# RoadRunner for Heterogeneous Web Pages Using Extended MinHash

A Suresh Babu[1], P. Premchand[2] and A. Govardhan[3]

[1]Department of Computer Science and Engineering, JNTUACE Pulivendula, India
asureshjntu@gmail.com
[2]Professor, Department of Computer Science Engineering, Osmania University, Hyderabad, India
p.prechand@uceou.edu
[3]Professor, Department of Computer Science Engineering, Director of evaluation JNTUH, Kukatpalli, Hyderabad, India.
govardhan_cse@yahoo.co.in

***ABSTRACT***

*The Internet presents large amount of useful information which is usually formatted for its users, which makes it hard to extract relevant data from diverse sources. Therefore, there is a significant need of robust, flexible Information Extraction (IE) systems that transform the web pages into program friendly structures such as a relational database will become essential. IE produces structured data ready for post processing. Roadrunner will be used to extract information from template web pages. In this paper, we present novel algorithm for extracting templates from a large number of web documents which are generated from heterogeneous templates. The proposed system focuses on information extraction from heterogeneous web pages. We cluster the web documents based on the common template structures so that the template for each cluster is extracted simultaneously. The resultant clusters will be given as input to the Roadrunner system.*

**KEYWORDS**

*Information Extraction, Clustering, Minimum Description Length Principle, MinHash*

## 1. INTRODUCTION

The sudden growth and popularity of World Wide Web has resulted in a huge amount of information sources on the Internet. However, due to the heterogeneity and lack of structure of Web information sources, access to this huge collection of information has been limited to browsing and searching. To automate the translation of input web pages into structured data, a lot of efforts have been devoted in the area of information extraction (IE). Unlike information retrieval (IR), which concerns with how to identify relevant documents from a document collection, IE produces structured data ready for post- processing, which is crucial to many applications of Web mining and searching tools. Formally, an IE task is defined by its input and its extraction target. The input pages can be unstructured documents like free text that are written in natural language or the semi-structured documents such as tables or itemized and enumerated lists.

Programs that perform the task of IE are referred to as extractors or wrappers. To give a simple but fairly faithful abstraction of the semantics of such data intensive web pages, we can consider the page-generation process as the result of two separated activities: (i) Execution of a number of queries on the underlying database to generate a source dataset. (ii)Second, the serialization of the

DOI: 10.5121/ijdms.2012.4103 

source dataset into HTML code to produce the actual pages, possibly introducing URLs links, and other material like images. Figure 1, refers to a fictional bookstore site. In that example, pages listing all books by one author are generated by a script, the script first queries the database to produce a nested dataset in which each tuple contains data about one author, her/his list of books, and for each book the list of editions, then, the script serializes the resulting tuples into HTML pages. When we run on these pages, our system will compare the HTML codes of the two pages, infer a common structure and a wrapper, and use that to extract the source dataset. The dataset extracted is produced in HTML format. As an alternative, it could be stored in a database. As it can be seen from the figure, the system deduces a nested schema from the pages. Since the database field names are generally not encoded in the pages and this schema is based purely on the content of the HTML code, it has anonymous fields (labelled by A, B, C, D, etc. in our example), which must be named manually after the dataset has been extracted.

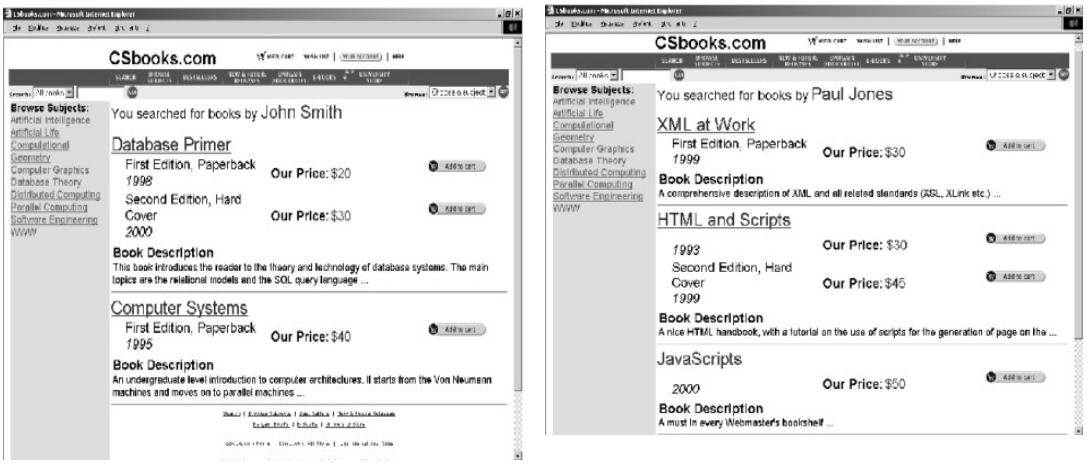


Fig.1. Input HTML Pages

However, for systems the unknown templates are considered harmful because they degrade the accuracy and performance due to the irrelevant terms in templates. The problem of extracting a template from the web documents conforming to a common template has been Studied in [2][3]. In such systems we are assuming that web pages generated will have common template, the solutions for this problems are applicable only when all the documents will have a common structure. However in real applications, it is not trivial to classify many crawled documents into homogeneous partitions in order to use these techniques.

To overcome the limitation of the techniques with the assumption that the web documents are from a single template, the problem of extracting the templates from a collection of heterogeneous web documents, which are generated from multiple templates. In this problem, clustering of web documents such that the documents in the same group belong to the same template is required, and thus, the correctness of extracted templates depends on the quality of clustering.

## 2. RELATED WORK

The template extraction problem can be classified into two broad areas. The first one is the site-level template detection where the template is decided based on number of pages from the same site. Crescenzi et al. [1] studied initially the data extraction problem and Yossef and Rajagopalan [2] introduced the template detection problem. Previously, only tags were considered to find templates but Arasu and Garcia-Molina [3] observed that any word can be a part of the template or contents. However, they detect elements of a template by the frequencies of words but we consider the MDL principle as well as the frequencies to decide templates from heterogeneous

documents. Zhao et al. [4] concentrated on the problem of extracting result records from search engines.

Garofalakis et al. [5] solved the problem of DTD extraction from multiple XML documents. The solutions for XML documents fully utilize these properties. In the problem of the template extraction from heterogeneous document, how to partition given documents into homogeneous subsets is important. However, the tree edit distance is expensive and it is not easy to select good training pages. Crescenzi et al. [7] focused on document clustering without template extraction. Biclustering or co clustering is another clustering technique to deal with a matrix [8], [9], [10]. Since an HTML document can be represented with a Document Object Model (DOM) tree, web documents are considered as trees and many existing similarity measures for trees have been investigated for clustering [6]. However, clustering is very expensive with tree-related distance measures. For instance, tree-edit distance has at least O ($n_1 n_2$) time complexity [6], where n1 and n2 are the sizes of two DOM trees and the sizes of the trees are usually more than a thousand. Thus, clustering on sampled web documents is used to practically handle a large number of web documents.

Reis et al. [6] presented a method in which a small number of sampled documents are clustered first, and then, the other documents are classified to the closest clusters. In both clustering and classifying, a restricted tree-edit distance is used to measure the similarity between documents.

In this paper, in order to overcome the limitations of the technologies, we investigate the problem of detecting the templates from heterogeneous web documents and present novel algorithms called Automatic Template Extraction. We represent a web page and a template as a set of paths in a DOM tree. As validated by the most popular XML query language XPATH [11], paths are sufficient to express tree structures and useful to be queried. By considering only paths, the overhead to measure the similarity between documents becomes small without significant loss of information.

For example, let us consider simple HTML documents and paths in Fig. 2 and Table 1. We will formally define the paths later. Document d1 is represented as a set of paths {p1, p2, p3, p4, p6} and the template of both d1 and d2 is another set of paths {p1, p2, p3, p4}.

Our goal is to manage an unknown number of templates and to improve the efficiency and scalability of template detection and extraction algorithms. To deal with the unknown number of templates and select good partitioning from all possible partitions of web documents, we employ Rissanen's Minimum Description Length (MDL) principle in [12], [13]. In our problem, after clustering documents based on the MDL principle, the model of each cluster is the template itself of the web documents belonging to the cluster. Thus, we do not need additional template extraction process after clustering. In order to improve efficiency and scalability to handle a large number of web documents for clustering, we extend MinHash [14].

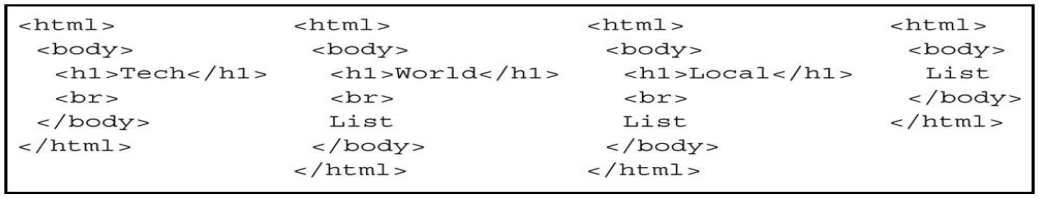


Fig.2. Simple web documents. (a) Document d1. (b) Document d2. (c) Document d3. (d) Document d4.

TABLE 1
Paths of Tokens and Their Supports

| ID | Path | Support |
|---|---|---|
| $p_1$ | Document\〈html〉 | 4 |
| $p_2$ | Document\〈html〉\〈body〉 | 4 |
| $p_3$ | Document\〈html〉\〈body〉\〈h1〉 | 3 |
| $p_4$ | Document\〈html〉\〈body〉\〈br〉 | 3 |
| $p_5$ | Document\〈html〉\〈body〉\List | 3 |
| $p_6$ | Document\〈html〉\〈body〉\〈h1〉\Tech | 1 |
| $p_7$ | Document\〈html〉\〈body〉\〈h1〉\World | 1 |
| $p_8$ | Document\〈html〉\〈body〉\〈h1〉\Local | 1 |

While the traditional MinHash is used to estimate the Jaccard coefficient between sets, we propose an extended MinHash to estimate MDL cost measure with partial information of documents. In summary, our involvement is as follows:

- We apply the MDL principle to our problem to effectively manage an unknown number of clusters.

- In our method, document clustering and template extraction are done together at once. The MDL cost is the number of bits required to describe data with a model and the model in our problem is the description of clusters represented by templates.

- Since a large number of web documents are massively crawled from the web, the scalability of template extraction algorithms is very important to be used practically. Thus, we extend MinHash technique to estimate the MDL cost quickly, so that a large number of documents can be processed.

## 3. PRELIMINARIES

### 3.1 Essential Paths and Templates

Given a web document collection D= { d1, d2, . . . , dn}, we define a path set $P_D$ as the set of all paths in D. Note that, since the document node is a virtual node shared by every document, we do not consider the path of the document node in $P_D$. The support of a path is defined as the number of documents in D. For each document $d_i$, we provide a minimum support threshold $t_{di}$ . Notice that the thresholds $t_{di}$ and $t_{dj}$ of two distinct documents $d_i$ and $d_j$, respectively, may be different. If a path is contained by a document $d_i$ and the support of the path is at least the given minimum support threshold $t_{di}$ , the path is called an essential path of $d_i$. We denote the set of essential paths of an HTML document $d_i$ by $E(d_i)$. For a web document set D with its path set $P_D$, we use a $|P_D|\times|D|$ matrix $M_E$ with 0/1 values to represent the documents with their essential paths. The value at a cell (I, j) in the matrix $M_E$ is 1 if a path pi is an essential path of a document dj. Otherwise, it is 0.

**Example 1**. Consider the HTML documents D={d1,d2, d3, d4} in Fig. 2. All the paths and their frequencies in D are shown in Table 1. Assume that the minimum support thresholds $t_{d1}$ , $t_{d2}$ , $t_{d3}$ , and $t_{d4}$ are 3, 3, 3, and 4, respectively. The essential path sets are $E(d_1)$={p1,p2, p3,p4}, $E(d_2)$={p1, p2, p3, p4, p5}, $E(d_3)$={p1, p2, p3, p4, p5}, and $E(d_4)$={p1,p2}. We have the path set $P_{D=\{}$pi/1≤i≤ 8} and the matrix $M_E$ becomes as follows:

$M_E$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

**Example 2:** In Fig. 2 and Table 1, the paths appearing at the document d2 are p1, p2, p3, p4, p5, and p7 whose supports are 4, 4, 3, 3, 3, and 1, respectively. Since 3 is the mode of them, we use 3 as the minimum support threshold value td2 . Then, p1, p2, p3, p4, and p5 are essential paths of d2.

### 3.3 Matrix Representation of Clustering

We next illustrate the representation of a clustering of web documents. Let us assume that we have m clusters such as C ={$c_1$, $c_2$, . . . $c_m$} for a web document set D. A cluster ci is denoted by a pair ($T_i$,$D_i$), where $T_i$ is a set of paths representing the template of $c_i$ and $D_i$ is a set of documents belonging to $c_i$. In our clustering model, we allow a document to be included in a single cluster only. That is, we have Di∩Dj =null set, for all distinct clusters $c_i$, $c_j$, and $\bigcup_{1\le i\le m} D_i$=D**.** To represent a clustering information C ={c1,c2. . cm} for D, we use a pair of matrices $M_T$ and $M_D$, where $M_T$ denotes the information of each cluster with its template paths and $M_D$ denotes the information of each cluster with its member documents. If the value at a cell (I, j) in $M_T$ is 1, it means that a path $p_i$ is a template path of a cluster $c_j$. Otherwise, $p_i$ does not belong to the template paths of $c_j$. Similarly, the value at a cell (i,j) in $M_D$ is 1 if a document $d_j$ belongs to a cluster $c_i$. Regardless of the number of clusters, we fix the dimension of $M_T$ as $|P_D| \times |D|$ and that of $M_D$ as $|D| \times |D|$. Rows and columns in $M_T$ and $M_D$ exceeding the number of clusters are filled with zeros. In other words, for a clustering with C ={c1,c2, . . . cm}, all values from (m+1)th to IDI th columns in $M_T$ are zeros, and all values from (m + 1)th to IDI th rows in MD are zeros. We will represent $M_E$ by the product of $M_T$ and $M_D$. However, the product of $M_T$ and $M_D$ does not always become $M_E$. Thus, we reconstruct $M_E$ by adding a difference matrix MΔ with 0/1/-1 values to $M_T.M_D$, i.e., $M_E = M_T.M_D+M_\Delta$.

$M_T$

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$M_\Delta$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$M_D$

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

**Example 3**. Consider the web documents in Fig. 3 and $M_E$ in Example 1 again. Assume that we have a clustering C = {c1,c2}, where c1 = ( {p1, p2, p3, p4, p5} {d1,d2,d3} ) and c2 = ( {p1,p2},{d4} ). Then, $M_T$ , $M_D$, and $M_\Delta$ are as follows and we can see that ME=$M_T$ .$M_D$+MΔ

### 3.4 Minimum Description Length Principle (MDL)

In order to handle unknown number of clusters and to select good partitioning from all possible partitions of HTML documents, we apply Rissanen's MDL principle [12], [13]. The MDL principle states that the best model inferred from a given set of data is the one which minimizes the sum of 1) the length of the model, in bits, and 2) the length of encoding of the data, in bits. We refer to the above sum for a model as the MDL cost of the model. In our setting, the model is a clustering C, which is described by partitions of documents with their template paths (i.e., the matrices $M_T$ and $M_D$), and the encoding of data is the matrix $M_\Delta$. The MDL costs of a clustering model C and a matrix M are denoted as L(c) and L(M), respectively. Considering the values in a matrix as a random variable X, Pr(1) and Pr(-1) are the probabilities of 1 s and -1 s in the matrix and Pr(0) is the probability of zeros. Then, the entropy H(X) of the random variable X [18], [19] is as follows:

$$\sum_{x\in(1,0,-1)} -pr(X)\log_2 pr(X) \quad \text{and}$$

L(M)= IMI.H(X).

The MDL costs of $M_T$ and $M_\Delta$ is $L(M_T)$ and $L(M_\Delta)$ respectively are calculated by the above formula. For $M_D$, we use another method to calculate its MDL cost. The reason is that the random variable X in $M_D$ is not mutually independent, since we allow a document to be included in a single cluster (i.e., each column has only a single value of 1). Thus, we encode $M_D$ by |D| number of cluster IDs. Since the number of bits to represent a cluster ID is $\log_2$ |D|, the total number of bits to encode $M_D$ (i.e., $LM_D$) becomes |D|. $\log_2$ |D|. Then, the MDL cost of a clustering model C is defined as the sum of the three matrices (i.e., $L(C) = L(M_T) + L(M_D) + L(M_\Delta)$). According to the MDL principle, for two clustering models $C =(M_T,M_D)$ and $C'=(M'_T,M'_D)$, we say that C is a better clustering than C' if L(C) is less than L(C').

Example 4. Again consider the clustering C in Example 3. Then, with $M_T$ in Example 3, Pr(1) = 7/32 and Pr(0) = 25/32 and we have $L(M_T)$ =|MT|. H(X) = 32.(- 7/32 $\log_2$7/32-25/32 $\log_2$25/32) = 24.25. Similarly, $L(M_D)$= 8, $L(M_\Delta)$= 6.42, and thus, L(C) = 38.67. For another clustering C', let us assume that $M'_T$, $M'_D$, and $M'_\Delta$ are as follows:

Then, $L(M_T)$=17.39,$L(M_D)$= 8, $L(M_\Delta)$=21.39, and thus, L(C')=46.78. Since L(C) < L(C'), we say C is a better clustering than C'. It is natural to think that d1, d2, and d3 are generated by the same template and d4 looks different from the others. Thus, the clustering C grouping d1, d2, and d3 together and isolating d4 is better than the other clustering C0 grouping d1, d2, d3, and d4 altogether. Thus, we can see that it is intuitively reasonable to prefer C to C'.

$M'_T$

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$M_E$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$M'_\Delta$

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

### 3.5 RoadRunner

### Matching Technique

This section is devoted to the presentation of algorithm match. It is based on a matching technique called ACME, for Align, Collapse under Mismatch, and Extract. To avoid errors and missing tags in the sources, the HTML code complies to the XHTML specification, a restrictive variant of HTML in which tags are required to be properly closed and nested. The matching algorithm works on two objects at a time: (i) a list of tokens, called the sample, and (ii) a wrapper, i.e., one union-free regular expression. Given two HTML pages (called page 1 and page 2), to start take one of the two, for example page 1, as an initial version of the wrapper, then, the wrapper is progressively refined trying to find a common regular expression for the two pages. This is done by solving mismatches between the wrapper and the sample. A mismatch happens when some token in the sample does not comply to the grammar specified by the wrapper. Mismatches are very important, since they help to discover essential information about the wrapper. Whenever one mismatch is found, it tries to solve the mismatch by generalizing the wrapper. The algorithm succeeds if a common wrapper can be generated by solving all mismatches encountered during the parsing.

**Mismatches** There are essentially two kinds of mismatches that can be generated during the parsing: (a) String mismatches, i.e., mismatches that happen when different strings occur in corresponding positions of the wrapper and sample. (b) Tag mismatches, i.e., mismatches between different tags on the wrapper and the sample, or between one tag and one string.

**String Mismatches: Discovering Fields** It can be seen that, if the two pages belong to the same class, string mismatches may be due to different values of a database field. Therefore, these mismatches are used to discover fields (i.e., #PCDATA).

**Tag Mismatches: Discovering Optional** Tag mismatches are used to discover iterators and optionals. In the presence of such mismatches, our strategy consists in looking for repeated patterns (i.e., patterns under an iterator) as a first step, and then, if this attempt fails, in trying to identify an optional pattern.

**Tag Mismatches: Discovering Iterators** Repeated tags will be replaced by a iterator (+)

## 4. PROPOSED APPROACH

### 4.1 Clustering With MDL Cost

Our clustering algorithm TEXT-MDL is presented in Fig. 3. The input parameter is a set of documents D = {d1, . . . , dn}, where $d_i$ is the i th document. The output result is a set of clusters C ={c1, . . . , cm} , where $c_i$ is a cluster represented by the template paths Ti and the member documents Di (i.e., ci =($T_i$,$D_i$). A clustering model C is denoted by two matrices $M_T$ and $M_D$ and the goodness measure of the clustering C is the MDL cost L(C) , which is the sum of $L(M_T)$, $L(M_D)$, and $L(M_\Delta)$. TEXT-MDL is an agglomerative hierarchical clustering algorithm which starts with each input document as an individual cluster (in line 1). When a pair of clusters is merged, the MDL cost of the clustering model can be\ reduced or increased. The procedure GetBestPair finds a pair of clusters whose reduction of the MDL cost is maximal in each step of merging and the pair is repeatedly merged until any reduction is not possible. In order to calculate the MDL cost when each possible pair of clusters is merged, the procedure GetMDLCost(ci, cj, C), where ci and cj are a pair to be merged and C is the current clustering, is called in GetBestPair and C is updated by merging the best pair of clusters. As we will discuss later in detail, because

the scale of the MDL cost reduction by merging a pair of clusters is affected by all the other clusters, GetBestPair should recalculate the MDL cost reduction of every pair at each iteration of while loop in line 7.

```
algorithm TEXT-MDL(D /* a set of document */)
begin
1.  C := {c_1, c_2, ..., c_n} with c_i = (E(d_i), {d_i});
2.  (c_i, c_j, c_k) := GetBestPair(C);
3.  /* Let c_i and c_j be the best pair for merging */
4.  /* Let c_k be a new cluster made by merging c_i and c_j */
5.  while (c_i, c_j, c_k) is not empty do {
6.     C := C − {c_i, c_j} ∪ {c_k};
7.     (c_i, c_j, c_k) := GetBestPair(C);
8.  }
9.  return C
end
procedure GetBestPair(C /* a clustering model */)
begin
1.  MDLcost_min := ∞;
2.  for each pair (c_i, c_j) of clusters in C do {
3.     (MDLcost, c_k) := GetMDLCost(c_i, c_j, C);
4.     /* GetMDLCost returns the optimal MDL cost
5.     when c_k is made by merging c_i and c_j */
6.     if MDLcost < MDLcost_min then {
7.        MDLcost_min := MDLcost;
8.        (c_i^B, c_j^B, c_k^B) := (c_i, c_j, c_k);
9.     }
10. }
11. return (c_i^B, c_j^B, c_k^B);
end
```

Fig.3. TEXT-MDL algorithm.

## 4.2 Extended minHash

To compute the MDL cost of each clustering quickly, we would like to estimate the probability that a path appears in a certain number of documents in a cluster. However, the traditional MinHash was proposed to estimate the Jaccard's coefficient. Thus, given a collection of sets X = {S1, . . . , Sk}, we extend MinHash to estimate the probabilities needed to compute the MDL cost. Let us start from defining a probability that an rj $\in \bigcup_{Sl \in X} Sl$ is included by m number of sets in X. We denote the probability as $\xi$(X,M), which is computed as follows:

$$\xi(X,M) = \frac{|\{r_j / r_j \text{ is included in m number of sets in X}\}|}{|S1 \cup \ldots \cup Sk|}$$

Then, $\xi$(X,M) is defined for $1 \leq m \leq |X|$ and $\xi(X,|X|)$is the same as the Jaccard's coefficient of sets in X. Now we introduce an extended signature of a collection of k sets, X ={S1, . . . , Sk}, for $\pi = \{ \pi_{1,\ldots,} \pi L \}$ as follows

$$\begin{cases} sigX[i] = (\min sig\ sj, \quad |argmin\ sig\ sj[i]|)\ for \prod i \\ sig\ X = [sig\ x[1],..,sigx[l]] \end{cases}$$

The extended signature sigXli| for $\prod_i$ is a pair of the minimum sig $s_{i|}$i| for all $s_i$ in X (i.e., minSjϵx sigSj |½|) and the number of sets whose signature for $\prod_i$ is the same as the minimum (i.e., |argminSjϵxsigs$_j$ |i|). The former is denoted as r(sig xli|) and the latter is denoted as n(sigXli|) in the rest of this paper. Notice that r(sigXli|) is the same as min($\prod_i$($S_1$U..$US_K$)), and thus, Pr(r(sigXli| =$\prod_i$($r_l$)=1/ |$S_1$U..$US_K$| for every $r_l \epsilon$ $S_1$U..$US_K$ and every $\prod_i \epsilon \prod$ very _i 2 _. Therefore, $\prod$ is still min wise independent for our extended r(sigXli|). The relation between Pr(n(sigXli|)= m) and $\xi$(X,M) is shown in the following lemma:

**Lemma 2**. Pr(n(sigXli|)= m) is the same as $\xi$(X,M)

**Proof.** $\prod$ is minwise independent for r(sigXli|), and thus, Pr(r(sigXli| =$\prod_i(r_l)$) =1/ |$S_1$U..US$_K$| for every $r_l \in S_1$U..US$_K$ and every $\prod_i \in \prod$. The set of $r_j$s contained in m number of $S_l$s in X is denoted as {rj/rj is included in m number of sets in X}, and therefore, Pr(r(sigXli| = m) is the same as {rj/rj is included in m number of sets in X}/|$S_1$U..US$_K$|, which is equivalent to $\xi$(X,M). According to Lemma 2, we can estimate $\xi$(X,M) with
Sig x as follows: $\xi$(X,M) = |{i/n(sigXli|)=m}|/$\prod$ (5)

### 4.3 Calculation of MDL Cost Using MinHash

Recall that, if we know sup($p_k$,$D_i$) for each $p_k$, we can decide the optimal $T_i$ and calculate the MDL cost of a clustering as shown in Theorem 2. However, even if we do not know the essential paths in each document, but we have 1) $\alpha$= Pr(1) and $\beta$= H(X) in $M_E$, 2) the number of the essential paths in each document, and 3) the probabilities that a path in Ei is supported by k documents in a cluster ci (i.e., $\xi$(Di,k)), we can compute the MDL cost of a clustering.

```
procedure GetHashMDLCost(c_i, c_j, C)
begin
1.  D_k := D_i ∪ D_j, c_k := (∅, D_k), C' := C − {c_i, c_j} ∪ {c_k};
2.  for each π_q in Π do {
3.     r(sig_{D_k}[q]) := min(r(sig_{D_i}[q]), r(sig_{D_j}[q]));
4.     if r(sig_{D_i}[q]) == r(sig_{D_j}[q]) then
5.        n(sig_{D_k}[q]) := n(sig_{D_i}[q]) + n(sig_{D_j}[q]);
6.     else n(sig_{D_k}[q]) is from the less one;
7.  }
8.  Calculate ξ̂(D_k, ℓ) by Equation (5);
9.  Compute n(D_k, k) by Lemma 4;
10. Get Pr(1) and Pr(−1) in M_T and M_Δ by Lemma 3;
11. MDL := Approximate MDL cost of C' by Equation (1);
12. return (MDL, c_k);
end
```

Fig.4. GetHashMDLCost procedure.

Computation of MDL cost using n(Di,k). Recall that sup(px,Di) is the number of documents in Di having the path px as an essential path. Let n(Di, k) represent the number of paths px whose sup(px,Di) is k. The following lemma shows that we can count the numbers of 1s and -1s in $M_T$ and $M_\Delta$ by using only n(Di, k) MDL cost estimation by MinHash. Now we present the procedure GetHashMDLCost in Fig. 4. Note that we estimate the MDL cost, but do not generate the template paths of each cluster. Thus, Tk of ck is initialized as the empty set (line 1). Instead of the template paths, the signature of ck (i.e., sig Dk) is maintained to estimate the MDL cost (lines 2-7). If we consider the length of a signature as a constant, the complexity of GetHashMDLCost is O(1). After finishing clustering, a post processing is needed to get the actual template paths. We refer to the processing as the template path generation step.

### 4.4 Clustering With MinHash

When we merge clusters hierarchically, we select two clusters which maximize the reduction of the MDL cost by merging them. Given a cluster ci, if a cluster cj maximizes the reduction of the MDL cost, we call cj the nearest cluster of ci. In order to efficiently find the nearest cluster of ci, we use the heuristic below: In Fig. 5, we provide the procedures to find the best pair using MinHash. In TEXT-MDL in Fig. 3, the GetBestPair at line 2 is replaced by GetInitBestPair and the GetBestPair at line 7 is replaced by GetHashBestPair. In GetInitBestPair, we first merge clusters with the same signature of MinHash (line 1). Next, for each cluster ci, we get clusters

with the maximal Jaccard's coefficient estimated by the signatures of MinHash (line 4) and compute the MDL cost of each pair

```
Procedure GetInitBestPair(C)
begin
1.   Merge all clusters with the same signature of MinHash;
2.   MDL_min := ∞;
3.   for each c_i in C do {
4.       N := clusters with the maximal Jaccard's coeff. with c_i;
         /* If the maximal Jaccard's coefficient is 0, N is ∅ */
5.       for each c_j in N do {
6.           (MDL_tmp , c_k) := GetHashMDLCost(c_i, c_j, C);
7.           if MDL_tmp < MDL_min then {
8.               MDL_min := MDL_tmp;
9.               (c_i^B , c_j^B , c_k^B) := (c_i, c_j, c_k);
10.          }
11.      }
12.  }
13.  return (c_i^B, c_j^B, c_k^B);
end
Procedure GetHashBestPair(c_k, C)
begin
1.   (c_i^B, c_j^B) := the current best pair;
2.   c_k^B := a cluster made by merging c_i^B and c_j^B;
3.   MDL_min := the current best approximate MDL cost;
4.   N := clusters with the maximal Jaccard's coeff. with c_k;
     /* If the maximal Jaccard's coefficient is 0, N is ∅ */
5.   for each c_ℓ in N do {
6.       (MDL_tmp , c_tmp) := GetHashMDLCost(c_k, c_ℓ, C);
7.       if MDL_tmp < MDL_min then {
8.           MDL_min := MDL_tmp;
9.           (c_i^B , c_j^B , c_k^B) := (c_k, c_ℓ, c_tmp);
10.      }
11.  }
12.  return (c_i^B , c_j^B , c_k^B);
end
```

Fig.5. GetInitBestPair/GetHashBestPair procedures

(lines 5-11). The complexities of GetInitBestPair and GetHashBestPair depend on the number of clusters with the maximal Jaccard's coefficient. If we consider it as a constant, the complexities of GetInitBestPair and GetHashBestPair are O(n) and O(1), where n is the number of documents in D. The complexity of the post process to compute the optimal Ti of each cluster is O(ne), where e is the average size of a document since we just scan documents in D once.

## 5. Experimental Results

All experiments reported in this section were performed on a Pentium-Core2 2.66 GHz machine with 2 GB of main memory. All algorithms were implemented in JAVA with JRE version 1.7.0 and we used HTML Parser version 1.6 (http://htmlparser.sourceforge.net) to parse the input HTML files.

**Algorithms:** TEXT-MDL: It is the naive agglomerative clustering algorithm with the approximate entropy model. It requires no input parameter.

TEXT-HASH: It is the agglomerative clustering algorithm with MinHash signatures. It requires an input parameter which is the length of MinHash signature.

ROADRUNNER: It is a novel technique to extract information from template web pages. It requires two input html pages at a time.

Considering the space constraint as a limited resource let us take the two web pages shown in fig.6 and 7 as an input pages which are belongs to different clusters(templates). The traditional RoadRunner works only for common template pages.

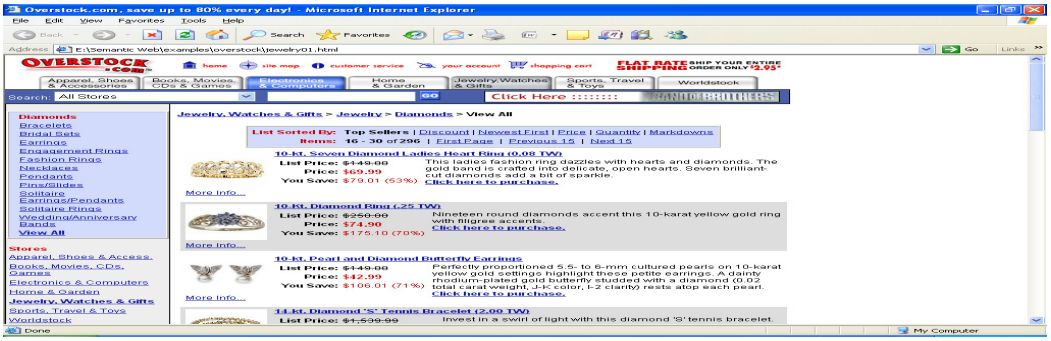

Fig.6. Input page1

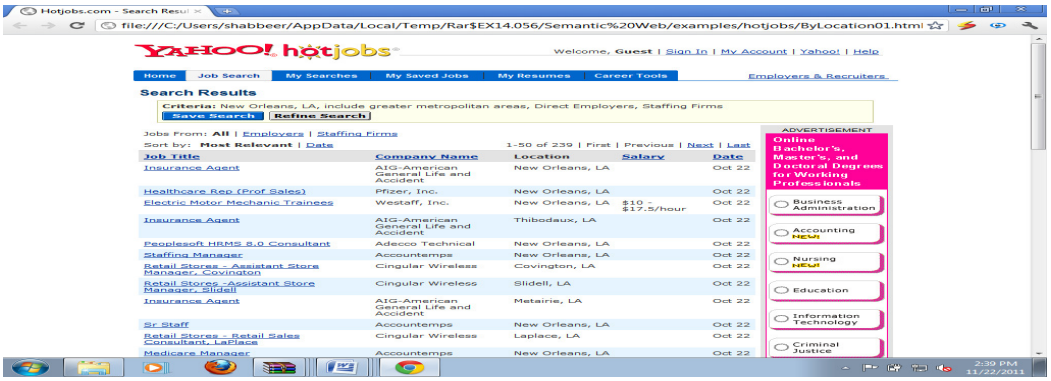

Fig.7. Input page 2

Now consider the system ”RoadRunner for heterogeneous web pages” which takes the heterogeneous html web pages as input and partitions them into clusters based on similarity. And finally the resultant clusters are given as input to the RoadRunner System. And the system will generate the Data extraction output for each cluster. The labelling for attributes have to give by the human manually. The expected output for input page1 Fig(6) is shown in fig. 8.

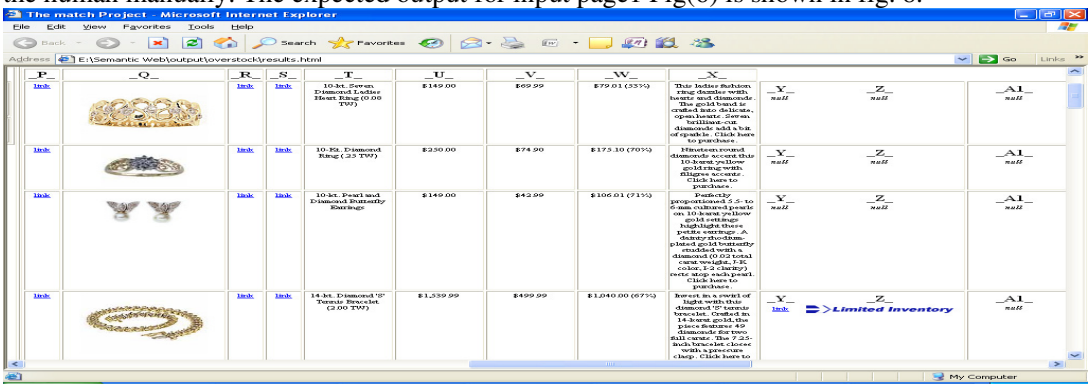

Fig.8. Data Extraction output (Dataset)

## 6. CONCLUSIONS

We introduced a novel approach of the template detection from heterogeneous web documents. We employed the MDL principle to manage the unknown number of clusters and to select good partitioning from all possible partitions of documents, and then, introduced our extended MinHash technique to speed up the clustering process. Experimental results with real life data sets confirmed the effectiveness of our algorithms. And finally resultant clusters will be given as

input to the Roadrunner system. The same technique can also be applied for any page-level web information extraction system.

## REFERENCES

[1] V. Crescenzi, G. Mecca, and P. Merialdo, "Roadrunner: Towards Automatic Data Extraction from Large Web Sites," Proc. 27th Int'l Conf. Very Large Data Bases (VLDB), 2001.

[2] Z. Bar-Yossef and S. Rajagopalan, "Template Detection via Data Mining and Its Applications," Proc. 11th Int'l Conf. World Wide Web (WWW), 2002.

[3] A. Arasu and H. Garcia-Molina, "Extracting Structured Data from Web Pages," Proc. ACM SIGMOD, 2003.

[4] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu, "Fully Automatic Wrapper Generation for Search Engines," Proc. 14th Int'l Conf. World Wide Web (WWW), 2005.

[5] H. Zhao, W. Meng, and C. Yu, "Automatic Extraction of Dynamic Record Sections from Search Engine Result Pages," Proc. 32nd Int'l Conf. Very Large Data Bases (VLDB), 2006.

[6] M.N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim, "Xtract: A System for Extracting Document Type Descriptors from Xml Documents," Proc. ACM SIGMOD, 2000.

[7] V. Crescenzi, P. Merialdo, and P. Missier, "Clustering Web Pages Based on Their Structure," Data and Knowledge Eng., vol. 54, pp. 279- 299, 2005.

[8] D. Chakrabarti, R. Kumar, and K. Punera, "Page-Level Template Detection via Isotonic Smoothing," Proc. 16th Int'l Conf. World Wide Web (WWW), 2007.

[9] I.S. Dhillon, S. Mallela, and D.S. Modha, "Information-Theoretic Co-Clustering," Proc. ACM SIGKDD, 2003.

[10] B. Long, Z. Zhang, and P.S. Yu, "Co-Clustering by Block Value Decomposition," Proc. ACM SIGKDD, 2005.

[11] Xpath Specification, http://www.w3.org/TR/xpath, 2010.

[12] J. Rissanen, "Modeling by Shortest Data Description," Automatica, vol. 14, pp. 465-471, 1978.

[13] J. Rissanen, Stochastic Complexity in Statistical Inquiry. World Scientific, 1989.

[14] Z. Chen, F. Korn, N. Koudas, and S. Muithukrishnan, "Selectivity Estimation for Boolean Queries," Proc. ACM SIGMOD-SIGACTSIGART Symp. Principles of Database Systems (PODS), 2000.