five heuristics to extract the hierarchical structure for each web page. From the structure, we can observe that most of the values are the children of their corresponding labels



(a) (b)

Figure 1.1: Example of wordings problem of attributes in web data

in the web page.

1.2.2 Discover Labels for the Same Attribute

For an attribute, labels are different in a set of web pages. For instance, in a job database, there is one attribute, say, description which is used to describe the duties of a job. Some web pages may use "*description*" but others may use "*responsibilities*". Yet another may use "*job duties*". As it turned out, we do not know which labels represent the same attribute. To illustrate this problem, consider the two web pages shown in Figure 1.1. In Figure 1.1(a), "*RESPONSIBILITIES*" represents the same attribute as "*Description*" in Figure 1.1(b). Without knowing the attribute represented by each label, the information exchange among different sources of data would be difficult.

To solve this problem, we proposed a **labels discovery algorithm** to discover labels that represent the same attribute in the web pages. This algorithm analyzes the text in the web documents based on the hierarchical structure of the web documents. In a database, the data of the same attribute are similar in nature. For example, for the requirement attribute of job opportunities web pages, the word 'required' may occur frequently. So, we can use this characteristic to discover labels that represent the same attribute by comparing the values of each label.

1.2.3 Classifying A Web Page

For the fact that about a thousand more web pages are appearing each hour, the database of any specific type of web data must be updated frequently. In fact, only a small portion of the new web pages belong to the database. Therefore we need to classify the new web pages so as to extract data from them. 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.

The knowledge obtained from our approach in the thesis can be used as the classifiers of a particular class. The knowledge is a set of label-sets where each label-set represents an attribute of the class along with the attribute's characteristics. For different classes, the set of attributes and their characteristics would not be the same and they can then be used to classify web pages.

1.3 Organization of the Thesis

The thesis is organized as follows.

In Chapter 2, we will give some backgrounds on the problem of web data by introducing some previous works in web data. We will pay particular attention to the structure extraction of web pages and the particular data retrieval in web pages. Moreover, most of the contents of web pages are text, therefore text processing techniques would be required to extract specific feature from the web pages. Some techniques of text processing will be introduced.

In Chapter 3, we will define the problem to be tackled in the thesis. First of all, we will define the structure of a web page. Also, the format of storing data in a web page will be described. We assume two main categories of data in a web page: labels and values. Afterwards we will describe the problem of finding similar labels.

In Chapter 4, we will introduce the hierarchical structure which is a concept hierarchy of the data in web pages. The structure organizes the data according to their concept hierarchical level. Five heuristics are used to construct the hierarchical structure according to the properties of HTML tags.

In Chapter 5, we will describe our approach on discovering similar labels in a class of web pages. The algorithm is based on the hierarchical structure extracted from each web pages. Each label will have a feature describing the topic of that label. Based on the feature, we then discover similar labels. In order to measure the representative of the knowledge obtained, it is used to classify web pages.

We will give a conclusion work in Chapter 6.

Chapter 2

Background

In this chapter, we will first introduce some issues on web data structuring. To illustrate these issues, some recent works are introduced. The data type on the web is mostly text. They are stored as articles in web pages. In an attempt to retrieve web data, text processing techniques must be applied to the content of web pages. Text processing techniques are used in traditional information retrieval on documents. For that reason, some techniques would be discussed afterwards.

2.1 Related Work on Web Data

In this section, some recent works of treating web data as a database will be discussed. There is no any research working on the topic discussed in this thesis. Therefore, we will introduce some works that are related to web data. Firstly, we will introduce an object-based information exchange model called **object exchange model (OEM)** [21] which is used to model semistructured data. Web data can be structured by this model [11, 24]. Afterwards, works on the problem of mining structure of web documents will be discussed. Mining the structure of web documents is critical to extracting information from them. The

works of extracting schema and discovering typical structure will be discussed. Last we wll discuss the work on information extraction of web data.

2.1.1 Object Exchange Model (OEM)

The Object Exchange Model (OEM) was introduced in the Tsimmis project at Stanford [21] which was designed for semistructured data. Some projects are based on this model to extract information from web data [19, 24]. The data in the model is a labeled directed graph. Some vertices have labeled edges pointing to another vertices. A movie document in OEM is shown in Figure 2.1. The vertices in the graph are objects with the following structure:

Label	Type	Value	Object-ID

where the four fields are:

- **Label**: A variable-length character string describing what the object represents.
- **Type**: The data type of the object's value. Each type is either an atom type (such as integer, string, real number, etc.), or the type set. The possible atom types are not fixed and may vary from information source to information source.
- **Value**: A variable-length value for the object.
- **Object-ID**: A unique variable-length identifier for the object or null.

In Figure 2.1, each vertex has a data &x where x is a number. This is the object-ID of the vertex. The word on the edge pointed from one vertex v_1 to another vertex v_2 is the label of v_2. For example, in Figure 2.1, there is a word "Title" on the edge pointed from &1 to &2 and "Title" is the label of &2. In the leaf node, e.g. &2, "Star Wars" is the value of &2.

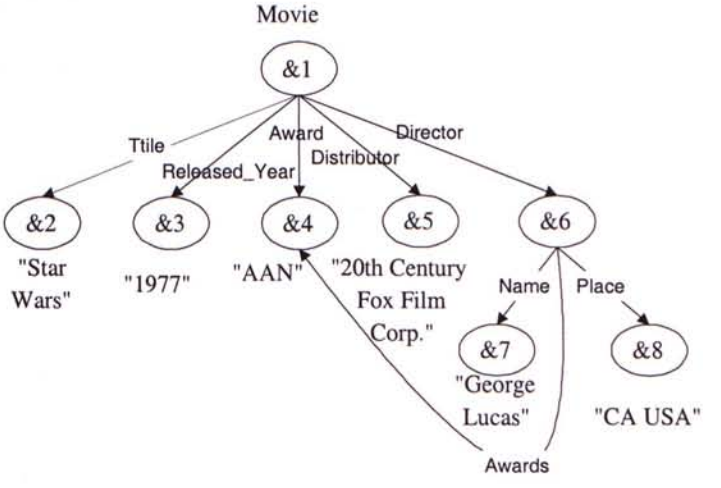


Figure 2.1: A movie document in OEM format

There are some works of extracting semistructured data based on the OEM. One of them is the TSIMMIS [11]. It uses a configurable tool for extracting semistructured data from a set of HTML pages and for converting the extracted information into database objects [15]. The input to the extractor is a declarative specification that states where the data of interest is located on the HTML pages, and how the data should be "packaged" into objects. The output data is in the form of the OEM.

A problem of this approach is that the extraction mechanism depends on user input for describing the structure of HTML pages. This becomes an issue when the structure of source files changes rapidly requiring frequent updates to the specification file. Besides, this approach can only extract information from a set of web pages with the same structure. In fact, web pages with the same type of information have slightly different structure. Hence some information will be lost in this approach.

The OEM is similar to a tagged file system [32]. The system uses labels instead of positions to identify fields. For example, electronic mail messages consist of label-value pairs (e.g. label "From" and value "wcwong@cse.cuhk.edu.hk").

However, in web pages, the label-value pairs do not follow a strict and regular format due to the irregularity of web data. What is more, in OEM, the label is defined uniquely in the data model; in contrast, in this thesis, labels may be different even if they describe the same attribute (it is the object class in OEM) in the web pages. OEM cannot handle this problem.

2.1.2 Schema Extraction

The problem of extracting schema from semistructured data is considered in [20]. It considers a general form of semistructured data based on labeled, directed graphs [10, 23]. The nodes in the graph represent objects and the labels on the edges are the semantic information about the relationships between the objects. The leaf nodes in the graph represent atomic objects that have values associated with them. An example is shown in Figure 2.2.

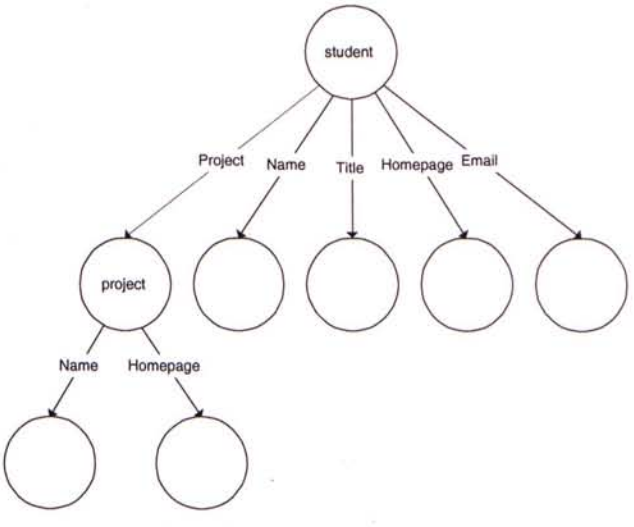


Figure 2.2: An example of database

Data sets found on Internet have no explicit structure and are fairly irregular but has implicit structure. The implicit structure in a particular data set may be of varying regularity. The objective of [20] is to find a typing that fits the data set. A typing is a schema of the data set. It contains a set of types in which

each type labels data in the data set similar to a table in a relational database. For instance, in Figure 2.2, there are two types: student and project. Student type describes the data of a student in the database. A student data should have several values: project, name, title, homepage and email as shown in the graph. The typing of the database is specified by a datalog program which is shown in Figure 2.3. The program shows the format of the types in the database. If a data in a data set matches the format of one type then the data is said to be of this type. In the typing program, the direction of the arrow over the label of an edge denotes whether the edge is incoming (left) or outgoing (right). The superscript denotes the type of the object at the other end of the edge. 0 indicates that it is atomic object that have values. 1 indicates that it points to type 1 and 2 indicates that it points to type 2.

$$\textit{project: } \tau_1 = \overleftarrow{Project}^2, \overrightarrow{Name}^0, \overrightarrow{Homepage}^0$$
$$\textit{student: } \tau_2 = \overrightarrow{Project}^1, \overrightarrow{Name}^0, \overrightarrow{Title}^0, \overrightarrow{Homepage}^0, \overrightarrow{Email}^0$$

Figure 2.3: A typing program for Figure 2.2

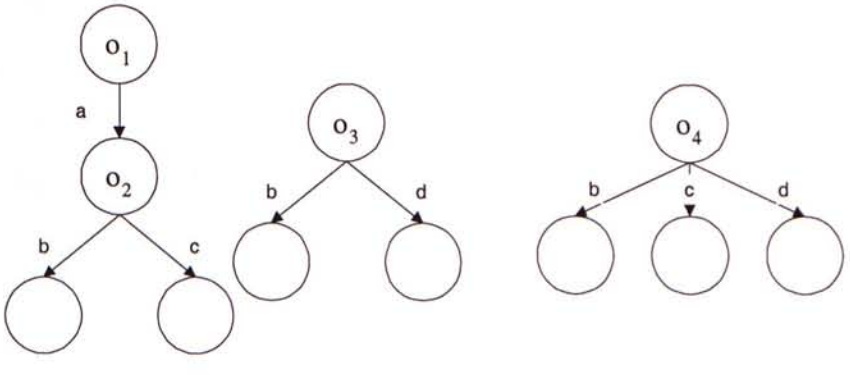


Figure 2.4: Example database

The objective of [20] is to find typing that approximately type a data set. The size of typing should be small and the typing should type the data in the data set with minimum error. Suppose we are given the database shown on Figure 2.4. To match all objects precisely to a type, it is true that four types are required without errors and the following typing program is the perfect typing

of the database.

$$
\begin{aligned}
type_1 &= \overleftarrow{a}^2 \\
type_2 &= \overrightarrow{a}^1, \overleftarrow{b}^0, \overleftarrow{c}^0 \\
type_3 &= \overleftarrow{b}^0, \overleftarrow{d}^0 \\
type_4 &= \overleftarrow{b}^0, \overleftarrow{c}^0, \overleftarrow{d}^0
\end{aligned}
$$

o_1, o_2, o_3, o_4 are mapped to $type_1, type_2, type_3, type_4$ respectively. However the size of a perfect typing is then very large which is roughly of the order of the size of the data set. Therefore, it is necessary to reduce the size of typing but there may be errors. For example, if we map o_4 to $type_2$ then the size is reduced to 3 but there is no link a from o_1 to o_4 and the link c from o_4 is missing.

The typing problem is first cast into the datalog program as shown in Figure 2.3 by assigning each object, excluding atomic object, a unique type. Then the size of typing is large. So it computes the greatest fixpoint M of the typing program. The function $M(type_i)$ returns a set of objects o_i such that $type_i$ coincides with the objects o_i on a link $\overrightarrow{c}^i$ or $\overleftarrow{c}^i$. For example, in Figure 2.4, there is a type $type_3 = \overleftarrow{b}^0, \overleftarrow{d}^0$. Then $M(type_3) = \{o_3, o_4\}$ as both o_3 and o_4 have these links $\overleftarrow{b}^0$ and $\overleftarrow{d}^0$. Then a more precise typing is generated by using the greatest fixpoint. An example of this process is shown in the following.

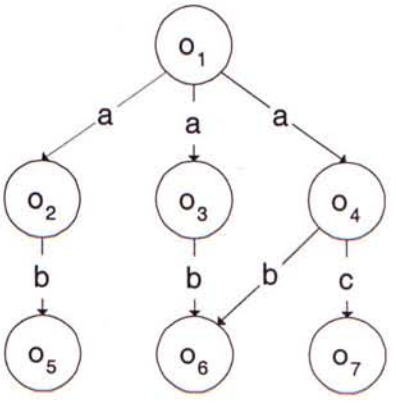


Figure 2.5: A simple database

Example 1 Consider the database D in Figure 2.5. The program Q of this database constructed initially is:

$type_1 = \overrightarrow{a}^2, \overrightarrow{a}^3, \overrightarrow{a}^4$ $\quad type_2 = \overleftarrow{a}^1, \overrightarrow{b}^0$

$type_3 = \overleftarrow{a}^1, \overrightarrow{b}^0$ $\quad type_4 = \overleftarrow{a}^1, \overrightarrow{b}^0, \overrightarrow{c}^0$

The greatest fixpoint M for Q is: $M(type_1) = \{o_1\}$, $M(type_2) = M(type_3) = \{o_2, o_3, o_4\}$, $M(type_4) = \{o_4\}$. Then the perfect typing is obtained by letting $\tau_1 = type_1$, $\tau_2 = [type_2, type_3]$ and $\tau_3 = [type_4]$. The program of this perfect typing is:

$\tau_1 = \overrightarrow{a}^2, \overrightarrow{a}^3$ $\quad \tau_2 = \overleftarrow{a}^1, \overrightarrow{b}^0$

$\tau_3 = \overleftarrow{a}^1, \overrightarrow{b}^0, \overrightarrow{c}^0$ ∎

The perfect typing program obtained will have many types to be similar to each other. They could then be collapsed into one in order to reduce the size and complexity of the typing program. Therefore the perfect types are clustered using a greedy algorithm to compute k types such that the sum of the distances of each type to the closest (among the k types) is minimized. It is similar to k-clustering [16]. The greedy algorithm gives an $O(\log n)$ approximation of the optimal solution.

The distance used to measure the distance between each types is the Manhattan path between two type points on the binary hypercube defined by the typed links in their definitions. That means the distance is the number of typed links in the symmetric difference between the bodies of their rule definitions. Following example shows the measurement of distance between two types, $d(\tau_1, \tau_2)$.

Example 2 Consider the following three types:

$\tau_1 = \overrightarrow{a}^0, \overrightarrow{b}^2$ $\quad \tau_2 = \overrightarrow{a}^0, \overrightarrow{b}^1$

$\tau_3 = \overrightarrow{b}^2, \overleftarrow{b}^1, \overleftarrow{b}^3$

For τ_1, τ_2 the symmetric difference consists of $\{\overrightarrow{b}^2, \overleftarrow{b}^1\}$, so $d(\tau_1, \tau_2) = 2$. For τ_1, tau_3, the symmetric difference consists of $\{\overrightarrow{a}^0, \overrightarrow{b}^1, \overleftarrow{b}^3\}$, so $d(\tau_1, \tau_3) = 3$. And

$d(\tau_2, \tau_3)$ is also 3. ∎

2.1.3 Discovering Typical Structure

In [31], the proposed method discovers the typical structures of semistructured documents. The structure of a document refers to the role and hierarchical relationships of subdocument references. We use an example in the original paper to explain. For example, the structure of a person document can tell that the person has labeled fields *Name*, *Address*, *Hobby*, and *Friend* and that the *Address* subdocument has labeled fields *Street* and *Zipcode*. It states that documents describing the same type of information would have similar structure. For example, every club documents has *Name* label and at least 10 *Player* labels; every player document has *Name* label; 50% player documents have *Nationality* label, etc.

The problem addressed is: given a collection of documents, find all "typical" structures that occur in a minimum number of documents specified by the user. The authors use OEM for representing the semistructured documents. They state that every node in OEM represents a subdocument(e.g. a HTML file) and has an identifier(e.g. URL). The arrows and their labels, identifiable by special tags or a grammar, represent subdocument references and roles. An example is shown in Figure 2.6.

The authors express the structures of several documents by a tree-expression to generalize the structure. The tree-expression of a structure is a tree representation $\{l_1 : te_1, \ldots, l_k : te_k\}$ in which te_i is a tree-expression of a subtree pointed by an edge labeled l_i from the root node of the structure. In the tree-expression, ? is used as the wild-card that matches any level. $\perp$ denotes nil schema containing no structure. For example, there are some tree-expressions of club documents shown in Figure 2.7. The expression $te_1 = \{Player : \{Name : \perp\}, Name : \perp\}$ is

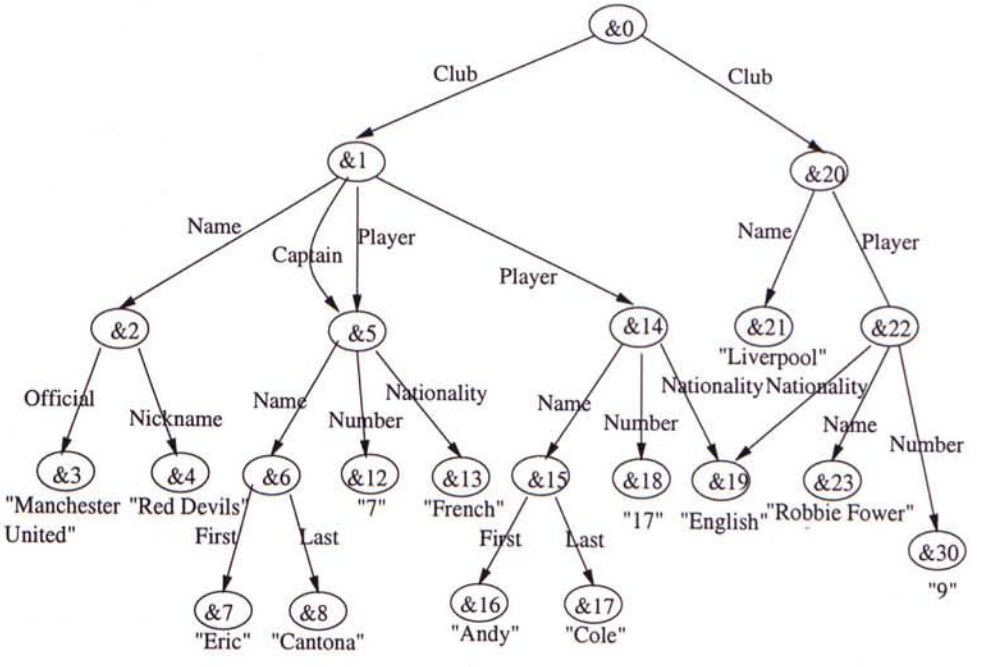


Figure 2.6: The premiership document

a tree-expression of the structure of document &20 in Figure 2.6. There are two subtree pointed from the node &20. The first one is pointed by the edge labeled "Name" but the subtree pointed is a null subtree, i.e. the subtree does not have any nodes, so the tree expression of this subtree is $\perp$. Another subtree is pointed by the edge labeled "Player". This subtree has a tree expression $\{Name : \perp\}$.

The typical structure discovery problem is to find all maximally frequent tree expressions of a collection of documents. An algorithm similar to that for mining association rules [3, 4] is used to find the typical structure of a collection of structures. In [4], constructing larger candidate subsets is done by joining smaller frequent subsets, and support counting is done by set containment test.

Similarly, the proposed method in [31] finds all frequent 1-tree-expressions which are tree-expressions containing one leaf node. Then it finds frequent k-tree-expressions $p_1 ... p_{k-2} p_{k-1} p_k$ by constructing two frequent (k-1)-tree-expressions $p_1 ... p_{k-2} p_{k-1}$ and $p_1 ... p_{k-2} p_k$. The support for each tree-expression is calculated like mining association rules. If the support is greater than a pre-defined thresh-

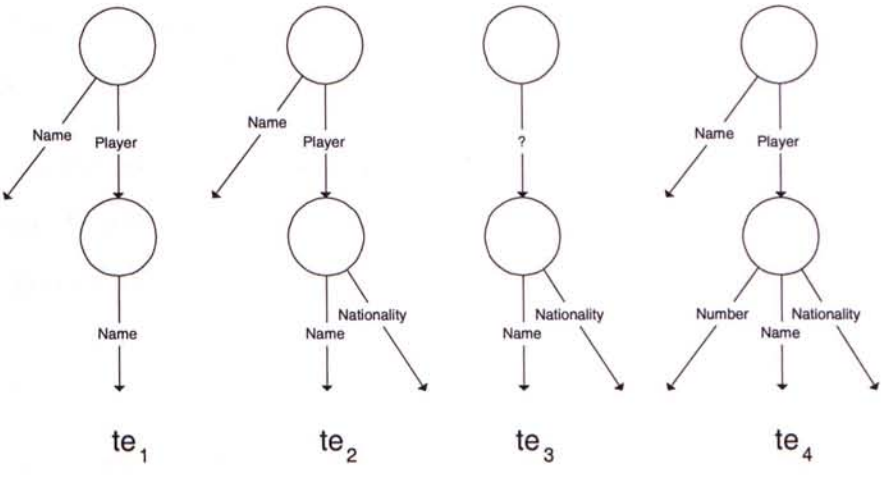


Figure 2.7: Tree-expressions of club documents

old *MINISUP*, then it is used to generate (k+1)-tree-expression. Lastly, all non-maximally tree-expressions are pruned. A frequent tree-expression te is said to be maximal if it is not weaker than other frequent tree-expressions, i.e. it is not less informative than other frequent tree-expressions. For example, in Figure 2.7, te_2 is weaker than te_4 as it is less informative than te_4.

Typical structures of a large number of documents can be used to discover the general information content and representation in the source. However, this approach only considers the relation between documents, i.e. the hyperlink information of web documents. It pays little attentions to the content of web documents.

2.1.4 Information Extraction of Web Data

There are some works on extracting particular data from the web documents. One of these is [8] which extracts a relation of (author, title) pairs from the web. The problem addressed in [8] is as follow. Let D be a large database of unstructured information such as the web. Let $R = r_1, ..., r_n$ be the target relation. Tuples of R occur in various locations in D using a variety of formats.

The goal is to discover as many tuples of R as possible with few, if any, false positives.

The algorithm proposed is called DIPRE - Dual Iterative Pattern Relation Expansion. It relies on a duality between patterns and relations. The proposed method first finds the pattern of the initial sample of book in a set of web pages. A pattern is a five-tuple: (order, url-prefix, prefix, middle, suffix) where order is a boolean value and the other attributes are strings. The order is used to indicate the position of author and title in the pattern expressions. If order is true, an (author, title) pair matches the pattern if there is a web document with a URL which contains url-prefix as its prefix and which author and title are in the following expression: prefix, author, middle, title, suffix. If order is false, then the title and author are switched.

author	title
Isaac Asimov	The Robots of Dawn
David Brin	Startide Rising
James Gleick	Chaos: Making a New Science
Charles Dickens	Great Expectations
William Shakespeare	the Comedy of Errors

Table 2.1: Initial sample of books

After finding the patterns of the initial sample, the method uses these patterns to discover other (author, title) pairs. Then the patterns of the newly discovered pairs are found in the collection of web documents. This process is repeated until the relation pairs found is large enough. The initial sample of books used is shown in Table 2.1 and the patterns of these books found is shown in Table 2.2.

URL Pattern	Text Pattern
www.sff.net/locus/c.*	<LI><B>*title*</B> by *author* (
dns.city-net.com/lmann/awards/hugos/1984.html	*title* by *author* (
dolphin.upenn.edu/dcummins/texts/sf-award.htm	*author* || *title* || (

Table 2.2: Pattern found in first iteration

This approach can find particular data of relation pairs in the web. It can help to retrieve the same type of information from different sources. However, web data is semistructured in nature and some relations might be missing. For example, a movie relation (name, director, actors, actresses, distributor) may miss director in some pages or distributor in other pages. Therefore, it can not discover a large set of data if the relation contains many attributes. It is for simple data only and does not consider the document content.

2.2 Automatic Text Processing

Traditional information retrieval would pre-process the documents in order to obtain the feature. Text processing techniques are applied to the documents [26]. Most of the content of web pages is textual. To obtain knowledge from the textual content, we have to apply text processing techniques before we can analyze the web pages. In this thesis, we will use stopwords elimination to remove stopwords from web pages and apply stemming to the remained words.

2.2.1 Stopwords Elimination

In traditional information retrieval, a word which appears in each of the documents in the collection is completely useless as an index term because it does not tell us anything about which documents the user might be interested in. On the other hand, a word which appears in just five documents is quite useful because it narrows down considerably the space of documents which might be of interest to the user. The words in first case are frequently referred to as stopwords.

Most web pages contain textual content and they can be seen as documents in traditional information retrieval but with some structure. We want to discover similar labels in a set of web pages in the thesis. Therefore the textual data should

be distinguishable. Nevertheless, stopwords are not good discriminators. Thus, they have to be removed from the set of web documents in order to improve the precision of our algorithm. Articles, prepositions, and conjunctions are natural candidates for a list of stopwords. Also, some verbs, adverbs, and adjectives can be treated as stopwords.

2.2.2 Stemming

Frequently, the same word is presented in different forms in a collection of documents. Plurals, gerund forms, and past tense suffixes are examples of syntactical variations. They would be treated as different word although originally they are the same word. This problem can be partially overcome with the substitution of the words by their respective stems. A *stem* is the portion of a word which is left after the removal of its affixes (i.e., prefixes and suffixes). A typical example of a stem is the word *connect* which is the stem for the variants *connected*, *connecting*, *connection*, and *connections*. Stems reduce variants of the same root word to a common concept.

In our methodology, we use affix removal as the stemming strategies. In affix removal, the most important part is suffix removal because most variants of a word are generated by the introduction of suffixes (instead of prefixes). While there are three or four well known suffix removal algorithms, the most popular one is that by Porter [22] because of its simplicity and elegance. Despite being simpler, the Porter algorithm yields results comparable to those of more sophisticated algorithms.

The Porter algorithm uses a suffix list for suffix stripping. The idea is to apply a series of rules to the suffixes of the words in the text. For instance, the rule,

$$s \longrightarrow \phi$$

is used to convert plural forms into their respective singular forms by substituting the letter s by nil. Notice that to identify the suffix we must examine the last letters in the word. Furthermore, we look for the longest sequence of letters which matches the left hand side in a set of rules. Thus, application of the two following rules

$$sses \longrightarrow ss$$

$$s \longrightarrow \phi$$

to the word stresses yields the stem stress instead of the stem stresse. By separating such rules into five distinct phases, the Porter algorithm is able to provide effective stemming while running fast. A detailed description of the Porter algorithm can be found in [22].

Chapter 3

Web Data Definition

We first define some notions of web data that will be used in our problems. Note that when we refer to a web page in this thesis, we refer to its textual content only on the ground that most of the data on the web is text. The hyperlinks would not be considered because we only concentrate on the web data stored in a web page and not the relationship between web pages. Afterwards our problems will be described in details.

3.1 Web Page

Web data is stored in web pages in a style that users can understand the concept of the data easily, hence data of same kind of information is always put together in the same web page. In common, almost all data in a web page belong to the same kind of information. For example, in a soccer web page, almost all the data in the web page is related to soccer. There are few non-related data, e.g. advertisement, but this is only a small proportion of the data in the web page. Therefore, each web page is used to present one main type of information.

Assumption 1 A web page stores one main type of information. The type of

information stored in a web page w_i is denoted by $I(w_i)$. ∎

Web pages are clustered into different categories in the search engine according to their textual contents. Web pages containing the same type of information are considered to be in the same category. For instance, in Figure 3.1, the two web pages are in the *soccer* category of a search engine as they contain the same type of information. We say that these web pages in the same category belong to the same **class**. A class of web pages is similar to a table in relational database. Each web page in the class is considered as a record in the table. Each class also has a number of attributes that describe the data stored.

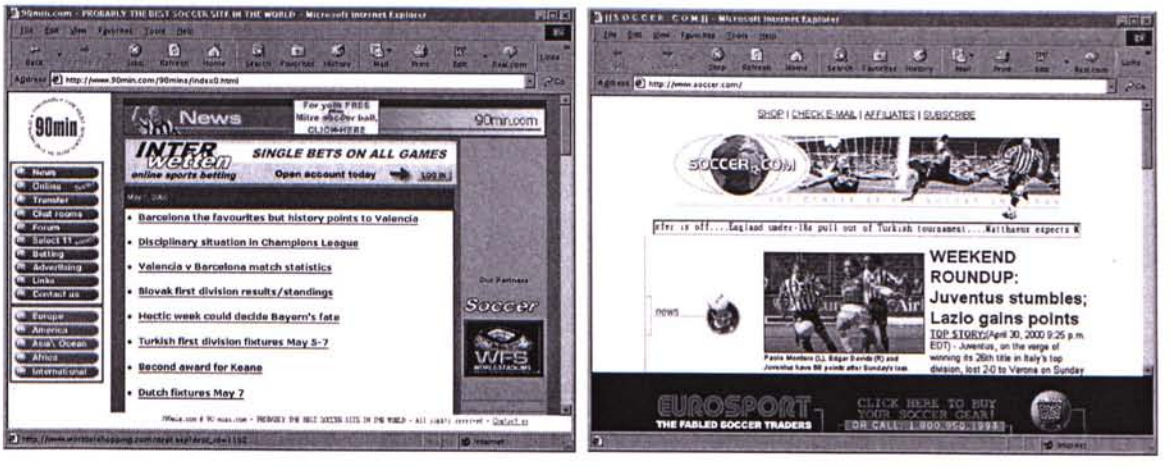


Figure 3.1: Two web pages in the same category of a search engine

Definition 1 A **class** of web pages is a set of web pages storing the same type of information. If a class contains web pages $w_1, w_2, ..., w_n$, then $I(w_1) = I(w_2) = ... = I(w_n)$. Each class of web pages has a number of attributes (A_1, A_2, ..., A_n) describing the data stored in the class. ∎

The schema of a web page is stored with the data. The **schema** is the set of attributes stored in the web pages. These attributes describe the data in the web pages. There are wordings used to label the attributes in each web page. The wordings used in a web page to label an attribute of a class is called a **label**

and the data of an attribute in the web page is called a **value**. Each label has its corresponding value in the web page.

Definition 2 A **label** l of a class C labels the data stored in a web page of class C. It corresponds to one of the attributes A_i of class C. We also denote the attribute that correspond to the label A by $A(l)$. Hence $A(l) = A_i$. ∎

Definition 3 A **value** is the data of an attribute stored in a web page. Each value has its corresponding label in a web page. ∎

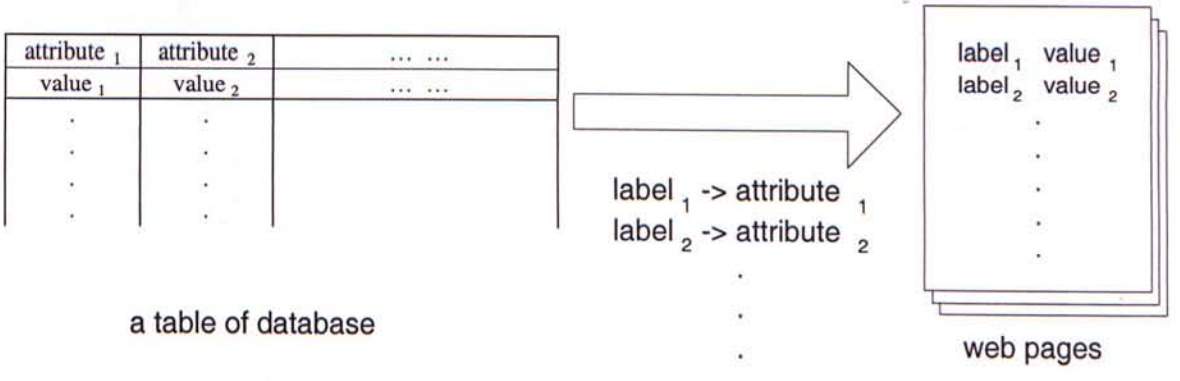


Figure 3.2: Web pages vs. relational database

The general overview of a class of web pages in comparison to a table of relational database is shown in Figure 3.2. Labels in a web page correspond to the attributes of the class. There are corresponding values to the labels and they are the data that users are most interested. A web page thus consists of a set of label-value pairs. As labels describe the data in the web page, the schema or structure of each web page can be represented by the set of labels in the web page.

Definition 4 A web page consists of a set of label-value pairs $((l_1, v_1), (l_2, v_2), ..., (l_n, v_n))$ where l_i is the label in the web page and v_i is the corresponding value of l_i. The schema or structure of a web page is represented by a set of labels stored in that web page $(l_1, l_2, ..., l_n)$. ∎

The following is an example illustrating the occurrence of labels and values in web pages.

Example 3 In a class of job opportunities web pages, we suppose that there are four attributes, say, (*category*, *name*, *description*, *requirement*). A job opportunities web page is shown in Figure 3.3. In this web page, the four data of the attributes are stored with the labels.

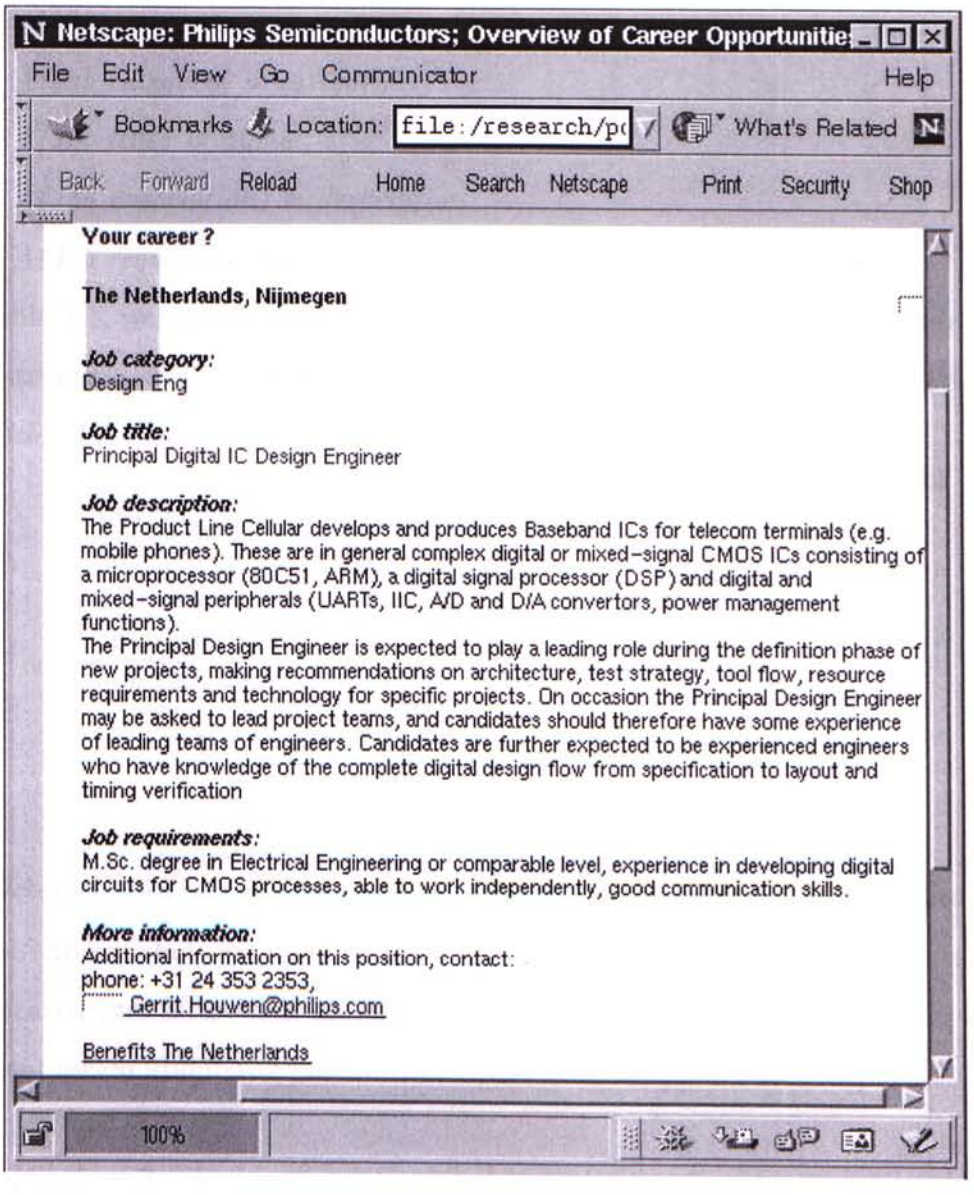


Figure 3.3: A web page of job opportunities

"*Job category* is the label, which is used to label the data of the attribute *category* as well as represent that attribute in this web page. The value of that label, i.e. the data of the attribute *category*, is "*Design Eng*". The three labels, "*Job title*", "*Job description*" and "*Job requirements*", are used to represent the attributes *name*, *description* and *requirement* respectively. Their values are "*Principal Digital IC Design Engineer*", "*The Product Line .....*" and "*M.Sc. degree .....*". The attributes represented by the labels in the web page are summarized by

$A(Job\ category) = category$

$A(Job\ title) = name$

$A(Job\ description) = description$

$A(Job\ requirements) = requirement$

In Table 3.1, we have the label-value pairs stored in the web page of Figure 3.3. The structure of the web page is represented by the four labels *(Job category, Job title, Job description, Job requirements).* ∎

label	value
Job category	Design Eng
Job title	Principal Digital IC Design Engineer
Job description	The Product Line .....
Job requirements	M.Sc. degree .....

Table 3.1: Label-value pairs of example 3

Web page can be considered as an interface for users to easily locate the data stored. Since labels are used to label the data, there is little doubt that values are located just after their corresponding label in the web pages. We can make an assumption about the positions of labels and values in the web pages.

Assumption 2 The location of a label is just preceding to its corresponding value in the web page. ∎

For example, in Figure 3.3, the label "*Job category*" is located next to its value "*Design Eng*" in the web page. Therefore a label can be used as an indication of the location of its corresponding value in the web page. We can extract the value by locating its corresponding label in the web page.

3.2 Problem Description

In a relational database table, an attribute has an unique reference name. On the contrary, since the standards are quite loose for publishing web pages, different labels would be used to represent the same attribute in a class of web pages. That is, the name of an attribute is not unique for all web pages in the same class. Different reference name of attributes can confuse the determination of the attribute of data in the web pages. As a result, data retrieval of web pages becomes difficult. For example, in Figure 1.1, the two web pages use "*responsibilities*" and "*description*" to represent the same attribute. What is more, we cannot determine whether "*responsibilities*" and "*description*" represent the same attribute. This is the main problem addressed in this thesis. In order to describe the problem, we first define some notions relating to properties of labels in a class of web pages.

From the observation of web pages, we can discover that the wordings used to represent two attributes in a class of web pages are different in most web pages. For example, in a class of job employment web pages, if "*responsibilities*" is used to represent the attribute, say, description, then it would not be used to represent other attributes, e.g. name, category, requirements, salary, etc. Therefore, we have an assumption on the labels.

Condition 1 Web pages in the same class will not use the same wordings to represent different attributes. For two labels l_1 and l_2 in the same class, if

$A(l_1) \neq A(l_2)$ then $l_1 \neq l_2$. ∎

Definition 5 (Identical labels) Two labels l_1 and l_2 in the same class of web pages are **identical** ($l_1 = l_2$) if the wordings of l_1 and l_2 are the same. By Condition 1, attributes represented by them are also the same, $A(l_1) = A(l_2)$. ∎

Definition 6 (Similar labels) Two different labels l_1 and l_2 ($l_1 \neq l_2$) in the same class of web pages are **similar** ($l_1 \sim l_2$) if the attributes represented by the labels are the same, $A(l_1) = A(l_2)$. ∎

The objective of this thesis is to discover similar labels in a class of web pages. One problem caused by the loose standard in web pages publication is the occurrence of similar labels in web pages of the same class. If similar labels can be identified, then data of a particular attribute in the class of web pages can be retrieved more efficiently.

For instance, we assume that a class of job employment web pages has the attribute *description*. Two web pages use two different labels, "*description*" and "*job summary*", to represent the attribute. If we do not know these two labels are similar, then either "*description*" or "*job summary*" may be considered as the attribute *description*. If users want to retrieve the data of *description*, then some data would be missed. Therefore, it is essential to discover that the two labels are similar ("*description*" $\sim$ "*job summary*") so as to retrieve all data of a specific attribute.

Moreover, the attributes of a class can be defined in terms of labels. By Definition 2, labels are used to represent attributes of a class. Due to the loose standard of publishing web pages, multiple reference names may be used for each attribute in the class of web pages. As it turns out, typically there is a set of

similar labels for each attribute in a class of web pages. Hence an attribute in the class could be represented by the similar labels set.

Definition 7 An attribute A_i in a class of web pages, C, is represented by a **label-set** $L = \{l_1, l_2, ..., l_n\}$. Each label in L occurs in some web pages of class C and the labels are similar to each other, i.e. they represent the same attributes, $\forall l_j \in L, A(l_j) = A_i$. ∎

Attributes are used to name different data categories in a relational database. They form the schema of a relational database. Likewise, the schema or structure of a class of web pages can also be described by a set of label-sets where each labels-set corresponds to an attribute in the class.

Definition 8 The **structure** or schema of a class of web pages, C, is a set of label-sets $\{L_1, L_2, ..., L_n\}$ where each labels-set L_i represent an attribute A_i in the class C. ∎

To illustrate the above definitions, we give an example of a class of employment web pages.

Example 4 Considering that a class of job openings web pages has three attributes: *name*, *description*, *requirement*. Each attribute is represented by a set of similar labels in the web pages: (title, position) for name, (description, duties) for description, (qualifications, requirements) for requirement. Then the structure of the class of job employment web pages is a set of label-sets for each attribute.

((title, position), (description, duties), (qualifications, requirements)) ∎

In our collection of web pages, we observe that most of the web pages contain a single record of data. Therefore, in this thesis, for simplicity, we consider single-record web pages only, e.g. for job openings, each web page contains information

about only one job opening. In relational database, a record contains one data value for each attribute. For instance, there will only be one data for the name of a job and one data for the requirement of job in a job employment record. In web pages, one record is stored in one page so each attribute of the class occurs at most once in the web page. Some attributes may be missing due to the semistructured nature of web data. Consequently, there is at most one label representing each attribute in the web page.

Condition 2 (Single-record web page) In a web page, only single record of data is stored. Each attribute of the class would appear at most once in a web page. ∎

Although we consider single-record web pages only, it is easy to extend our works to multiple-records web pages. The boundary for each record in the multiple-records web pages is required to be discovered first so that each record can be extracted [13]. If boundary can be found then a web page can be separated into several sub-webpages. Each sub-webpage contains a single record. Accordingly, we can consider each sub-webpage as a web page defined in this thesis. We can apply our methods to these web pages.

From Conditions 1 and 2, we can deduce one property of labels in a web page as follow.

Property 1 (Uniqueness of labels in one web page) Labels are unique within one web page. That is, all labels are different within one web page w; for a set of labels $\{l_1, l_2, ..., l_n\} \in w$, $l_1 \neq l_2 \neq ... \neq l_n$. Also, all labels are not similar, $l_1 \not\sim l_2 \not\sim \ldots \not\sim l_n$. ∎

Proof: In Condition 2, only a single record is stored in a web page and each attribute occurs at most once in a web page w. That means no two labels

represent the same attribute in one web page. That is, if labels $\{l_1, l_2, ..., l_n\} \in w$ then $A(l_1) \neq A(l_2) \neq ... \neq A(l_n)$. In addition, in Condition 1, for two labels l_j and l_k in the same class, if $A(l_j) \neq A(l_k)$ then $l_j \neq l_k$. Therefore we can deduce that all labels are different within one web page, $l_1 \neq l_2 \neq ... \neq l_n$. Likewise, by the definition of similar labels, all the labels in one web page are not similar, $l_1 \not\sim l_2 \not\sim \ldots \not\sim l_n$ as they represent different attributes. ∎

Chapter 4

Hierarchical Structure

Web data is structured by the HTML tags [30] in a web document. The data is organized in a format that users could understand easily. Therefore, the labels always appear just preceding their corresponding values in Assumption 2. For example, the label "*Position Title*" is located next to its corresponding value "*IT Specialist*" in a web page.

However, the computer does not know which data in the web page is a label as there is not any rule about which wordings should be labels or values. To organize labels and values in a web page, authors would store them in different patterns by using the properties of HTML tags. For instance, they may use the heading tag `H1, H2, ..., H6` to enclose label and use the list tag `ul` to enclose the corresponding value under the heading tag. For this reason, in many cases, we could discover label and value based on the properties of HTML tags.

Labels, like the headings describing the topic of a paragraph, describe the data of their values. Their concept is more general than that of their values in a web page. We say that they are at a higher concept hierarchical level than their values. Therefore, we should reveal the concept hierarchical relation of data in a web page. In our proposed method, a **hierarchical structure** of a web page is constructed based on the concept hierarchical relation of data in

the web page by five heuristics according to HTML tags' characteristics. The hierarchical structure shows the concept hierarchical level of data in a web page.

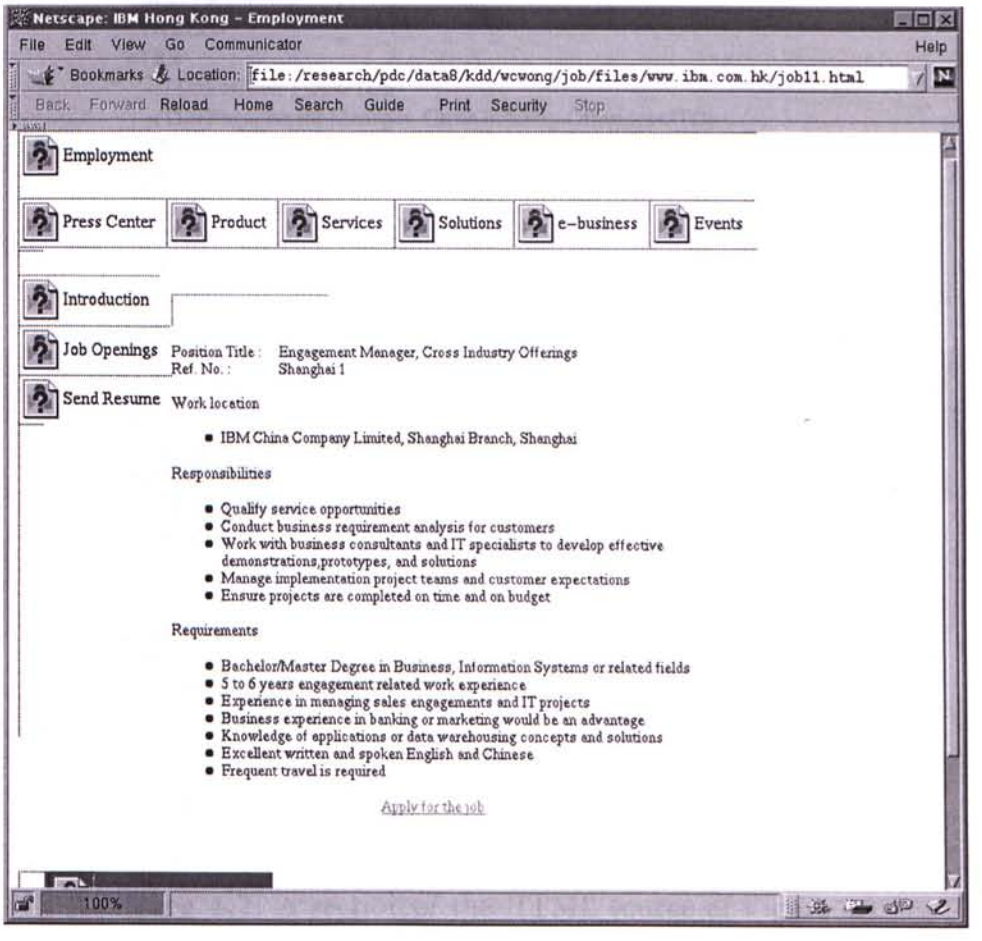


Figure 4.1: A snapshot of a web page

4.1 Types of HTML Tags

Web pages are constructed by the HTML tags which provide the format of displaying data stored in the web page. Figure 4.1 shows a web page storing the data of job employment. The data is organized by the HTML tags as shown in Figure 4.2. 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. Different tags would provide different information to the structure of the data. Therefore,

```
<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 4.2: A section of the HTML source of Figure 4.1

we divide the HTML tags into three groups with different structural information: **rigid structure tags**, **loose structure tags** and **non-structure tags**. As stated in Chapter 3, we consider only textual data in the web pages. The structural information provided by the tags is based on the textual data only. Other types of structural information would not be considered, e.g. images, video, hyperlinks, sound, etc.

Here are the details of the three types of tags.

1. **Rigid Structure Tags.** They provide structural information of the enclosed textual data. For example, the `table` tag provides a tabular structure of text data within it. There is some relation between data within

the same column or row in the table. In Figure 4.3, HTML codes of table and the display of these codes in web pages are shown. From the table, we could observe that there is relation between data within the same column, e.g. "Name" is the heading of the column.

```
<table>
<tr><th>Name</th><th>Age</th><th>Salary</th></tr>
<tr><td>Peter</td><td>39</td><td>$10000</td</tr>
<tr><td>John</td><td>50</td><td>$12450</td></tr>
</table>
```

HTML codes of a table

Name	Age	Salary
Peter	39	$10000
John	50	$12450

Table displayed in web page

Figure 4.3: HTML codes and display of table in web page

2. **Loose Structure Tags.** They give structural information to the data enclosed by themselves and other tags. For example, the heading tags, `H1`, `H2`, ..., `H6`, make the enclosed data as headings of different importance. Typically, H1 is the most important, H2 is comparably less important, and so on down to H6, the least important. These tags give hierarchical structure of the enclosed data.

3. **Non-structure Tags.** They do not provide any information to the structure of textual data in the web pages. They include tags that are used to enclose data other than text and all single tags. For example, the image tag, `img`.

The HTML tags in the three types of tags are shown in Table 4.1. The table shows most of the HTML tags but there are some other tags that are not included. Single tags that are not included in the table should be classified as

non-structure tags as they do not enclose any textual data in the web pages. Tags that enclose textual data are classified as loose structure tags if they are not in the table.

Types	HTML tags
Rigid structure tags	dir, div, dl, menu, ol, table, ul
Loose structure tags	title, h1, h2, h3, h4, h5, h6, p, b, i, em, strong, font, big, small, strike, ti, u, a, cite, dfn, code, samp, kbd, address
Non-structure tags	img, applet, map, area, hr, br, bgsound, base, frame, input, isindex, link, meta, param, sound, other single tags

Table 4.1: The three types of HTML tags

Since we want to extract the structure of a web page and non-structure tags do not provide structural information to the textual data, they can be ignored. Rigid structure tags and loose structure tags would be considered in the construction of hierarchical structure.

4.2 Tag-tree

Web pages have some linguistic conventions in their page layout. These conventions are determined by the HTML tags. HTML tags divide web pages 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** [12] can be constructed. The tag-tree in [12] contains HTML tags only as they focus on tags only. However, we would put the textual data in the tag-tree because our objective is to discover relations of textual data in the web page.

The textual data in a web page is divided into separated data which is enclosed by HTML tags. This separated data is called **text data**. Each text data

has its own meaning in the web page. They may be label or value.

Definition 9 A **text data** is the textual data enclosed by a pair of HTML tags, start and end tags, in the web pages. ∎

For example, in the following HTML segment,

```
<ul>
<li> Ability to work with senior management</li>
<li>60% of working time is in China</li>
</ul>
```

'Ability to work with senior management' and '60% of working time is in China' are two text data. Then a web page consists of a set of text data.

From the definition of text data in a web page, we could define the content of labels and values in the web pages in terms of text data.

Assumption 3 (Content of label) A label in the web page consists of exactly one text data. ∎

Assumption 4 (Content of value) A value in the web page consists of one or more text data. ∎

We assume that the label in a web page is a word or phrase which is enclosed by a pair of HTML tags. By Condition 2, an attribute occurs only once in a web page. So, there will be no other wordings used to represent the same attribute. A label then consists of one text data in a web page. On the other hand, usually, the data of an attribute would be organized in a list format or point format such that it could be understood by users. Each item of the formats is enclosed by HTML tags. Therefore there can be several text data for one value in the web page. For example, in Figure 4.1, the value of the label "*Responsibilities*" is divided into four items in which each item is enclosed by tags individually. Therefore there are four text data in the value of that label in the web page.

HTML tags could be enclosed by other tags. They are said to be nested in other tags. The following definition describes the algorithm used to construct a tag-tree by using this nested characteristic of HTML tags.

Definition 10 (Rules of tag-tree construction) Based on the nested property of HTML tags, two rules are used to construct the tag-tree.
Rule 1: If the tag t_1 or text data d is directly nested in tag t_2, then t_1 or d is the **child** of t_2.
Rule 2: If two tags (or text data), 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. ∎

In Figure 4.4, two examples show the rules of tag-tree construction. Circles represent tags in the tree and small boxes represent the text data enclosed in the tags. For Rule 1, as "tr" is nested in "table" directly, "table" is the parent of "tr" in the tag-tree. Likewise, "data1" is a text data enclosed in the tag "td", therefore "data1" is the child of "td" in the tag-tree. For Rule 2, the first "tr" that encloses "data1" is above the second "tr" that encloses "data2", so the first tr is the left sibling of the second one. Note that single tags and text data do not enclose other tags or data so they are the leaves of the tag-tree.

By applying these two rules to all the tags in a web page, we could obtain a tag-tree from the web page. In Figure 4.2, we show a segment of an HTML document. The snapshot of the corresponding web page is shown in Figure 4.1. The tag-tree of the document extracted by the two rules is shown in Figure 4.5.

The tag-tree is useful in the extraction of data from web documents [12]. From the tag-tree, the relation among discrete regions is represented by the tags in the internal node of the tree. The meaning of a subtree is represented by the tag in the root node of the subtree. For instance, in Figure 4.5, the subtree rooted at "TABLE" tells us that all the text data in it is organized

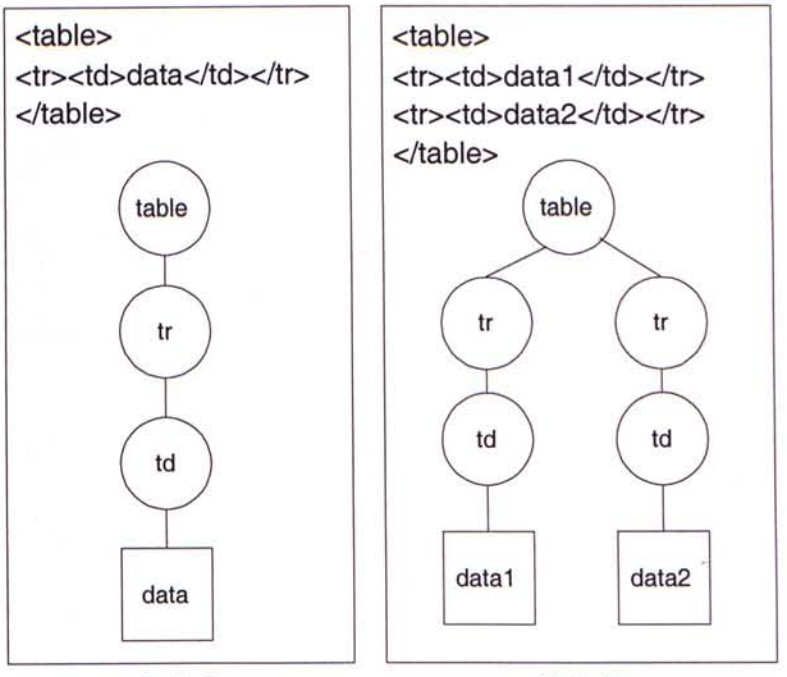


Figure 4.4: Example of the rules of tag-tree construction

in a table format. In addition, the number of rows is equal to the number of children of the "table" node and the number of columns is equal to the number of children of the "tr" node. Therefore it is more efficient to analyze the web page by using the tag-tree. In the construction of the hierarchical structure, the characteristics of HTML tags are used, so from the tag-tree we could determine these characteristics more efficiently.

After the construction of tag-tree, it has to be pruned. As we have mentioned in previous section, non-structure tags are useless in the formation of hierarchical structure, they could be removed. The subtrees of non-structure tags are pruned. However, some of the remaining tags in the tree would be required to be pruned. The reason is that some non-structure tags may be enclosed by rigid structure tags or loose structure tags. For example, the image tags may be enclosed in table tag. Obviously, the pruned tree may have tags as the leaves. These tags do not enclose any text data and they are useless in the hierarchical structure

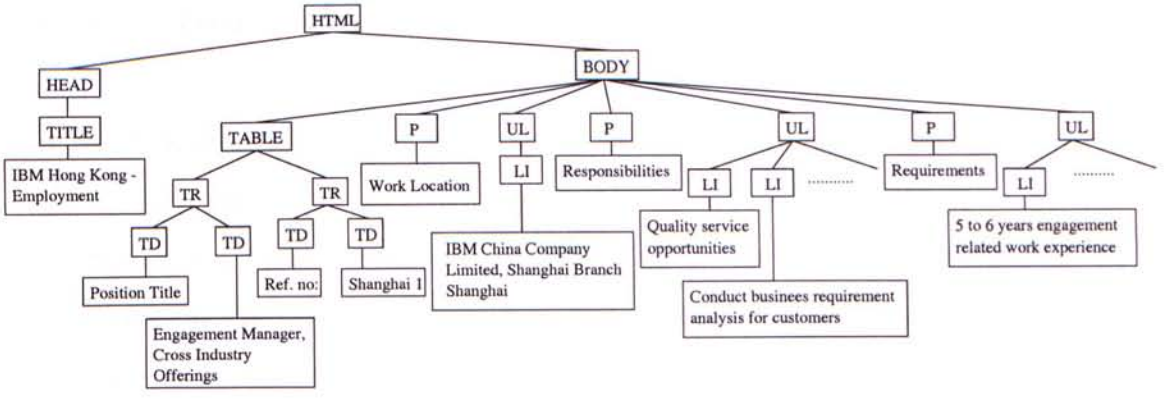


Figure 4.5: The tag-tree constructed from the web page in Figure 4.1

construction. So, they must be pruned. The pruning process is propagated upward in the tree until no such tags remain.

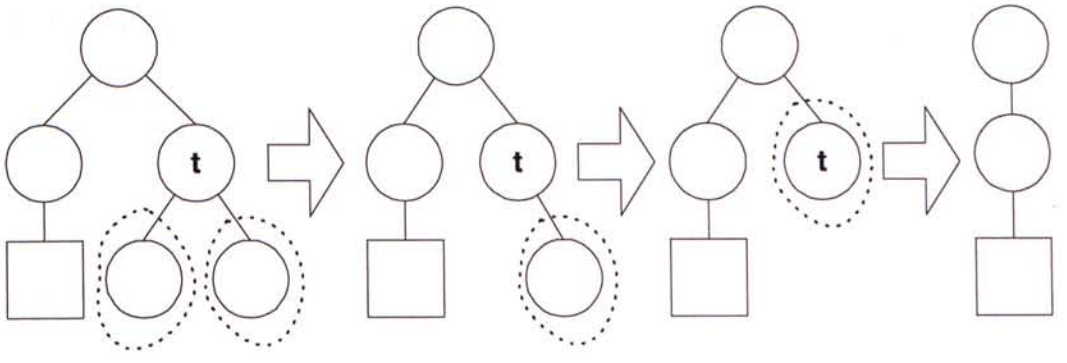


Figure 4.6: The process of pruning non-structure tags in tag-tree

The pruning process is illustrated in Figure 4.6. Circles represent tags and boxes represent text data. The dash line circle the tags to be pruned in the tag-tree. Each time one such tag is removed from the tree and some tags may become leaf nodes. The tag t becomes leaf node after removing its two children. Then t should be removed. The removal process is stopped unless the tree stll contains non-structure tags. Finally, the leaves of the tag-tree are all text data.

4.3 Hierarchical Structure Construction

A **hierarchical structure** is a concept hierarchical tree, in which each **node** corresponds to a text data in the web page. A text data with more general concept has a higher concept hierarchical level. The positions of data in the web page may indicate the concept hierarchical relation with other data. If two data are next to another one, then they are probably similar in the concept. However, labels and values are exceptions. For example, the data "*book title*" is always next to the title of a book, say, "*Thinking in C++*" but "*book title*" has a higher concept hierarchical level than "*Thinking in C++*" which is an instance of "*book title*". Similarly, label has a higher concept hierarchical level than its corresponding value as it is used to describe the value in the web page.

Definition 11 Label has higher **concept hierarchical level** than its value in a web page. ∎

In the hierarchical structure, the root of the tree is at **level** 0, we say that it is at the highest concept hierarchical level in the structure. A child of a node at level i is at level $i + 1$. We say that the node in level i has higher concept hierarchical level than the children of that node in level $i + 1$. By organizing the data in a hierarchical structure, we could determine the concept hierarchical relation more efficiently. In Figure 4.7, the hierarchical structure of the web page in Figure 4.1 is illustrated. From the structure, we could observe that "*IBM Hong Kong - Employment*" is the root node which has the highest concept hierarchical level. In fact, it is the title of the web page and it describes all data in the web page.

To construct the hierarchical structure, we use the characteristics of HTML tags. Each tag has some meaning to the page layout. They are used to display the text data in a format that is more convenient for the users to locate the

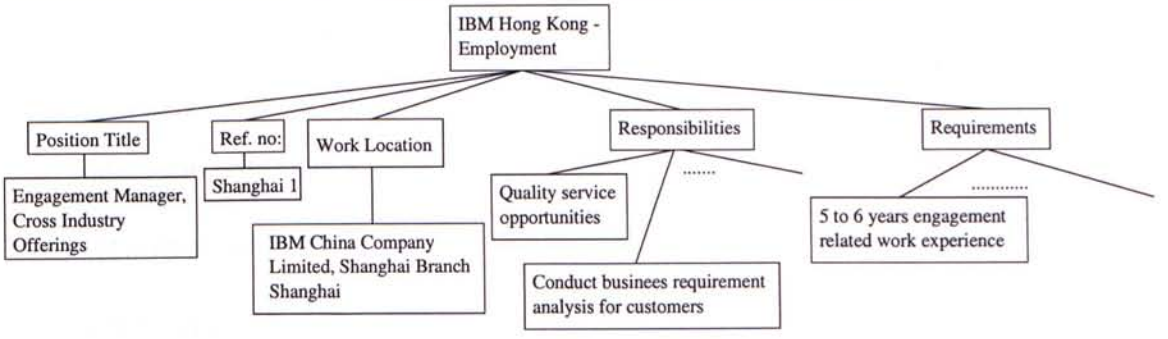


Figure 4.7: The hierarchical structure extracted from the web page in Figure 4.1

information in the web page. Based on the properties of the tags and the relations among them, we propose five heuristics to construct the hierarchical structure from a web page.

Heuristic 1: Heading loose structure tags

In the loose structure tags, there are heading tags (`H1, H2, ..., H6`) which are used to enclose the headings in the web pages. A heading briefly describes the topic of the section it introduces. There are six levels of headings in HTML with `H1` being the most important and `H6` the least. The text data enclosed by the heading tags are used to describe the topic of data in the section of web page under it. Therefore, its concept hierarchical level is higher than that of the data in the section under it. Likewise, the `title` tag is used to enclose the title of the web page. The title briefly describes the topic of the web page it introduces. Therefore the text data in the title has the highest concept hierarchical level in a web page.

The concept hierarchical level of the text data enclosed by heading tags is in the order as the number of the heading tags. That is, the text data in `H1` is higher than that in `H2` which is higher than those in `H3`, etc. However, there may be several headings with the same heading tag number. A web page is then divided into discrete region by the heading tags. The data in the region divided

by a heading tag is the children of the text data in that heading tag.

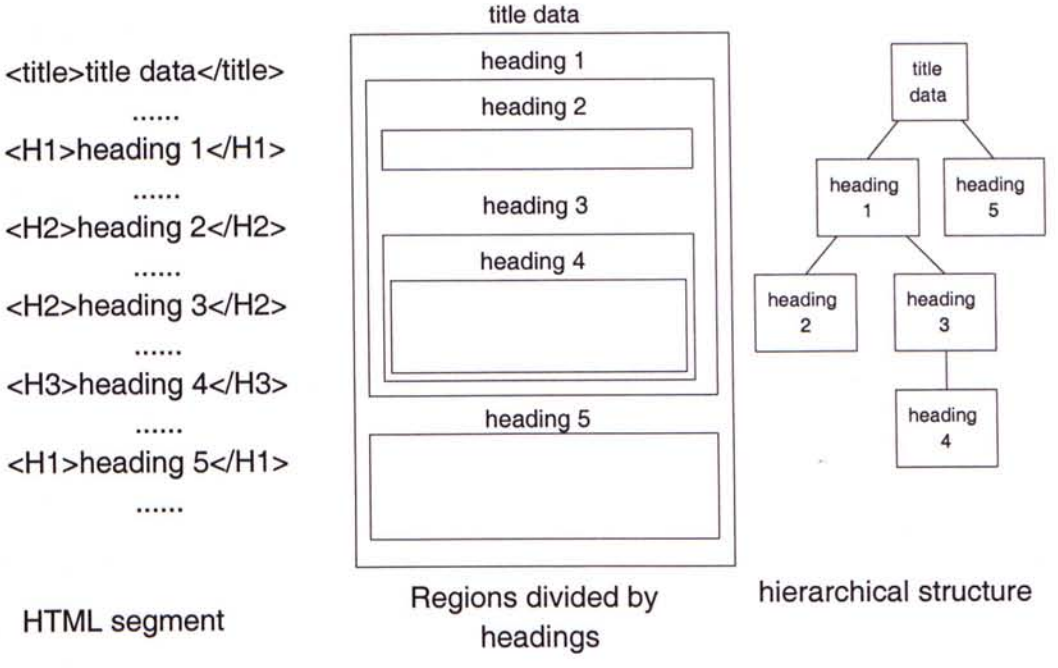


Figure 4.8: Example of Heuristic 1

An example is shown in Figure 4.8. We can see how the heading tags divide the web page. Boxes represent the region of data under the heading. "*heading 2*", "*heading 3*" and "*heading 4*" are in the region of "*heading 1*" as they are just below it and above "*heading 5*". Since "*heading 5*" is in `H1`, it is not included in the region of "*heading 1*". Moreover, "*heading 4*" is directly under "*heading 3*" so it is included in the region of "*heading 3*" not "*heading 2*". The hierarchical structure constructed is illustrated at the rightmost side of Figure 4.8. In the example, we ignore the other data in the web page and the structure contains only text data in the heading tags. This structure only displays the hierarchical structure of text data in heading tags. As "*title data*"is the title of the web page, it has the highest concept hierarchical level. Since "*heading 2*", "*heading 3*" and "*heading 4*" are in the region of "*heading 1*", they are under the subtree of "*heading 1*". Likewise, "*heading 4*" is the child of "*heading 3*".

Heuristic 2: Non-heading loose structure tags

Non-heading loose structure tags include all the loose structure tags excluding heading and title tags. There is no difference in the concept hierarchical level among them. That is, most of them have the same concept hierarchical level. However, some of them will be higher than others. To explain this, we should consider the difference between the headings and paragraph below them in an article or book. Headings describe briefly the topic of their following paragraph and they should have a higher concept hierarchical level as mentioned in Heuristic 1. As well, we should notice that the length of the headings must be shorter than their following paragraph. That is, the number of words in the headings is much fewer than the paragraph followed. Based on this pattern, we could distinguish the different level of text data enclosed by non-heading loose structure tags.

As there is not any rules guiding people to put headings into heading tags, people may use other tags to enclose the headings. Also, there is not any pattern of using which non-heading loose structure tags to enclose the headings. So, in order to reveal these headings, we could use the pattern of headings and their paragraph in an article described above. Then we have the follow heuristic. For the non-heading loose structure tags, if the length of a text data is shorter than its follow one, then it has a higher concept hierarchical level and it is the parent of the following text data. The length of text data is determined by the number of words in the text data. For example, the following is a HTML segment.

Position title

Principal Digital IC Design Engineer

The length of text data "*Position title*" is 2 and the length of "*Principal Digital IC Design Engineer*" is 5. Therefore, the concept hierarchical level of "*Position title*" is higher than that of "*Principal Digital IC Design Engineer*" and "*Position title*" is the parent.

Heuristic 3: Hierarchical rigid structure tags

Hierarchical rigid structure tags are rigid structure tags including all the list tags, `ul`, `ol`, `dir`, `div`, `dl`. These tags provide lists of information. Lists may also be nested and different list types may be used together, e.g. `ul` may be nested in `ol`. The text data in the list is embedded in the list item tags, e.g. `li` is the list item tag of `ul`. Therefore, these types of tags organize the text data in the order of concept hierarchical level. These tags could be seen as a concept hierarchy tree.

We first consider list tags without nested list. Usually, in the list of data, the list items are all related. Their concept hierarchical level are also the same. So, all the text data in the list is under the same parent in the hierarchical structure. We illustrated this case in Figure 4.9. The text data "*item one*", "*item two*" and "*item three*" are the list items and so they all under the same parent. Their location in the list determine their order in the tree. As "*item one*" is above "*item two*", it is the left sibling of "*item two*" in the tree.

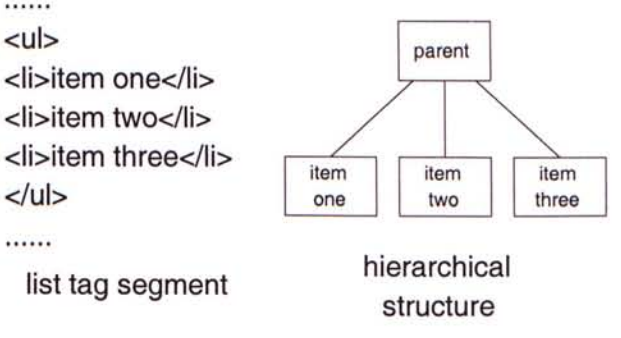


Figure 4.9: List tags without nested lists

List may contain other lists. One of the list item may be used to describe the topic of a list of information. Therefore, the nested lists have a lower concept hierarchical level. In addition, the nested lists are the children of their preceding text data. This case is shown in Figure 4.10. Like the previous case, "*item one*"

and "*item two*" are under the same parent and "*item one*" is the left sibling of "*item two*". A list is under "*item two*", so the list items are children of "*item two*". They are not the children of "*item one*" because they are not directly below it.

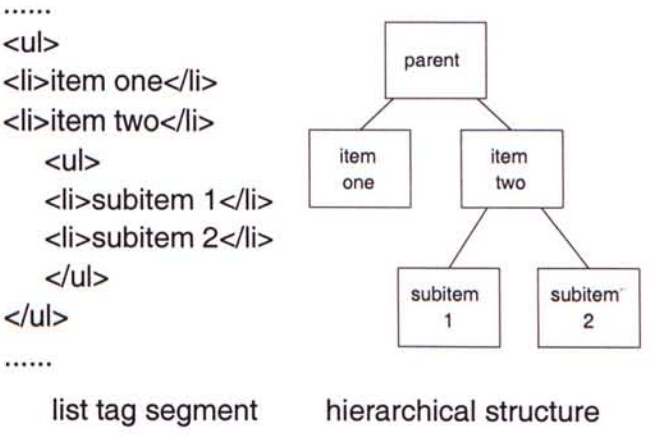


Figure 4.10: List tags with nested lists

Heuristic 4: Tabular rigid structure tag

Tabular rigid structure tags include `table` tag. This tag organizes the data into a tabular structure as shown in Figure 4.3. The concept hierarchy of the text data is implied in the table. However, there are two types of table that could occur in the web pages. One has header entry but another does not. The concept hierarchy in these two types of table are different. Also, each table may have an associated caption enclosed by the tag `caption`. It describes briefly the topic of the data in the table. So, the concept hierarchical level of the caption is higher than all data in the table and it is the parent.

In web pages, the header entry is enclosed by the tag "th". If the header entries are in the first row, then this type of table is similar to a table in relational database. The headings describe the type of data in the columns under them. Therefore, the headings have higher concept hierarchical level and they are the

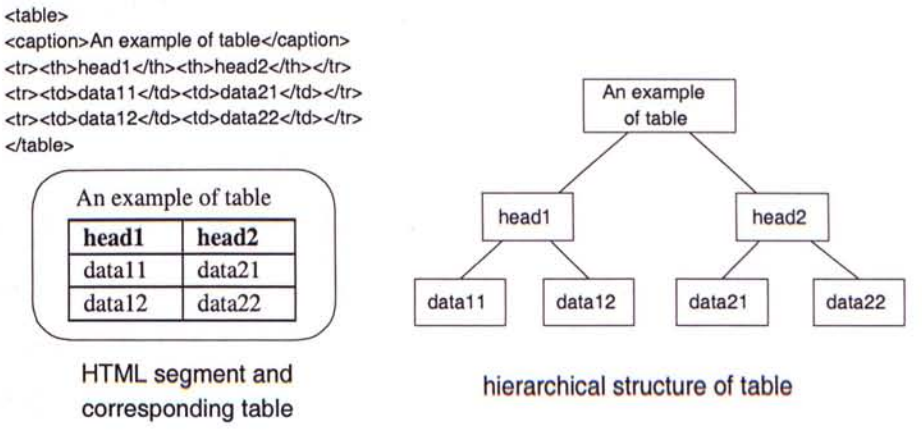


Figure 4.11: Hierarchical structure of table with header entry

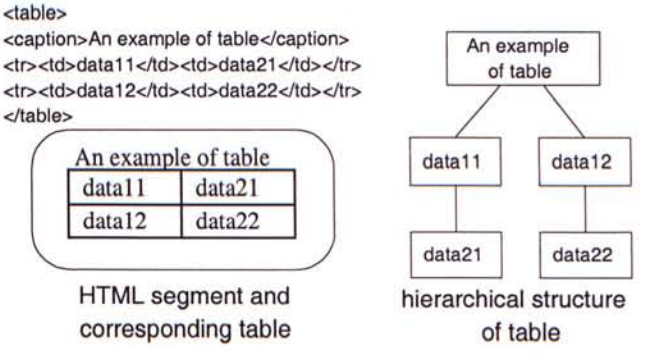


Figure 4.12: Hierarchical structure of table without header entry

parent of the data in the corresponding columns. An example is shown in Figure 4.11. Therefore the header entry has higher concept hierarchical data in the table.

If there is no header entry, then we could consider the leftmost entry of each row in the table having higher concept hierarchical level than other entries in the same row. Usually, in the web pages, the first entry of each row would be used as the header for each row if there is no header entry in the table. Each row would be considered as the same level in the hierarchical structure. An example is shown in Figure 4.12. The leftmost entry of a row is the parent of the other entries in the same row.

Heuristic 5: Rigid structure tags and loose structure tags

This heuristic is used to relate the text data from rigid structure tags and loose structure tags. There are two types of loose structure tags: heading and non-heading. They relate to the rigid structure tags in different ways. As stated in Heuristic 1, text data in the title tag describes the topic of the data stored in the web page, so it is at the highest hierarchical level among all data enclosed by rigid structure tags and loose structure tags.

In addition, the heading tags are often used to divide the web pages into several regions and the text data enclosed by the heading tags are used to describe the text data in the corresponding region. All the data, including those in loose and rigid structure tags in the region has lower concept hierarchical level than the text data of the heading tag above the region.

For the non-heading loose structure tags, if the text data is just above the rigid structure tags, then it acts as a heading that describes the topic of the data of rigid structure tags. Therefore it has higher concept hierarchical level than the data in the rigid structure tags. That is, it is the parent in the hierarchical structure.

Using Tag-tree to construct Hierarchical Structure

The tag-tree could be used to construct the hierarchical structure more efficiently. There are several subtrees in the tag-tree. For example, the left of Figure 4.13 is a subtree of the tag-tree in Figure 4.5. The root of the subtree is "TABLE" tag so it is called "TABLE" subtree. It is also a subtree of the "BODY" subtree. According to the five heuristics, structure is extracted from each subtree. The right of Figure 4.13 is the structure extracted from the "TABLE" subtree by Heuristic 4. The root node of the structure indicate that it is extracted from

a "TABLE" subtree. The right of Figure 4.14 is the structure extracted from the "UL" subtree by Heuristic 3. As there is only one text data under the "UL" subtree, the structure consists of only one node. Structure is then extracted from each subtree and the root node of the structure shows the HTML tag from where the structure is extracted.

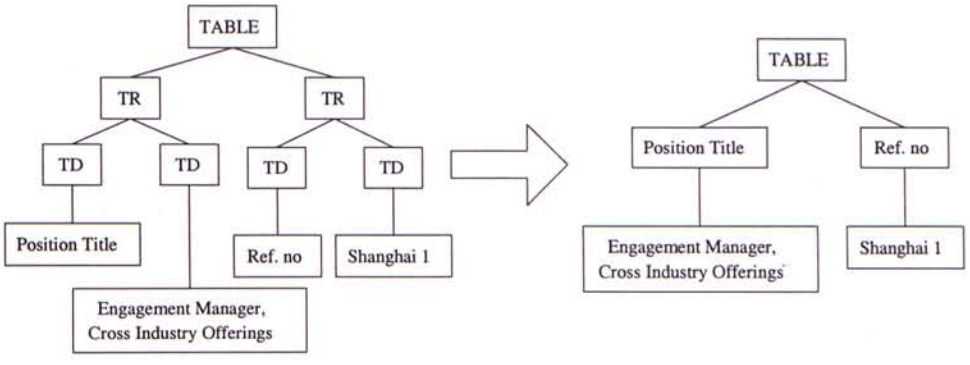


Figure 4.13: The structure extracted from a "TABLE" subtree

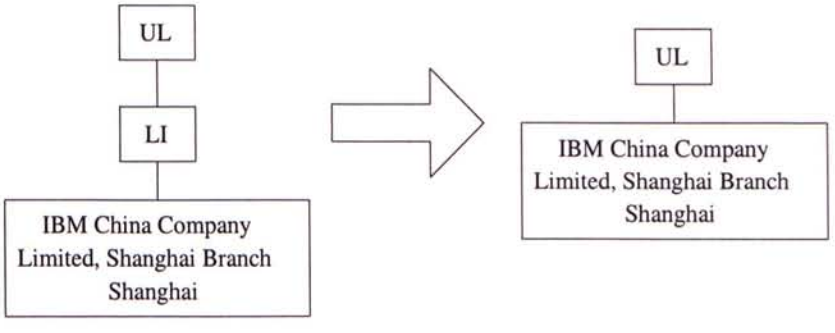


Figure 4.14: The structure extracted from a "UL" subtree

If two subtrees are under the same node in the tag-tree then the structure extracted from these two subtrees are merged to a new structure by the five heuristics. For example, in Figure 4.5, "TABLE" subtree and "P" subtree are under the same node "BODY" so the structures extracted from them are merged according to the five heuristics. Figure 4.15 shows the process of merging the structures from three subtrees. We assume that the "BODY" node only has these three subtrees. By Heuristic 5, the structure from "P" subtree is the parent of that from "UL" subtree and is the right sibling of that from "TABLE" subtree.

Now the new structure is rooted with "BODY" node. Finally the structure from "TITLE" subtree is in the highest level in the hierarchical structure so the structure of "BODY" subtree is the children. The hierarchical structure is constructed as shown in Figure 4.7.

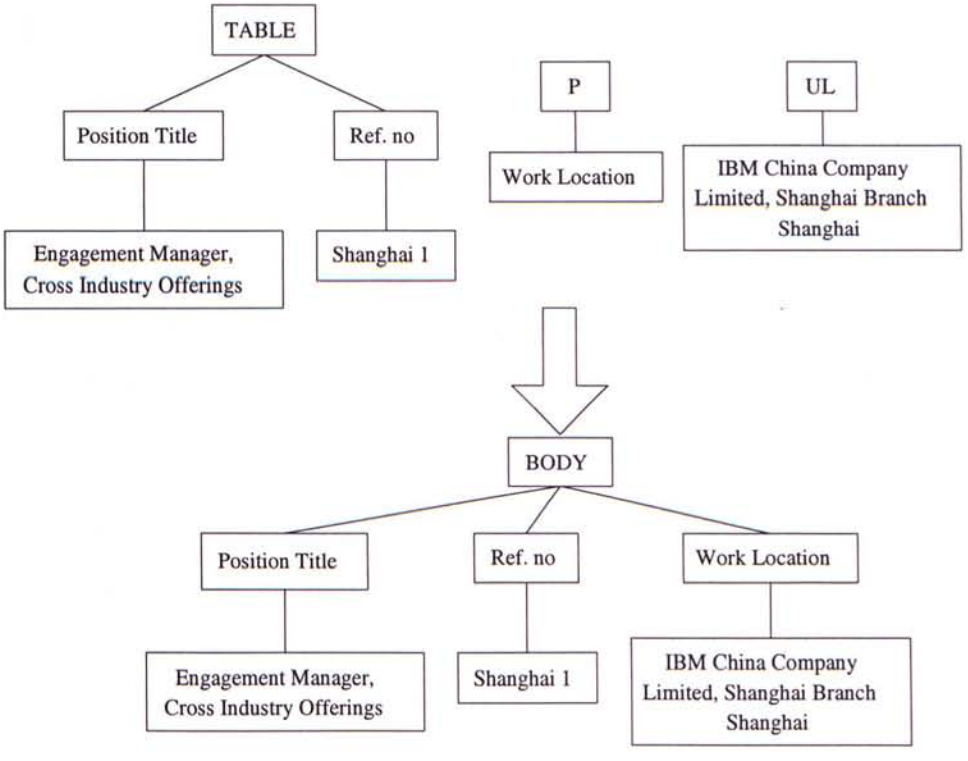


Figure 4.15: The structure of "BODY" subtree

4.4 Hierarchical Structure Statistics

Hierarchical structure organizes data in the order of their concept hierarchical level in the web pages. So, we would like to determine whether this rule is obeyed in a set of web pages. One of the evaluation of the performance of hierarchical structure is measured by studying the position of labels and their corresponding values in the structure. Labels have higher concept hierarchical level than their values, so they should be the parent of their values in the hierarchical structure.

Therefore, we would like to determine this relation by studying a class of web pages. Labels and values could be distinguished manually if the set of web pages is in the same class.

We extract a set of hierarchical structure from 7651 web pages in the same class. In the set of structure, there are total 158,285 nodes in the 7651 hierarchical structures. The labels and values are determined manually from the structure according to the attributes of the class. The class we used is the job employment class and we know that some attributes should be in the class, e.g. job title, job requirements, job description, work location, etc. According to these pre-defined attributes, we could determine which text data should be labels and represent which attributes in the web pages. Also, based on the attributes represent by the text data, we could determine which one is the value of the corresponding label.

In the set of nodes, we find that there are 28,527 nodes containing text data that should be labels and there are 49,583 nodes containing text data that should be values. We could observe that some labels would have more than one text data as values. For example, the requirements of a job would have 3 values which represent 3 different required skills of the job and the values are enclosed by different tags. So, for one label, the value may be located in different neighbour nodes. Therefore, we discover the location of each value corresponding to their labels in the hierarchical structure.

In Figure 4.16, the blackened node is the label and then the possible position of its values would be parent, left sibling, right sibling and children. In Table 4.2, the distribution of the location of values relative to their corresponding labels is shown. In the set of web pages, we observe that almost all the values are located at the children node of their labels. However, there are some located at the right sibling. No values are located at the parent and left sibling nodes. In the table, others means the positions other than the four neighbour nodes and

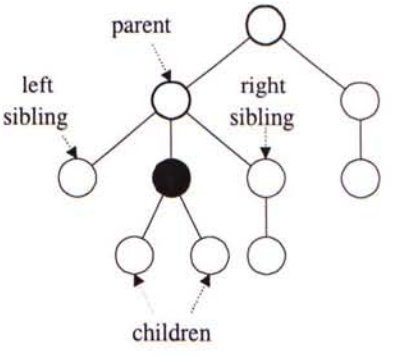


Figure 4.16: The possible neighbour of a node

Location of values	Number of values
Parent	0.00%
Left sibling	0.00%
Right sibling	6.09%
Children	93.20%
Others	0.71%

Table 4.2: Location of values statistics

the node itself. Very few values are located at other positions. That means only few cannot be organized in the order. The hierarchical structure could organize almost all labels and values in the concept hierarchical order.

Chapter 5

Similar Labels Discovery

Web pages in the same class store the same type of information and each class has a number of attributes as stated in definition 1. The attributes are stored as labels with their values in each web pages. However, the labels may be different for the same attributes in a class of web pages: for two labels l_1 and l_2, $A(l_1) = A(l_2)$ but $l_1 \neq l_2$. We said that these two labels are similar $l_1 \sim l_2$ as defined in definition 6. If similar labels could be discovered then we could know that whether two values belong to the same attributes.

To deal with the similar labels problem, we introduce **labels discovery algorithm** based on the hierarchical structure to reveal similar labels. Before looking into our algorithm we would like to introduce the **structure-expression** of the hierarchical structure extracted from a web page. This expression could make the algorithm more efficiently than using a tree structure.

5.1 Expression of Hierarchical Structure

The hierarchical structure is a tree in which each node corresponds to one text data in a web page. Each node has four possible neighbour nodes: parent, left sibling, right sibling and children. Information of a node could be obtained from

its neighbour nodes. For example, values are always in the child nodes of their labels as shown in previous chapter. Therefore, we would like to express a node associated with the data in its neighbour nodes. We then express each node as a structure-expression with the following format.

The structure-expression of a node n in a hierarchical structure is associated with five data sets $\{d_o, d_p, d_l, d_r, d_c\}$. They are the sets of text data from the node itself d_o, from the parent d_p, left sibling d_l, right sibling d_r and child d_c nodes. They are called own data set, parent data set, left sibling data set, right sibling data set and children data set respectively. These notations are summarized in Table 5.1.

Symbols	Data sets	Descriptions
d_o	own data set	text data from the node itself
d_p	parent data set	text data from the parent node
d_l	left sibling data set	text data from all left sibling nodes
d_r	right sibling data set	text data from all right sibling nodes
d_c	children data set	text data from all child nodes

Table 5.1: Notation of data sets

The left sibling, right sibling and children data sets may contain text data from more than one node. Some of these data sets except own data set may be empty as some nodes may not have parent, left sibling, etc. Recall that text data is the textual content from the web pages, so each text data consists of a set of words. Therefore the content of each data set is a set of distinct words.

Definition 12 (Structure-expression of a node) A node n in the hierarchical structure is expressed as $\{d_o, d_p, d_l, d_r, d_c\}$. Each data set d_i consists of a set of words $\{w_1, w_2, ..., w_n\}$ from the text data of the corresponding nodes. ∎

Example 5 In Figure 5.1 is an example of hierarchical structure. The node "*Position Title*" should be expressed with the five data sets as follows:

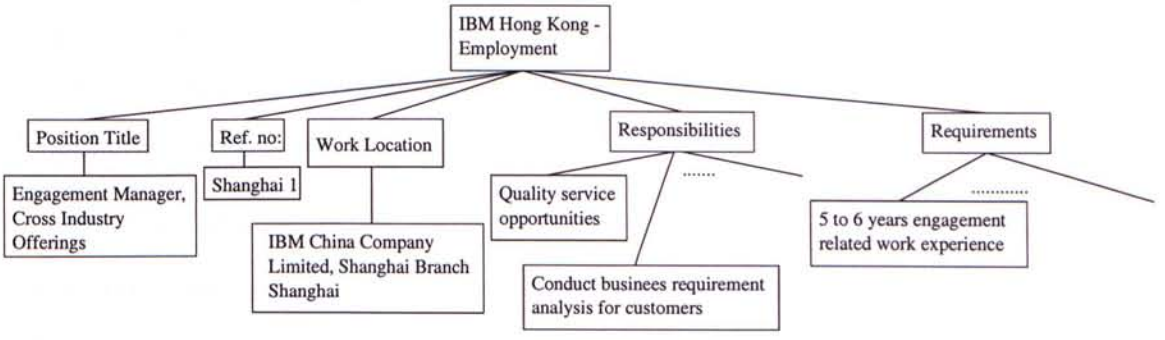


Figure 5.1: A hierarchical structure of web page

d_o = (Position, Title)

d_p = (IBM, Hong, Kong, Employment)

$d_l = \phi$

d_r = (Ref, no, Work, Location, Responsibilities, Requirements)

d_c = (Engagement, Manager, Cross, Industry)

"*Position Title*" does not have left sibling in the hierarchical structure, so the left sibling data set is empty, denoted by ϕ. ∎

Each node in the hierarchical structure is expressed in a structure-expression. Then a hierarchical structure consists of a set of nodes $\{n_1, n_2, ...\}$ where each node has five data sets. This expression could let us retrieve the surrounding data of a node more efficiently.

5.2 Labels Discovery Algorithm

Labels always are stored with their values in the web pages. The labels represent attributes in the class of web pages. They are used to describe the topic of their values in a web page. Some values of the same attribute have similar characteristics. That is, the values of similar labels may contain some common words. For example, "*requirements*" and "*qualifications*" represent the same attribute and the word "required" occur in both values of these labels. Therefore,

by comparing the values of each labels, we could determine the similarity between them and then discover similar labels. Labels discovery algorithm bases on this characteristic to reveal similar labels.

In a set of hierarchical structure extracted from a class of web pages, there are nodes containing labels, values or others, there are two categories of nodes: **label node** and **non-label node** in the structure. They are defined in the follows.

Definition 13 (Label node) Label node contains text data in its own data set such that the text data is used as label in the web page. ∎

Definition 14 (Non-label node) Non-label node contains text data in its own data set such that the text data is not label but may be value or others in the web page. ∎

We now have problems on labels. If we do not know which nodes contain labels, how can we discover similar labels? Furthermore, in oder to discover similar labels, only label nodes are useful. Non-label nodes are useless and acts as noise in similar labels discovery. Therefore we have to eliminate non-label nodes in each hierarchical structure. By using the properties of labels, we then set some rules to eliminate non-label nodes from the hierarchical structure.

Labels discovery algorithm then consists of three phases to discover similar labels from the set of hierarchical structures of web pages in the same class. Phase one removes non-label nodes from the set of hierarchical structures of web pages. Label nodes are then identified from the remaining nodes in phase two. Lastly, phase three discover similar labels by measuring the similarity of the label nodes.

5.2.1 Phase 1: Remove Non-label Nodes

In phase 1 of labels discovery algorithm, non-label nodes are revealed and eliminated from the hierarchical structures of a class of web pages. These non-label nodes would not be used in the similar labels discovery and they may cause error to the result. In order to remove non-label nodes, we will then illustrate the determination of non-label nodes. If non-label nodes could be determined then the removal process would be easy. In fact, it is not the case. Here we will illustrate the problem and the solution.

In the web pages, instead of the data of the main type of information, there are other minor information. For example, in Figure 5.2, there are two job employment web pages from the same company and each one stores data of one job opportunity. In the web pages, instead of the job employment data there is the information of the application methods, "*Interested in applying for this job .....*" which is not a data of the job stored in the web page. Moreover, data of advertisements or company's information are both not the main data of the job class of web pages. Main data of a web page means the data of the main type of information in the web page. For example, the main data of job employment web pages is the data of a job, including position, requirements, description of the job, ..., etc. The main data of books web pages is the data of a book, including title, authors, publishers, ..., etc. Then the others are not the main data of the class such as advertisement in a book web pages, company information of a job employment web pages. In the construction of hierarchical structure, this information will become several non-label nodes in the structure.

However, this type of non-label nodes is not easily discovered as they may appear in different place in the web page and different format. For example, the advertisements may be located at the top in some web pages but at the bottom in other web pages. In addition, there are different types of information that is

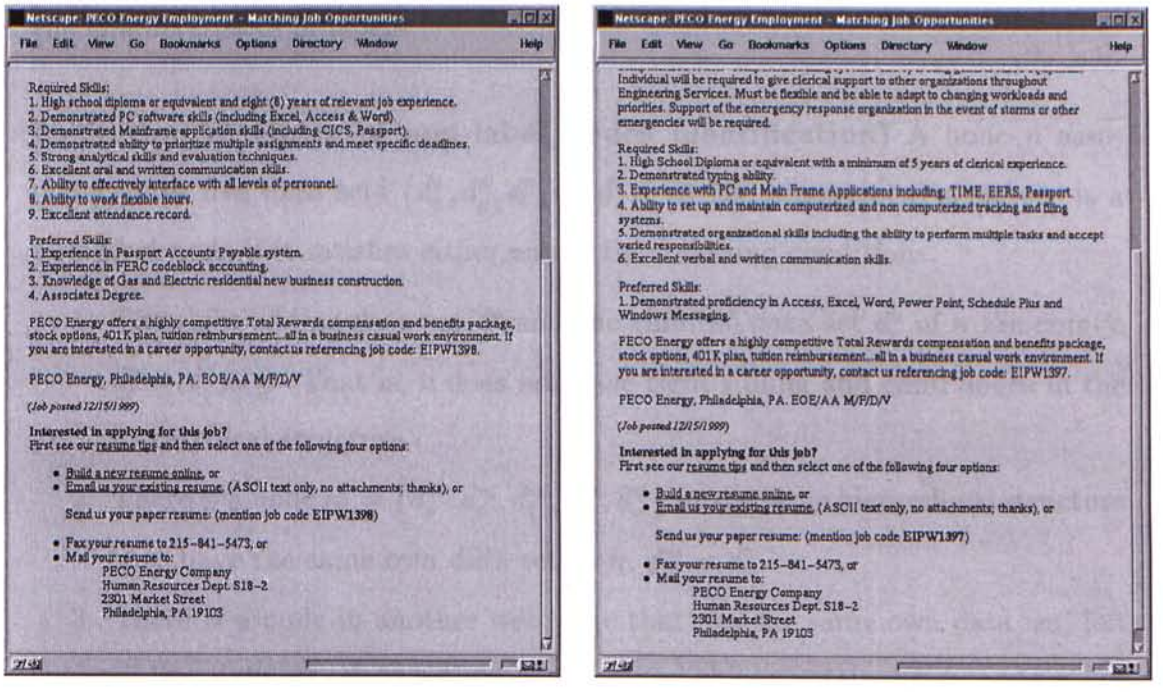


Figure 5.2: Two job employment web pages from one company

not main data in the web pages. Therefore, there is no specific characteristics of this data. It is difficult to determine their existence in the web pages.

Non-label nodes also contain values in the web page. Among the nodes in the hierarchical structure, the number of non-label nodes are more than label nodes as each label may have more than one text data as its value in the web page. To identify which node contains values is difficult because the textual content of values may be long or short and are different in different web pages. For example, the value of a job's title may be short as "*IT Specialist*" but the values of a job's description would be long. Therefore it is not easy to determine them by their content.

It is difficult to determine the characteristics of non-label nodes but we could determine them by using the properties of labels. If the data in the node does not obey the properties of labels, then we could determine that the node is non-label node. Therefore we define the rules of identify non-label nodes by using

the characteristics of labels.

Definition 15 (Rule of non-label nodes identification) A node n associated with five data sets $\{d_o^n, d_p^n, d_l^n, d_r^n, d_c^n\}$[1] in the hierarchical structure is a non-label node if it satisfies either one of the following conditions.

1. The right sibling data set d_r^n and the children data set d_c^n of n are empty, $d_r^n = d_c^n = \phi$. That is, n does not have right sibling and child nodes in the hierarchical structure.
2. There is a node $m = \{d_o^m, d_p^m, d_l^m, d_r^m, d_c^m\}$ in the same hierarchical structure that have the same own data set as n, $d_o^m = d_o^n$.
3. There is a node in another web page that has the same own data set, left sibling data set, right sibling data set and children data set as node n. ∎

We would then give the arguments and examples of the three rules of non-label nodes identification.

Rule 1 Each label in a web page has its corresponding value. From the statistics of hierarchical structure in Chapter 4, the values are located at the right sibling nodes and mainly at child nodes of their corresponding labels in the structure. Therefore, if a node n does not have right sibling nodes and children nodes, then the text data in n is not a label. Then we considered n as non-label node.

In Figure 5.3, there are four cases of node that would be considered in Rule 1. Case 1 has right sibling and child nodes, then n is a label node. Case 2 has no right sibling nodes but two child nodes, then n is also a label node. Case 3 has one right sibling node but no child node, then n is also a label node. Case 4 has no right sibling and child nodes, then n is a non-label node. Therefore, in these four cases, n is considered as non-label node in case 4. ∎

[1] d_o^n denotes the own data set of node n where o for own data set, p for parent data set, l for left sibling data set, r for right sibling data set, c for children data set. These notations will be used in the remaining of thesis

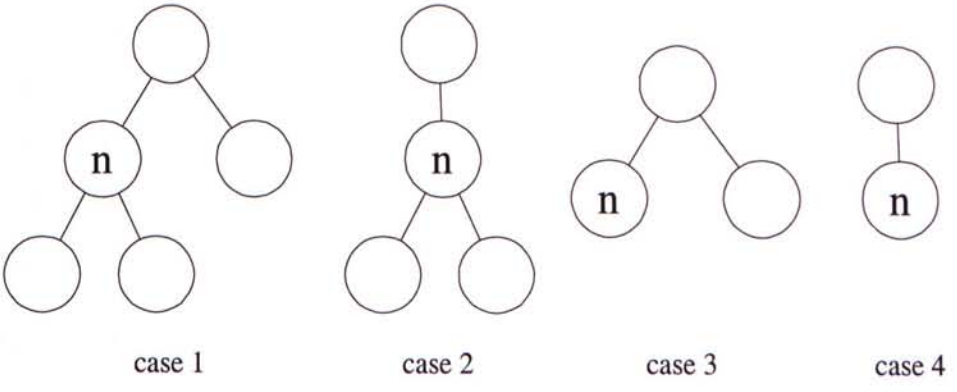


Figure 5.3: Four cases of a node in a hierarchical structure in Rule 1 of non-label nodes identification

Rule 2 In Property 1, in one web page, all labels are different and each label occurs once. A label consists of one text data in the web page. As labels are unique within one web page, if a text data t is label, then it will not be located in another position of the web page. Therefore t will not be in two nodes of hierarchical structure. Reminded that content of own data set of a node n is from the text data of n itself. For a web page w, suppose that the node n is a label node in w, then $\forall n_i \in w$, $d_o^n \neq d_o^{n_i}$ if $n \neq n_i$. Therefore, if the own data set of two nodes are equal, then they are non-label nodes. ∎

Rule 3 Very few web pages in the same class would have the same label-value pair. The reason is that different authors will use their own style to represent the same data and it is rare that two authors will use the same wordings to represent the same thing. For example, in most cases, the name of a job will be different for two different companies even the two jobs are working on the same field. However, there are cases that the same label-value pair are stored in two different web pages. For example, the authors of a book in books web pages may be stored in another page as the authors may publish many books.

However, the text data next to these label-value pairs would always be different in different web pages. For example, there may be book title next to the

authors in some pages but in others there may be publisher next to the authors. Therefore, if we look at the neighbour of a label in the hierarchical structure then we could discover that they would be different in almost all pages in the same class. Although there may be some cases that they will all the same, the number become much fewer and they could be ignored.

As a result, if the neighbour and the node itself occur exactly the same pattern in another page, then we could determine them as non-label nodes. To determine a non-label node, the own data set, left sibling data set, right sibling data set and children data set of a node should be the same as another node in another web page. We omit the parent data set because it is too strict to the condition of having all the neighbour to be the same. This will only identify few non-label nodes. In order to identify more, we loose the condition to omit the parent data set as this omission will discover more non-label nodes and the errors are very small as shown in our experiments. ∎

From a class of web pages, non-label nodes in hierarchical structure are identified by using the above rules. Then they are removed from the hierarchical structure. The remaining nodes would have a higher possibility to be label nodes as it is difficult to identify all non-labels. Some non-label nodes may be remained but the number will be greatly reduced.

5.2.2 Phase 2: Identify Label Nodes

After phase 1, the number of non-label nodes is largely reduced but there are still some remained in each web page. However, the proportion of label nodes is increased and the number is more than non-label nodes. Therefore, in this phase, we will discover **candidate label nodes** that have the higher possibilities to be label nodes.

Although different labels are used to represent the same attributes in different

pages, there are some labels that many web pages in the same class would use. That is, there are identical labels in different web pages. Therefore in the set of web pages, there are nodes that have the same own data set.

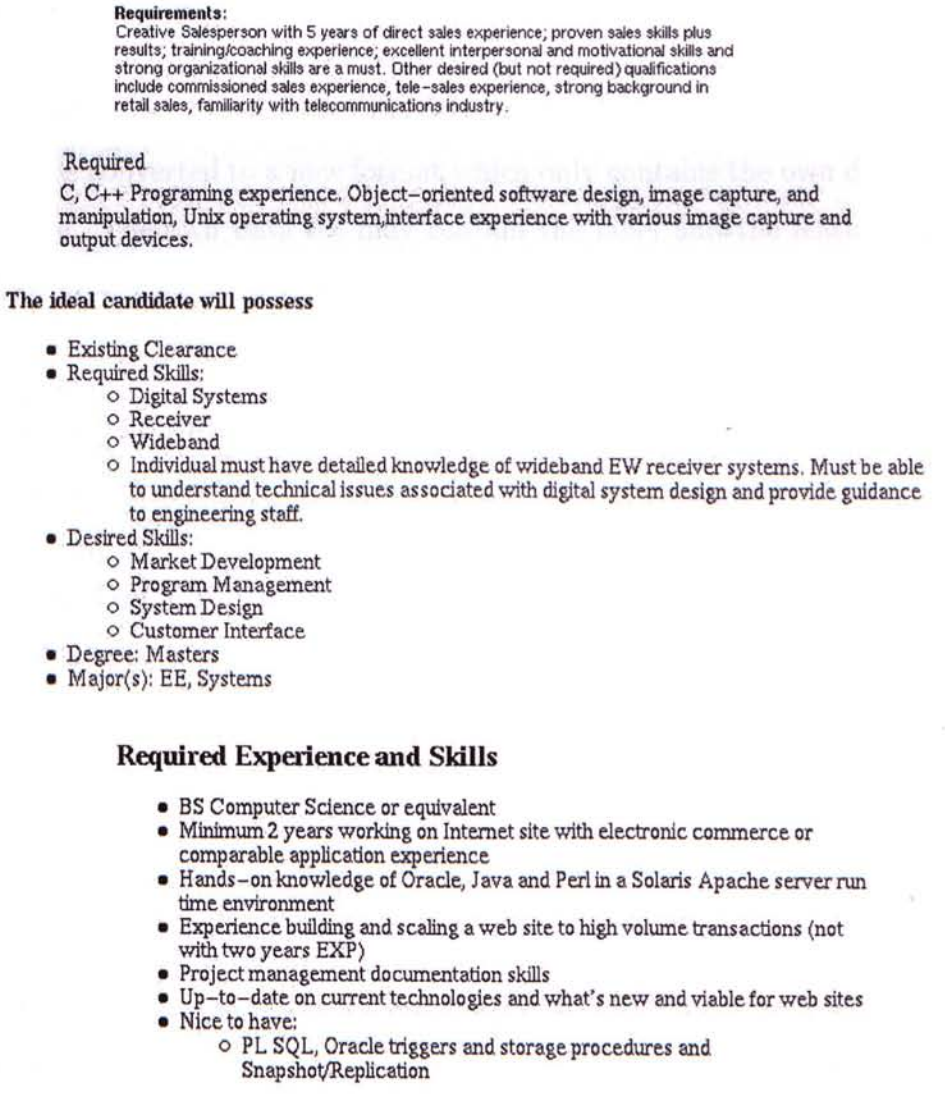
Furthermore, it is rare that the non-label node with the same own data set would be in many web pages. The data in the non-label node are values and other type of information instead of the main type of the class. For values, they would not be the same text data in many web pages as data of an attributes would be different for different records. For example, the name of a job would not appear exactly the same in many web pages as there are many different jobs in a class of job employment web pages.

There are other information instead of the main information of the class stored in the web pages. Different web pages may have different type of this information. As this is not the main information, the same data of one type of this information would not be stored in many web pages of the class. Therefore, for these types of non-label nodes, very few have the same own data set in many web pages.

According to these characteristics, we could discover candidate label nodes by the frequency of the nodes in the set of web pages. The frequency is counted by the number of nodes with the same own data set. Each node is distinguished by their own data sets. For example, if there is another node with the same own data set of node n then the frequency of n is 2. We then count the frequency of each of such nodes in the set of nodes remained in phase 1.

For the same attribute, the values would be similar in their content. Some words always appear in different values of the same attribute. For example, in Figure 5.4, there are four web pages segment which each displays the label and values of the attribute job requirement. In four values, we could discover that some words always appear, e.g. experience, skills, etc. Therefore for the values

of different labels, there are different characteristics for different labels. As a result, the values could be used to discover similar labels.



Requirements:
Creative Salesperson with 5 years of direct sales experience; proven sales skills plus results; training/coaching experience; excellent interpersonal and motivational skills and strong organizational skills are a must. Other desired (but not required) qualifications include commissioned sales experience, tele-sales experience, strong background in retail sales, familiarity with telecommunications industry.

Required
C, C++ Programing experience. Object-oriented software design, image capture, and manipulation, Unix operating system,interface experience with various image capture and output devices.

The ideal candidate will possess

- Existing Clearance
- Required Skills:
  - Digital Systems
  - Receiver
  - Wideband
  - Individual must have detailed knowledge of wideband EW receiver systems. Must be able to understand technical issues associated with digital system design and provide guidance to engineering staff.
- Desired Skills:
  - Market Development
  - Program Management
  - System Design
  - Customer Interface
- Degree: Masters
- Major(s): EE, Systems

Required Experience and Skills

- BS Computer Science or equivalent
- Minimum 2 years working on Internet site with electronic commerce or comparable application experience
- Hands-on knowledge of Oracle, Java and Perl in a Solaris Apache server run time environment
- Experience building and scaling a web site to high volume transactions (not with two years EXP)
- Project management documentation skills
- Up-to-date on current technologies and what's new and viable for web sites
- Nice to have:
  - PL SQL, Oracle triggers and storage procedures and Snapshot/Replication

Figure 5.4: Four web page segments

In most case, the values are located at the child nodes of their corresponding labels. Some also are located at the right sibling nodes. Therefore to extract the content of values, we consider both child nodes and right sibling nodes. Although other data will also be included, we could also include all the data of value and the other data will not affect characteristics of values. Then all the data of these two nodes are combined to form the **feature** of the node.

Definition 16 The union of the child and right sibling data sets of a node n, $d_r^n \cup d_c^n$ are called the **feature** of n, denoted by f_n. The feature is a set of distinct words $\{w_1, w_2, ..., w_n\}$. ∎

The feature for every same own data set nodes are generated. These nodes will then be converted to a new format which only contains the own data set and the feature. The own data set may contain the label and the feature contains the value of the label. The content of a node n is now a set of two data sets $\{d_o^n, f_n\}$.

Then the **support** of these nodes is calculated. The support of each node is defined as follow.

Definition 17 (Support) In a set of N web pages in the same class, the frequency of a node n with the same own data set is $F(n)$. Then the **support** of n, $Sup(n)$, is defined as follow.

$$Sup(n) = \frac{F(n)}{N} \tag{5.1}$$

∎

Nodes with same own data set are merged by union their feature to produce a new node. For k nodes $n_1, n_2, ..., n_k$ with $d_o^{n_1} = d_o^{n_2} = ... = d_o^{n_k}$, they are merged to form a new node $m = \{d_o^m, f_m\}$ such that $d_o^m = d_o^{n_1} = d_o^{n_2} = ... = d_o^{n_k}$ and $f_m = f_{n_1} \cup f_{n_2} \cup ... \cup f_{n_k}$. Then there would be a new set of nodes produced $M = \{m_1, m_2, ..., m_l\}$ where the own data set of each node is distinct. Each node in M is associated with its support value. In addition, each word in the feature has a count. The count of a word w_i is the number of nodes with the same own data set containing w_i in the feature.

With a high support, a node has a high possibility to be a label node. Therefore, there is a pre-defined threshold **min_support** such that if the support of a

node is greater than the threshold then it is **candidate label node**. Otherwise, it is **non-candidate label node**. The own data set of candidate label node contains label. In contrast, the own data set of non-candidate label node has a little chance to contain label. That is, few nodes in the non-candidate label nodes would contain a label.

Definition 18 (Candidate label node) If the support of a node m is greater than the min_support threshold ε_{sup}, $Sup(m) > \varepsilon_{sup}$ then m is a candidate label node, otherwise it is non-candidate label node. The own data set of this node is said to contain label. ∎

Furthermore, the words in the feature of a candidate or non-candidate label node m are the characteristics of m. In traditional information retrieval, each document is represented by an n-dimensional vector, where n is the number of distinct keywords or terms in the collection of documents. The vector is the knowledge of the document. It could be used for query, classification. Likewise, the feature of a node is the knowledge of the node. It represents the data in own data set of the node. Hence, each word in the feature is a characteristic of the feature. There is a value called **confidence** indicating the significance of a word in the feature of a node. The confidence is calculated by the count of each word in the feature.

Definition 19 (Confidence) Suppose a candidate or non-candidate label node m is formed by merging a set of $F(m)$ nodes, N, with the same own data set and a word w_i occurs in the feature of $freq_m(w_i)$ nodes in N. Then the confidence of a word w_i in the feature of a candidate or non-candidate label node m, $Conf_m(w_i)$, is defined as follow.

$$Conf_m(w_i) = \frac{freq_m(w_i)}{F(m)} \tag{5.2}$$

∎

Example 6 Given 100 nodes with the same own data set, they are merged to form a new node m. Then the frequency of m, $F(m) = 100$. The feature of m is union by all feature of 100 nodes. If a word w_i occurs in the feature of 50 nodes that formed m, then the frequency of w_i in node m, $freq_m(w_i) = 50$. So the confidence of w_i in the feature of node m is calculated as

$$Conf_m(w_i) = \frac{50}{100} = 0.5$$

■

In short, here is the summary of phase 2 algorithm.

1. **Discover nodes with same own data set.** From a set of nodes $N = \{n_1, n_2, ..., n_k\}$, discover nodes with same own data set and count their frequency in the set. Support of each node is calculated.

2. **Generate feature for each node.** A feature f_{n_i} is generated by union the right sibling and children data sets for each node $n_i \in N$. Then the node in N is converted to a set of two data sets, $n_i = \{d_o^{n_i}, f_{n_i}\}$.

3. **Identify candidate label nodes.** The set of nodes N is transformed to $M = \{m_1, m_2, ..., m_l\}$ by merging nodes in N with the same own data set to a node m_i and all feature of the nodes with same own data set are union to a new feature f_{m_i}. Each node in M is associated with a support. If $m_i \in M$ and $Sup(m_i) > \varepsilon_{sup}$ then the node m_i is candidate label node, otherwise, m_i is non-candidate label node.

5.2.3 Phase 3: Discover Similar Labels

From the candidate and non-candidate label nodes obtained in phase 2, phase 3 discovers nodes containing similar labels. Therefore we have to measure the similarity of the nodes. The similarity is calculated by the feature of each node. In

this section, we will first introduce the similarity function used in the algorithm. Afterwards the algorithm of phase 3 will be introduced to discover similar labels.

Similarity Function

In the traditional information retrieval, each document is represented by an n-dimensional vector, where n is the number of distinct keywords or terms in the collection of documents [25]. The vector is the knowledge of the document. It could be used for query and classification. Similarly, a query is also represented by an n-dimensional vector. The similarity between a query and a document is measured by the closeness of the corresponding vectors in the n-dimensional space. Likewise, classification also find the closeness of the vectors of two documents in the n-dimensional space.

Each entry in the document vector corresponds to a word in the collection of documents. Basically, the simplest format of a document vector is binary. That is, each component of a vector is either 0 or 1 (where 0 and 1 represent the absence and the presence, respectively, of a term in the document or query). Furthermore, the value in the vector of a word could be used to indicate the significance of the word in the document [27]. The value could be document frequency and term occurrence frequency. The document frequency of a term is the number of documents having the term. Usually, the more documents having the term, the less useful the term is in discriminating those documents having it from those not having it. In addition, the term occurrence frequency is the number of times the term occurs in the document. If a term occurs many times, then it is likely that the term is significant in representing the contents of the document because the author keeps on using it.

A similarity function is defined to measure the closeness between any two vectors. Let the two vectors be $X = (x_1, ..., x_i, ...x_n)$ and $Y = (y_1, ..., y_i, ..., y_n)$.

In information retrieval, Cosine function between two vectors is used frequently as the similarity function and it is given as follows.

$$Cosine(X,Y) = \frac{X \cdot Y}{\sqrt{(X \cdot X) \bullet (Y \cdot Y)}} \tag{5.3}$$

where $X \cdot Y$ is the dot product given below,

$$X \cdot Y = \sum_{i=1}^{n} x_i y_i \tag{5.4}$$

and the big dot ($\bullet$) denotes the familiar scalar multiplication. If the value of an entry in the vector x_i is 0 then the word w_i does not occur in the document. In the dot product, if the word w_i does not appear in either one of the document then the value $x_i y_i$ of w_i is 0. Therefore the dot product is greater than 0 only when there is common words between the two documents. Hence the similarity function measures the degree of words overlap between two documents. So the cosine function could be rewritten to indicate the degree of words overlap between two documents D_X and D_Y.

$$Overlap(D_X, D_Y) = \frac{\sum_{w_i \in D_X \wedge w_i \in D_Y} x_i y_i}{\sqrt{\sum_{w_i \in D_X \wedge w_i \in D_Y} x_i^2 \sum_{w_i \in D_X \wedge w_i \in D_Y} y_i^2}} \tag{5.5}$$

Similarly, in our algorithm, each node has its feature which each element is a word associated with a confidence. The feature is used to describe the own data set of each node. It indicates the meaning of the own data set of the node. Each entry in the feature is a characteristic of the feature. The confidence of a word indicates the significance of the word in the feature. The feature plays the same role as the vector of a document in the traditional information retrieval. The vector of a document indicates which words occur in the document. To find the similarity between the own data set of each node we could measure the similarity between the features of each own data set.

The similarity function between two features in our algorithm is based on the cosine function discussed above. In our algorithm, the feature could be used to

denote the attribute that may be represented by the label in the own data set of a node. For the same attribute, the data will be similar. That is, some words always occur in the data of the same attribute of different records. The feature in the node n contains the value of the corresponding label in the own data set of n. Therefore the features of two nodes n_1 and n_2 must have overlap on words if the own data set of n_1 and n_2 contains similar labels. The more two features overlap, the more they are similar. As a result, the similarity function between two features is defined according to the cosine function in traditional information retrieval.

Definition 20 (Similarity function of two features) There are two nodes n_1 and n_2 with the features f_{n_1} and f_{n_2} respectively. In the features, there are words w_i associated with confidence. Then the similarity function between the two features f_{n_1} and f_{n_2} is defined as follow.

$$similarity(f_{n_1}, f_{n_2}) = \frac{\sum_{w_i \in f_{n_1} \cap f_{n_2}} Conf_{n_1}(w_i) \times Conf_{n_2}(w_i)}{\sqrt{\sum_{w_i \in f_{n_1}} Conf_{n_1}(w_i)^2 \sum_{w_i \in f_{n_2}} Conf_{n_2}(w_i)^2}} \quad (5.6)$$

■

Example 7 There are two features with the confidence of the word in the bracket.

(apple{0.776}, orange{0.349}, banana{0.662}, mango{0.955})

(apple{0.446}, banana{0.965}, lemon{0.489})

Then the similarity value between these two feature is:

$$\frac{0.776 \times 0.446 + 0.662 \times 0.965}{\sqrt{(0.776^2 + 0.349^2 + 0.662^2 + 0.955^2) + (0.446^2 + 0.965^2 + 0.489^2)}} = 0.532$$

■

After we defined the similarity function between two features, we could define the similarity function between two nodes. From Property 1, the label is unique

within one web page. It means that the labels within one web page are not similar. That is, if a node once occurs with another node in the same web pages, then they are not similar at all. Therefore the similarity value between them is 0. Otherwise, the similarity function between two features of the nodes will be used to measure the similarity value of the nodes. Here is the formula used to calculate the similarity value between all the nodes.

Definition 21 (Similarity function of two nodes) There is a set of web pages $W = \{w_1, w_2, \ldots, w_n\}$. Suppose that there are two nodes n_1 and n_2 in the set of web pages with the features f_{n_1} and f_{n_2} respectively. The similarity function of n_1 and n_2 is defined as follow.

$$Sim(n_1, n_2) = \begin{cases} 0 & \text{if } \exists w_i \in W \text{ s.t. } n_1, n_2 \in w_i; \\ similarity(f_{n_1}, f_{n_2}) & \text{otherwise.} \end{cases} \tag{5.7}$$

■

Algorithm

After phase 2, there are two set of nodes: candidate label nodes N_c and non-candidate label nodes N_n. Candidate label nodes contain label in their own data set and non-candidate label nodes have a lower possibilities to contain label. In phase 3, similar nodes will be discovered from these two sets of nodes and then merged to form a new node. Then we could discover similar labels. The process is called **MergeSimilarNode**.

First the similarity values between all the candidate label nodes will be measured as these nodes are said to contain labels. In fact non-candidate label nodes may contain labels as well. Therefore, we need to discover these nodes from the non-candidate label nodes by measuring the similarity between the candidate label nodes and them. The reason is that if there is a label l in the non-candidate label nodes, then there may be a similar label of l in the candidate label nodes.

However, there is some labels in the non-candidate label nodes that have no similar labels in candidate label nodes. Then we could not discover these labels but these labels occur only in very few web pages as the support of the nodes containing them is small. There is a pre-defined threshold called **min_sim**, ε_s such that if $sim(n_1, n_2) < \varepsilon_s$, then n_1 and n_2 are said to be different.

Consequently, each node may have more than one node with similarity value greater than min_sim as there may be many nodes containing similar labels. We could model these as a graph to explain clearly. In the graph, each vertex is a node and each edge means the similarity value between the two nodes greater than min_sim. The weight of the edge is then the similarity value. In Figure 5.5, we shows one example of this graph. The value $Sim(n_2, n_6)$ on the edge between n_2 and n_6 is the similarity value of these two nodes. Later, we would use this graph to explain our process.

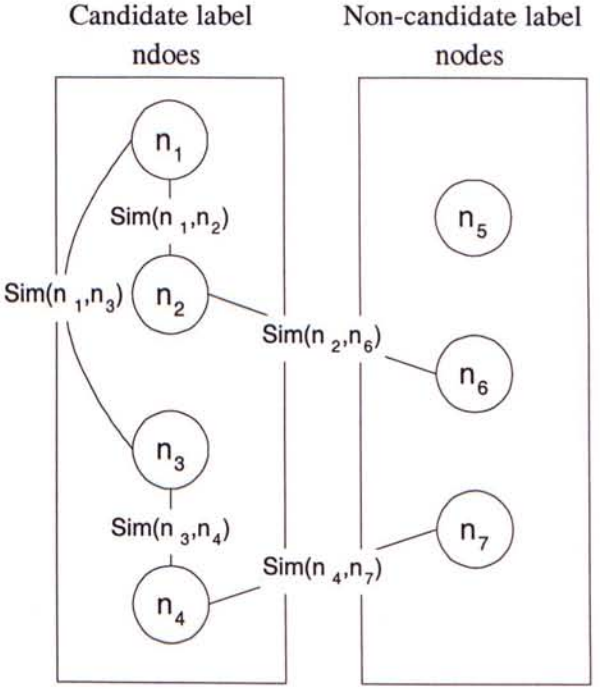


Figure 5.5: Graph representation of the relation of nodes

Two nodes n_1 and n_2 are similar if and only if the similarity value of them

$Sim(n_1, n_2)$ is the highest for both. That is, n_2 is the highest similarity value among other nodes to n_1 and vice versa. To explain, we could use the graph in Figure 5.5. Node n_1 has two possible similar nodes n_2 and n_3. We assume that $Sim(n_1, n_2) > Sim(n_1, n_3)$. Then for n_2, if $Sim(n_1, n_2) > Sim(n_2, n_6)$ then n_1 and n_2 are similar, otherwise they are not similar in this situation. In this phase, all these similar nodes pairs are discovered at the first step.

Afterwards, for each similar nodes pair, the two similar nodes are merged. The features of the two nodes are unioned. The confidence of each word in the feature is then updated. In addition, the new node will contain a set of own data sets which the elements come from the own data sets of the two similar nodes. The two similar nodes are then removed from the set where they belong.

The occurrence of either one of the similar nodes will be considered as an occurrence of the new node. Therefore the support of the new node is calculated by summing the support of the two similar nodes. This is illustrated as follow. If the frequency of two similar nodes n_1 and n_2 are $F(n_1)$ and $F(n_2)$ in N web pages then the frequency of the new node m is $F(m) = F(n_1) + F(n_2)$. Therefore the support of m is calculated below.

$$\begin{aligned} Sup(m) &= \frac{F(m)}{N} \\ &= \frac{F(n_1)}{N} + \frac{F(n_2)}{N} \\ &= Sup(n_1) + Sup(n_2) \end{aligned}$$

As one of the merged nodes is candidate label node, $Sup(m) > \varepsilon_{sup}$. The new node is considered as candidate label node. It is then added to the candidate label nodes set N_c.

After all similar node pairs are merged, this process is then repeated to the new set of nodes. The similarity values between the nodes will be calculated again as the feature of the new nodes are updated. We use the graph representation to illustrate this process. From Figure 5.5, if n_1 and n_2 are similar then they

merge to form a new node n_1n_2 as in Figurefig:graph1. Note that after two nodes are merged, their similarity values with other nodes may be changes as the new node's feature is changed. Sometimes the similarity value will be smaller than min_sim. As in Figure 5.6, n_1n_2 has no edge to n_3. Therefore each round we could

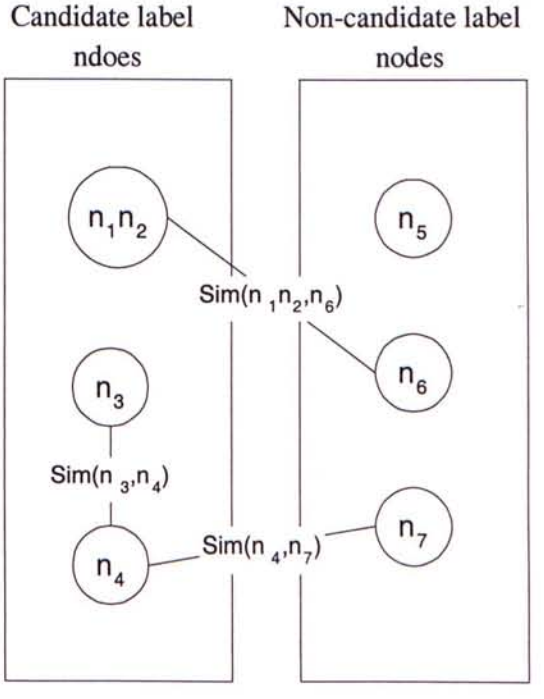


Figure 5.6: Graph representation of nodes after one round of MergeSimilarNode

discover different similar nodes according to the features of new nodes. As each time similar nodes are merged, the feature would become more representative to the node and the similar nodes could then have higher similarity value and dissimilar nodes could then have lower similarity value. The process is repeated until there is no similar node pairs are revealed. The process is shown briefly in Figure 5.7 and the notations used in the figure is summarized in Table 5.2.

At the end of phase 3, there are two sets of of nodes, candidate label nodes N_c and non-candidate label nodes N_n. Each node m_i in these sets consists of a set of own data sets $D_{m_i} = \{d_1^{m_i}, d_2^{m_i}, \ldots, d_k^{m_i}\}$ and a feature f_{m_i}, $m_i = \{D_{m_i}, f_{m_i}\}$. The candidate label nodes are considered to contain labels and the non-candidate label nodes contain non-labels. Therefore non-candidate label nodes could be

Symbols	Descriptions
N_c	candidate label nodes set
N_n	non-candidate label nodes set
ε_s	min_sim
$sim\_node(n_i)$	the node similar to n_i with the highest similarity
$max\_sim(n_i)$	the highest similarity value of n_i
$merge(n_i, n_j)$	merge the nodes n_i and n_j

Table 5.2: Notations in MergeSimilarNode process

removed. Candidate label nodes are the result of phase 3 finally.

In the set of own data sets in the candidate label nodes, each own data sets $d_k^{m_i}$ contain a label l_k. All the labels in D_{m_i} of a node m_i are similar. That is, each candidate label node m_i represents an attribute A_i of the class of web pages. From the set of own data sets D_{m_i}, we could obtain a set of similar labels $\{l_1^{m_i}, l_2^{m_i}, \ldots, l_k^{m_i}\}$ where $l_1^{m_i} \sim l_2^{m_i} \sim \ldots \sim l_k^{m_i}$ and they all represent an attribute A_i in the class $A(l_1^{m_i}) = A(l_2^{m_i}) = \ldots = A(l_k^{m_i}) = A_i$. The features f_{m_i} of each node are the characteristics of the corresponding attributes A_i. The features could be used to describe the set of similar labels in the same node. Then we replace the set of own data sets by the set of labels obtained from it. These are the results obtained from a class of web pages by labels discovery algorithm.

In short, the output of the labels discovery algorithm is described briefly as follows. From a class of web pages, a set of nodes are obtained.

$$M = \{m_1, m_2, \ldots, m_n\} \text{ with } m_i = \{L_{m_i}, f_{m_i}\}$$

where L_{m_i} is a set of labels

$$L_{m_i} = \{l_1^{m_i}, l_2^{m_i}, \ldots, l_k^{m_i}\} \text{ with } l_1^{m_i} \sim l_2^{m_i} \sim \ldots \sim l_k^{m_i}$$

and f_{m_i} is a feature consists of a set of words with their confidence.

Algorithm 5.1 MergeSimilarNode(N_c, N_n, ε_s)

```
1  Q ← φ;
2  for each nodes n_i ∈ N_c do
3      max_sim(n_i) ← 0;
4      sim_node(n_i) ← φ;
5      for each nodes n_j ∈ N_c and n_i ≠ n_j do
6          if Sim(n_i, n_j) > ε_s and Sim(n_i, n_j) > max_sim(n_i) then
7              max_sim(n_i) ← Sim(n_i, n_j);
8              sim_node(n_i) ← n_j;
9          end if
10     end for
11     for each nodes n_j ∈ N_n do
12         if Sim(n_i, n_j) > ε_s and Sim(n_i, n_j) > max_sim(n_i) then
13             max_sim(n_i) ← Sim(n_i, n_j);
14             sim_node(n_i) ← n_j;
15         end if
16         if Sim(n_i, n_j) > ε_s and Sim(n_i, n_j) > max_sim(n_j) then
17             max_sim(n_j) ← Sim(n_i, n_j);
18             sim_node(n_j) ← n_i;
19         end if
20     end for
21 end for
22 for each nodes n_i ∈ N_c do
23     n_j ← sim_node(n_i);
24     if sim_node(n_i) = sim_node(n_j) then
25         n_s ← merge(n_i, n_j);
26         add n_s to Q;
27         if n_j ∈ N_n then
28             remove n_j from N_n;
29         end if
30     else
31         add n_i to Q;
32     end if
33 end for
34 N_c ← Q;
```

Figure 5.7: MergeSimilarNode process

5.3 Performance Evaluation of Labels Discovery Algorithm

Experiments using web pages have been carried out. 4000 web pages are used. They are all job employment web pages from 34 different companies. Each web page contains one job employment. These companies are chosen arbitrarily from the company listings of yahoo search engine [17]. Then from these web pages we would show the results in different phases of labels discovery algorithm.

5.3.1 Phase 1 Results

Phase 1 of labels discovery algorithm removed non-label nodes from the hierarchical structure constructed from the web pages. Each web page consists of a set of nodes. From the 4000 hierarchical structures of the job employment web pages, there are 96,264 nodes. We determine 12,984 nodes containing labels manually. Therefore 13.49% nodes are label nodes. There are many non-label nodes. This has been improved substantially after phase 1.

After phase 1, there are 16,782 nodes remained. That is, 83.57% nodes are identified as non-label nodes and removed. This reduces the size of nodes dramatically. In the remaining nodes, there are 12,455 label nodes. The proportion of label nodes is increased from 13.49% to 74.22%. That means large amount of non-label nodes are eliminated in phase 1. 94.80% non-label nodes are identified correctly and eliminated. However, 529 label nodes are identified wrongly as non-label nodes and eliminated. These nodes are only a small proportion, 4.07%, of label nodes. Although few label nodes are removed, it is only a small proportion and similar labels could still be discovered. These removed labels could be discovered later by the results of the algorithm. In fact, the error of mis-identification of non-label nodes is very small, only 0.67%. So phase 1 could

	Before	**After**
Number of nodes	96,264	16,782
Number of label nodes	12,984	12,455
Proportion of label nodes	13.49%	74.22%
Proportion of non-label nodes	86.51%	25.78%

Table 5.3: Statistics of Phase 1

improve the quality of data to be processed.

In short, we could observe that phase 1 of labels discovery algorithm could largely reduce the number of non-label nodes in a set of hierarchical structures but retain almost all label nodes. The statistics are shown in Table 5.3. Non-label nodes are regarded as noise in the algorithm as they will decrease the accuracy of similar labels discovery. From the statistics of phase 1, noise is reduced greatly in this phase. Therefore we could say that the rule of non-label nodes identification defined in definition 15 can identify large amount of non-label nodes correctly.

5.3.2 Phase 2 Results

Phase 2 of labels discovery algorithm discovers candidate label nodes from the remained nodes after phase 1. Nodes with same own data set are merged. From the 16,782 nodes remained in phase 1, there are 163 nodes with distinct own data set. 76 nodes contain labels and 87 nodes contain non-labels. Their support is then evaluated according to their frequency in the set of web pages. As well, their features are formed. Stopwords are then removed from the features of each node and stemming is applied to the words in the feature.

Statistics on the support values and features of each node are shown in Table 5.4. The average support of nodes is 0.025739 which is very low. The low support means that each node occurs in few web pages only. This should be explained by the reason that many different labels are used to represent the same attributes. In addition, in Figure 5.8, we could observe the distribution of each node's support

average support	0.025739
average support of label nodes	0.041786
average support of non-label nodes	0.011721
average feature length	152.79
average feature length of label nodes	267.89
average feature length of non-label nodes	52.24

Table 5.4: Statistics of Phase 2

value. It shows that most of the nodes having support smaller than 0.1. However, there are some nodes having large support. Also, in the 163 nodes, over half of them are non-label nodes. These nodes have low support as the average support of non-label nodes is 0.011721. In contrast, label nodes have a higher support that the average support is 0.041786. The distribution of support values of label and non-label nodes is shown in Figure 5.9. We could observe that most label nodes have higher support values than that of non-label nodes. A large proportion of non-label nodes have very low support values. Therefore we could observe that non-label nodes always have small support. Accordingly, from the results, a node with a high support has a higher chance to be label node.

	$Sup(n) > \varepsilon_{sup}$		
ε_{sup}	**No of nodes**	**Label nodes**	**Label nodes included**
0.0418	28	71.43%	26.32%
0.0257	34	67.65%	30.26%
0.0117	39	66.67%	34.21%
0.0045	49	59.32%	46.05%
0.0018	80	52.50%	55.26%
0.0015	85	51.76%	57.89%

Table 5.5: Proportion of label nodes for different ε_{sup}

In Table 5.5, we shows the statistics of label nodes for different min_support ε_{sup}. For different threshold values, the number of nodes with support higher than the threshold (candidate label nodes), the proportion of label nodes in these nodes and the proportion of label nodes with support greater than the threshold are shown. For example, for $\varepsilon_{sup} = 0.0418$, there are 28 nodes with

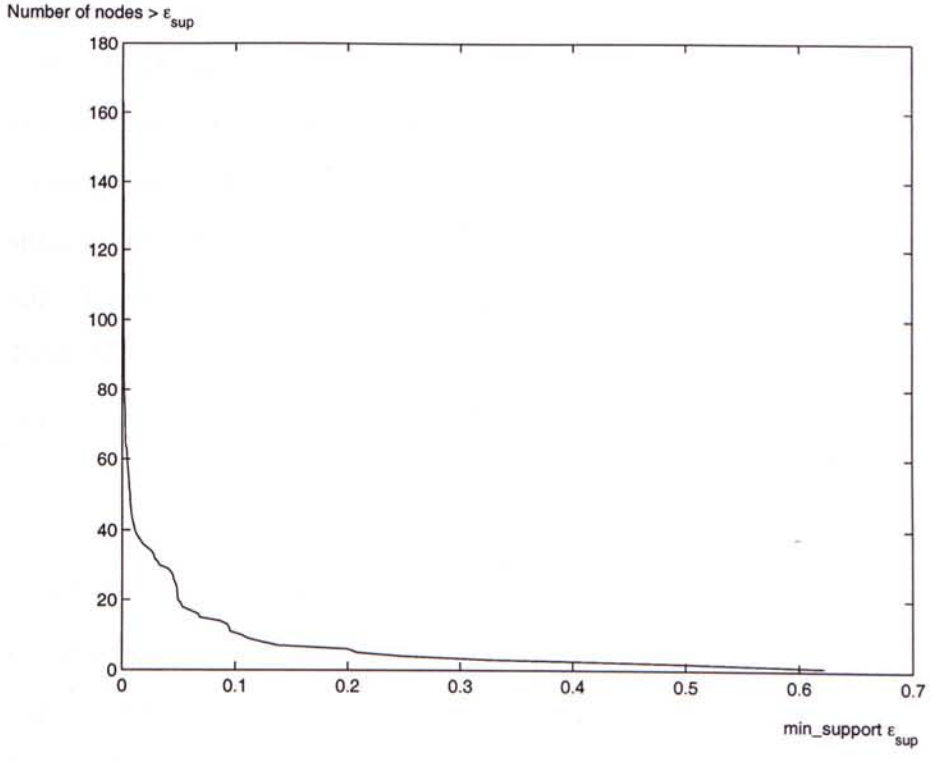


Figure 5.8: Distribution of support of nodes

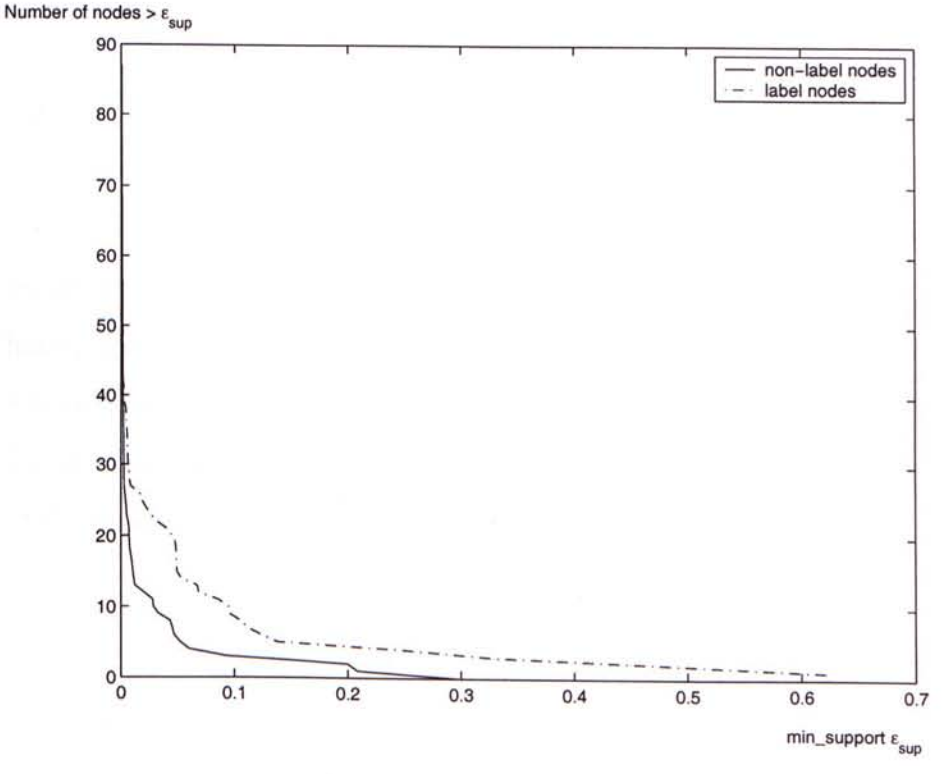


Figure 5.9: Distribution of support of label and non-label nodes

support > 0.0418 and in these 28 nodes, there are 71.43% label nodes. As well, there are 26.32% of all label nodes with support > 0.0418. As we do not have any methods to choose ε_{sup}, the threshold values are chosen from the median and mean support of all nodes, label nodes, as well as non-label nodes. The median support of all nodes, label nodes and non-label nodes are 0.0018, 0.0045 and 0.0015 respectively. The mean support for these three categories are shown in Table 5.4. It is obvious that the threshold chosen should make no non-label nodes become candidate label nodes. However, it is difficult to include no non-label nodes as candidate label nodes as fewer label nodes will be included as candidate label nodes. We could observe this from the Table 5.5. When the proportion of label nodes increase for different threshold, the label nodes included as the candidate label nodes become fewer. Therefore, we should choose the threshold such that more label nodes are included and more non-label nodes are excluded. However, it is difficult to discover a method for choosing such value, so we choose the value arbitrarily from the six values.

Another interest point of the results is that the average feature length of label nodes, which is 267.89 words, is greater than that of non-label nodes, which is 52.24 words. That is more words are in the feature of label nodes than non-label nodes. The reason is that values of labels are located in the feature of each label nodes and they provide information of the labels in the nodes. On the other hand, non-label nodes do not provide any information in the web page so their features are not representative of the non-labels in the nodes. Therefore the feature could then be used to determine similar labels as it provides information of the labels in the nodes.

5.3.3 Phase 3 Results

Phase 3 of labels discovery algorithm discovers similar labels and their features. There are two thresholds that affect the results of the algorithm. They are min_sup ε_{sup} and min_sim ε_s. ε_{sup} affects the number of label nodes regarded as candidate label nodes and the number of labels that could not be discovered. ε_s affects the result of similar labels discovery. These could be illustrated by the results of experiment.

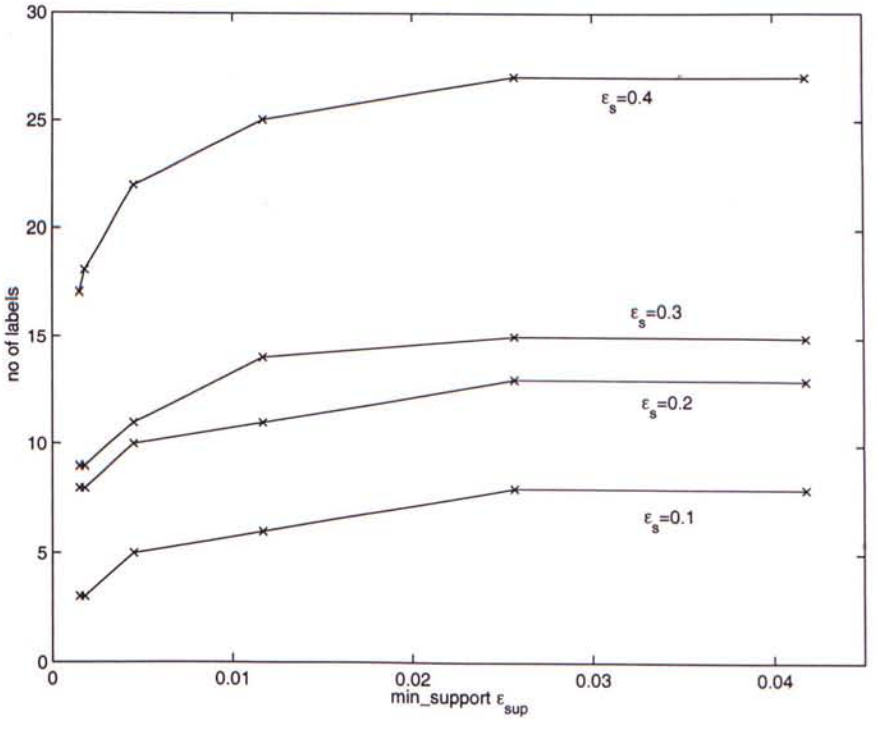


Figure 5.10: Number of labels with support $< \varepsilon_{sup}$

Figure 5.10 shows the number of label nodes regarded as non-candidate label nodes after phase 3 for different values of ε_{sup} and ε_s. We could observe that with a lower ε_{sup} fewer label nodes are regarded as non-candidate label nodes. However, with a lower ε_{sup} more non-label nodes are regarded as candidate label nodes and they will remain in the final result. In addition, the similarity threshold ε_s also affect the number of label nodes remained as non-candidate label nodes as

shown in Figure 5.10. The reason is that if ε_s is higher then higher similarity value between two nodes should be high enough to be similar and merged together. That means fewer nodes will be considered as similar and then fewer label nodes in non-candidate label nodes will be discovered as similar nodes to candidate label nodes. However, lower ε_s will let more dissimilar nodes merged together. Figure 5.11 shows the number of wrong similar labels for different values of the two thresholds. In a candidate label node, there is a set of labels $\{l_1, l_2, l_3, l_4\}$ where $l_1 \sim l_2 \sim l_3$ but l_4 is dissimilar to them, then l_4 is called the wrong similar label to this node. In the Figure, if ε_s is lower then the number of wrong similar labels is higher. Therefore the choice of ε_{sup} and ε_s could affect the results of the algorithm.

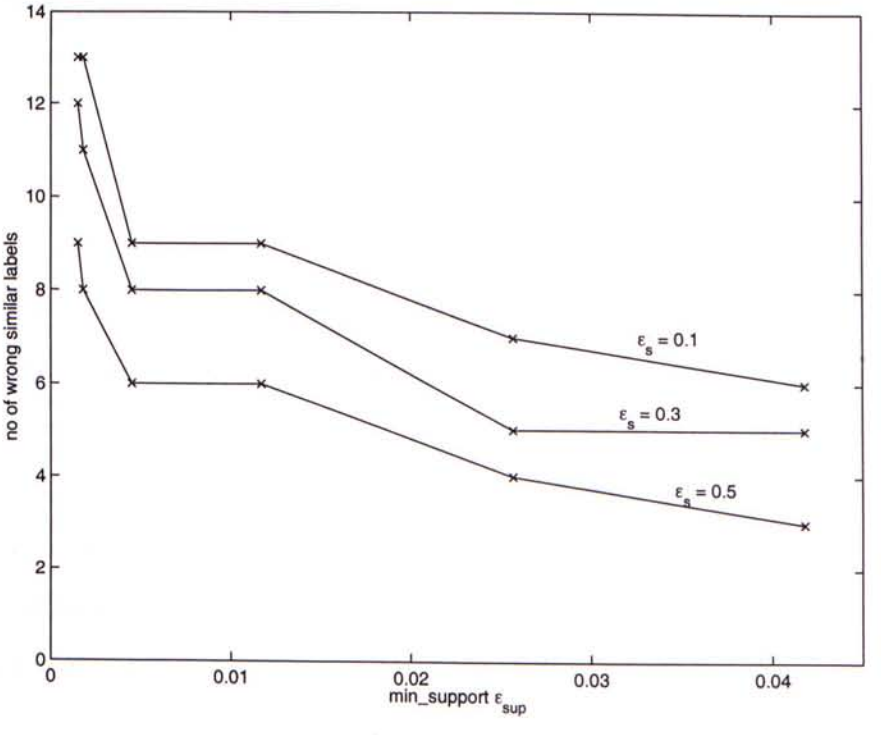


Figure 5.11: The number of wrong similar labels

Now we would show the similar labels discovered by the algorithm. Figure 5.12 shows the five nodes with highest support for $\varepsilon_{sup} = 0.0117$ and $\varepsilon_s = 0.3$. The wordings in a box are similar labels. In each box, all the wordings are correct similar labels. They represent the same attribute in the job employment class.

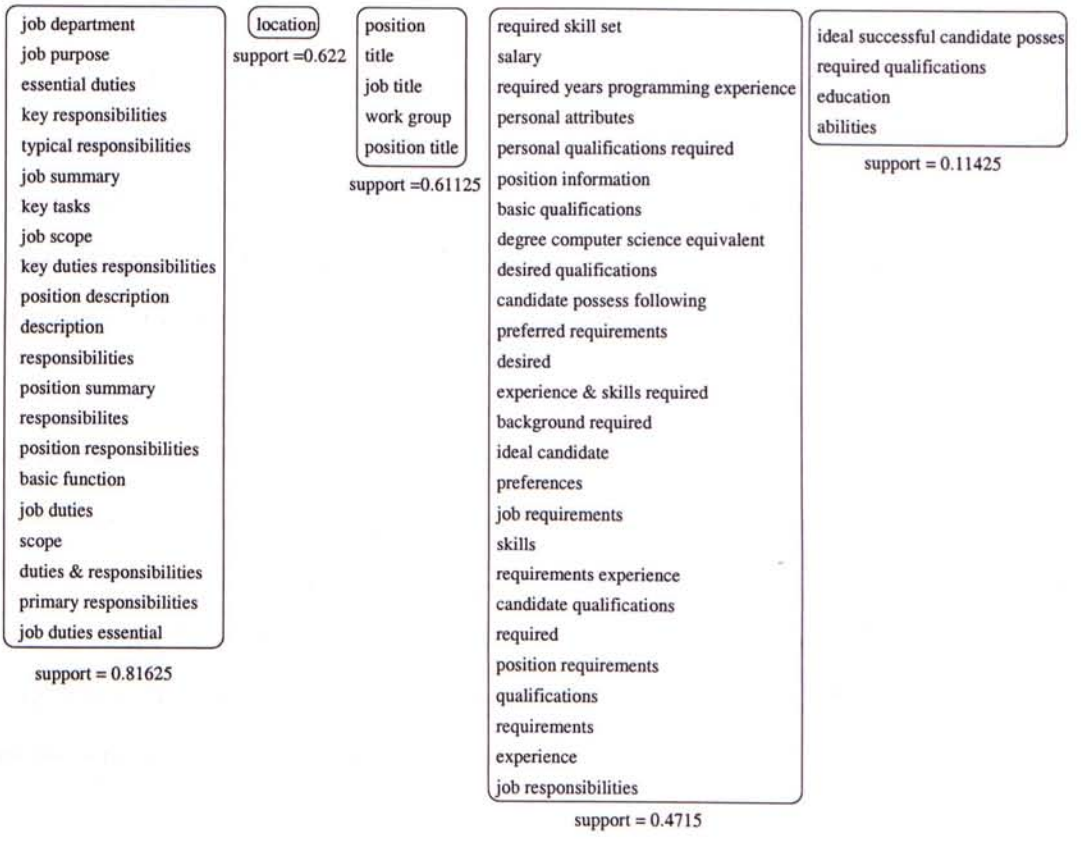


Figure 5.12: The five highest support nodes with $\varepsilon_{sup} = 0.0117$ and $\varepsilon_s = 0.3$

In the node with support=0.81625, all the labels describe the duty of a job, e.g. "job purpose", "typical responsibilities", "job summary", "key tasks", "job scope", etc. We should observe that most of similar labels could be discovered correctly. However, there are few errors. For example, "job department" is not similar to the labels in the node with support=0.81625. There are only very few labels are wrong. Therefore the labels discovery algorithm could discover similar labels in a class of web pages correctly.

5.4 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.

In labels discovery algorithm, we obtain a set of nodes which each contains a set of similar labels and a feature. Each node represents an attribute in the class of web pages. The feature is the characteristics of the corresponding attribute. This set of nodes could be used as the knowledge of the class of web pages. It also could be used as a classifier for the class. So we use the set of nodes to determine whether a web page belongs to the class. In next section, the similarity measurement of a web page to a class is defined.

5.4.1 Similarity Measurement

For a web page w to belong to a class C, the information stored in w is the same as that in the web pages in C. A class has a set of attributes used to describe the data stored in the class of web pages. Therefore, each web page in the class should be stored similar data with some common attributes. In definition 4, the structure of a web page is represented by a set of labels in that web page. In addition, in definition 8, the structure of a class of web pages is a set of labels-sets. The labels in the web page that belong to that class should be similar or identical to the labels in the set of labels-sets of the class. Therefore, by comparing the labels in the web page w to that of a class C, we could determine if w belongs to C.

In an attempt to measure the similarity between labels in a web page w and a class C of web pages, the hierarchical structure of the web page w should be

constructed as this could reveal the labels and values in w. Then each node is expressed as structure-expression. That is, a node n_i has a set of data set $\{d_o^{n_i}, d_p^{n_i}, d_l^{n_i}, d_r^{n_i}, d_o^{n_i}\}$. Then w consists of a set of nodes. Each node is then compared with those nodes in the knowledge obtained from labels discovery algorithm and has a similarity value. The similarity values indicate whether they are one of the labels in the class. If a node has high similarity then this node should contain a label that is in the class.

To compare the nodes between the web page w and the class C, each node in w should have a feature as described in the labels discovery algorithm. The right sibling and children data sets are union to form the feature of the node. Therefore w now consists of a set of nodes $N = \{n_1, n_2, \ldots, n_k\}$ where each node n_i contains an own data set $d_o^{n_i}$ and a feature f_{n_i}. However, unlike the nodes in the knowledge, the words in the feature in the nodes of w are not associated with confidence values.

There are two cases for the similarity measurement. First, if the own data set of a node n in w appears in a labels set of one node m in the class structure, then we said that the similarity value between the two nodes is 1. Otherwise, the features f_n and f_m of two nodes will be compared by the following similarity function.

$$sim\_feature(f_n, f_m) = \frac{\sum_{w_i \in f_n \cap f_m} Conf_m(w_i)}{\sum_{w_i \in f_m} Conf_m(w_i)} \times \frac{M}{N_{f_n}} \tag{5.8}$$

where M is the number of words matched in the two features, $|f_n \cap f_m|$, and N_{f_n} is the number of distinct words in f_n, $|f_n|$. If the intersection of the two features is large, then the two features would be similar as they contain many common characteristics. That is, the more two features have common words, the more they are similar. As well, if the common words have higher confidence then these words would be significance in the feature and the two features are more similar.

Then the similarity function between the node in w and C could be introduced accordingly. In the structure of a class of web pages, there is a node m_i containing

a set of labels L and a feature f_{m_i}. Then in a web page, there is a node n_j containing an own data set $d_o^{n_j}$, which may be a label, and a feature f_{n_j}. The similarity function of n_j and m_i is defined as follow.

$$sim\_node(n_j, m_i) = \begin{cases} 1 & \text{if } d_o^{n_j} \in L; \\ sim\_feature(f_{n_j}, f_{m_i}) & \text{otherwise.} \end{cases} \tag{5.9}$$

Each node n_j in w is compared to all nodes $M_C = \{m_1, m_2, \ldots, m_k\}$ in the class C. Then n_j has a set of similarity values. The highest one is regarded as the similarity value of n_j to C. The similarity function between a node n_j in a web page and a class C is defined below.

$$sim\_class(n_j, C) = \max_{m_i \in M_C} sim\_node(n_j, m_i) \tag{5.10}$$

Consequently, each node in the web page w has a similarity value the class C. We could define the similarity function between a web page w and the class C based on the similarity value of each node in w to C. We have two pre-defined thresholds ε_n and ε_m such that if w 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 C. The condition to be satisfied to classify a web page w with a set of nodes $\{n_1, n_2, \ldots, n_k\}$ to belong to a class C is then defined as follow.

$$|\{n_j : sim\_class(n_j, C) > \varepsilon_m\}| > \varepsilon_n \tag{5.11}$$

These two thresholds 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 thresholds.

5.4.2 Performance Evaluation

Experiments on the classification of web pages have been carried out. We use the structural knowledge obtained from the 4000 job employment web pages

in Section 5.3.3 for classification. There are 10897 web pages used to test the performance of the knowledge obtained in the labels discovery algorithm. In the 10897 web pages, there are 3651 job employment web pages and 7246 web pages of other classes. The 3651 job employment web pages are from 23 different companies arbitrarily chosen from the company list in yahoo search engine. These 23 companies are different to the 34 companies used in the experiments of labels discovery algorithm. The reason is that a company usually used the same format to represent its job employment data. That is structure of the web pages are the same. Therefore if the web pages from the same company are used to be classified then the performance should be very good. On the other hand, the 7246 non-related web pages used as noise to the experiment are retrieved from yahoo search engine. By using the yahoo search engine, web pages are iteratively retrieved by following the links in the retrieved web pages. These web pages are obtained by removing those web pages containing the keyword *job*.

	ε_m				
ε_n	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 5.6: Correctness for classification of positive web pages

	ε_m				
ε_n	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 5.7: Correctness for classification of negative web pages

The knowledge we used in the experiment is obtained with the threshold values, $\varepsilon_{sup} = 0.0117$ and $\varepsilon_s = 0.3$. 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 results are shown in Tables 5.6 and 5.7. Table 5.6 shows the accuracy of correctly identifying the web pages in the class. Table 5.7 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%.

Chapter 6

Conclusion

Seeing that large amount of data is available on the World Wide Web, we need to manage web data so as to query and search more efficiently. However, due to the semistructured nature of web data problems are arisen. Different labels representing the same attribute is one of these problems and it was discussed in this thesis.

First we define the characteristics of web data. Web pages containing the same type of information belong to the same class. Each class contains a number of attributes used to describe the data stored in the class of web pages. However, the attributes appear in different format in the web pages. We call the data used to represent the attribute in a web page to be label. Then the data of the attribute is called value. Due to loose standard of web pages publishing, different labels are used to represent the same attributes and we called them similar labels. Therefore similar labels are required to be discovered.

In an attempt to make the web data more convenient for discovering similar labels, a hierarchical structure, which is constructed by five heuristic methods, is introduced for each web page. The structure organizes data in the web page according to the concept hierarchical relation of data in the web page. As label describes briefly its value, it has higher concept hierarchical level than value. As

a result, from the structure we could discover that most values are in the children and right sibling nodes of their labels. From this relations, we could then propose labels discovery algorithm to discover similar labels.

Labels discovery algorithm consists of three phases. Phase one removes non-label nodes. Phase two identifies candidate label nodes. Phase three discover similar labels. The similarity between labels is calculated by their values. From the hierarchical structure, a feature containing values is formed for each label. By comparing the features of two labels we could determine whether they are similar labels. Experiments show that the algorithm could discover similar labels successfully from a class of web pages. The similar labels and their features are then used to classify web pages. From the experiments of web pages classification, high accuracy of classification is obtained. The similar labels and their features could be useful for web pages classification.

Bibliography

[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.M.G. Apers. Identifying internet-related database research. In *Proceedings of the 2nd International East-West Database Workshop*, pages 183–193, February 1998.

[6] G.O. Arocena and A.O. Mendelzon. Weboql: Restructuring documents, databases and webs. In *Proceedings of the Fourteen International Conference on Data Enineering*, February 1998.

[7] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* New York:ACM Press, Harlow, England, Addison-Wesley, 1999.

[8] S. Brin. Extracting patterns and relations from the world wide web. In *International Workshop on the Web and Database, WebDB'98*, Valencia, Spain, March 1998.

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

[10] 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.

[11] 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.

[12] 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 data 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.

[13] 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.

[14] 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.

[15] 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.

[16] D.S. Hochbaum. Heuristics for the fixed cost median problem. In *Mathematical Programming, 22*, pages 148–162, 1982.

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

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

[19] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data, September 1997.

[20] 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.

[21] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings Data Engineering Conference*, Taipei, Taiwan, March 1995.

[22] M. F. Porter. An algorithm for suffix stripping. In *Program, 14(3)*, pages 130–137, 1980.

[23] 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.

[24] D. Quass, J. Widom, R. Goldman, K. Haas, Q. Luo, J. McHugh, S. Nestorov, A. Rajaraman, H. Riveroa, S. Abiteboul, J. Ullman, and J. Wiener. Lore: A lightweight object repository for semistructured data. In *Proceedings of*

the ACM SIGMOD International Conference on Management of Data, page 549, Montreal, Canada, June 1996.

[25] G. Salton. *Automatic Information Organization and Retrieval.* New York: McGraw-Hill, 1968.

[26] G. Salton. *Automatic Text Processing: The transformation, analysis, and retrieval of information by computer.* Addison-Wesley, 1989.

[27] G. Salton, C. Yang, and C. Yu. A theory of term importance in automatic text analysis. *Journal of the American Society for Information Science*, pages 33–44, 1975.

[28] 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.

[29] J. Ullman. *Principles of Database and Knowledge-Base Systems.* Computer Science Press, Rockville, Maryland, 1989.

[30] The World Wide Web Consortium (W3C). http://www.w3.org/.

[31] 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.

[32] G. Wiederhold. *File Organization for Database Design.* McGraw Hill, New York, 1987.