Journal of Web Engineering, Vol.2, No.3 (2004) 176-192

A PREPROCESSING FRAMEWORK AND APPROACH FOR WEB APPLICATIONS

ZHIGANG ZHANG, JING CHEN, XIAOMING LI

Peking University, Beijing

{zzg,jin,lxm}@net.pku.cn

Received October 9, 2003
Revised January 28, 2004

Aiming to meet the common requirements of several typical web applications, we propose a new preprocessing framework and the corresponding approach. The framework includes three parts: Web page cleaning, replica removal and Web page integration. After the preprocessing stage, Web pages are purified and transformed into a general model called DocView. The model consists of eight elements, identifier, type, content classification code, title, keywords, abstract, topic content, relevant hyperlinks. Most of them are meta data, while the latter two are content data. The approach first partitions a page into several content blocks according to some selected tags in the markup tag tree. Based on a set of heuristics, it identifies the blocks that contain the topic content of the page. Then a quantitative measure (a feature vector) of the blocks with respect to the topic is obtained. From the topic feature vector, the elements of DocView are extracted by corresponding algorithms. The main advantage of our approach is no need for other information beyond the raw page, while additional information is usually necessary for previous related work. The preprocessing framework and approach have been applied to our search engine (Tianwang [15]) and web page classification system. The strong evidence of improvement in applications shows the practicability of the framework and verifies the validity of the approach. It's not difficult to realize that after such a preprocessing stage, we can set up a well-formed, purified, easily manipulated information layer on top of any Web page collection (including WWW) for Web applications.

Key words: World Wide Web, Data preprocessing, Data cleaning, Data integration
Communicated by: D Schwabe & Q Li

1 Introduction

Today, most information resources on the WWW are still published as HTML pages, and the number of the web pages is increasing rapidly with the expansion of the Web. In order to make better use of Web information, technologies that can automatically re-organize and manipulate web pages are pursued such as Web information retrieval, Web page classification and other Web mining work. These technologies usually face two problems.

First: Useful information on the Web is often accompanied by a large amount of "noise content", which is a main difference between Web pages and traditional texts. Web "noise content" can be grouped into two categories according to their granularities [3]:

> *Global noises*: These are noises on the Web with large granularity, which usually include mirror sites, duplicated Web pages and near-replicas. Global noises not only tamper with the quality of crawling and ranking function of Web information retrieval system such as search engine, but also lead archiving applications to waste large disk space storing the duplicated pages.

Local (intra-page) noises: These are noisy content within a Web page. Local noises are usually incoherent with the topic contents of the Web page such as advertisements, navigation panels, and copyright announcements, etc.. Local noises make it very difficult for programs to grasp the topic content of a page, so they also worsen the quality of Web applications. We will discuss the problems in detail in section *1.2*. (Related Work).

Maybe these "noises" are useful information in some Web applications, but in most cases, they are generally harmful to Web applications.

Second: Meta data of web pages such as abstract and content classification code are often wanted to improve the quality and efficiency of Web applications. For example, in order to improve the efficiency of retrieval, an index of abstract is used to substitute or assistant the full-text index. And content classification is widely used in personalization and focused search. But meta data information is not indicated explicitly in the raw page.

From what has been mentioned above, we can see that preprocessing stage becomes critical for improving the quality and efficiency of Web applications. In the Web page preprocessing stage, global and local noises should be detected and cleaned. We call the process of cleaning local noise *Web page cleaning*, and call the process of cleaning global noise Replica removal. In fact, *Web page cleaning* can improve the accuracy of *replica removal*. So in this work, we first present an approach for Web page cleaning then apply the result to the replica removal process. Another key task in the Web page preprocessing stage is to extract the common required meta data information from the raw Web pages. From the efficiency point of view, the extracting processes of each meta data generally include some analogous intermediate phases, which are usually the bottleneck of the whole process. Such a status is even more serious in Chinese information retrieval, because segmenting Chinese sequences into words is an inevitable and nontrivial task. So, in the preprocessing stage, it's helpful and efficient to extract all the common required information at one time and represent the preprocessing result as a general model (we call this preprocessing step *Web page integration*).

1.1. Our Contribution

In this paper, we propose a new preprocessing framework and approach for Web applications. The tasks designed in our preprocessing framework are motivated to meet the common requirements of Web applications, and integrating all common required functions into a process can improve the efficiency of preprocessing stage. For a Web page collection, through such a preprocessing stage, the Web page collection becomes more refined and the Web pages are integrated into a more general model called DocView, which evolves from DC (Dublin Core[13]) and EAD (Encoded Archival Description[14]). The model consists of the following eight elements: identifier, type, content classification code, title, keywords, abstract, topic content and relevant hyperlinks.

Based on a set of heuristics and traditional Information Retrieval (IR) methods, we present an approach to automatically extract the elements of DocView from raw source pages. We distinguish our approach from prior art in the following important ways: First, our unsupervised approach needs no extra information beyond the raw page, such as the manually specified module of the raw page. Second, it adds no restraint on raw pages; For example, it does not require raw pages to be presented by the same template [11]. While these additional information or restraints are usually necessary for previous related work. The advantage makes our approach more adaptive to web applications, since web page layouts can be very arbitrary.

Totally, the design of the preprocessing framework and the advantages of the corresponding approach make the preprocessing framework and approach more general and efficient. So it can be widely used in Web applications.

1.2. Related Work

The study presented in this paper is proposed to design a more rational preprocessing stage for several typical Web applications. It includes noise (both local noise and global noise) cleaning and meta data extraction. Next, we will respectively illustrate studies of the fields suffering from noise content and lack of meta data.

In web information retrieval (IR) field, we use search engine (SE) as a sample to illustrate the problems caused by noise content. Our work concentrates on two processes: replica removal and indexing. In order to avoid appearing replicas in the retrieval result, an IR system generally contains a replica removal phase before the indexing phase. Three approaches ([12] [19] [20]) are well known for detecting replicated pages, of which [20] is used by Google. All these detection algorithms are based on the same thought: Compute a set of fingerprints for each page. If the amount of identical fingerprints in two pages reaches a threshold, the two pages are considered as a pair of replicas. However, lots of replicated pages on the web are not exact copies of each other, but the same topic content embedded into different templates (we call them *near-replica*). And the noise contents in each template are different. In this condition, two near-replica pages often have few fingerprints in duplicate, thus can't be detected successfully. On the other hand, non-replicas may be deleted by error because of the same noise content in their templates. In the indexing phase, an index file is created in order to match queries with kept documents and retrieve relevant results. Due to the difficulty in ranking results, two best-known algorithms, HITS [8] and PageRank [2], are proposed by analysing the hyperlink structure of the web. Yet as indicated in the appendix described in [2], ranking results are inherently biased toward advertising pages because noise content usually comes up with hyperlinks. In addition, if noise content is not reduced, it will be indexed as well as meaningful topic content. Obviously, search engines need automatic processes to find topic content for indexing and for improving the precision of retrieval. Moreover, reducing noise content can also decrease the size of both page and index file, which is correlative with the efficiency of indexing and retrieval. Such benefits were mentioned by Lin and Ho [11], when they proposed their new approach to discover informative content blocks from a set of web pages generated by the same template (page clustering), and applied it to their news search engine. But their method had restrictions that the entire data set should be presented by the same template. This certainly sets a limit to applications. Most other previous approaches have similar limitations, such as manually generated specification of the page module. These methods are not feasible in case that large amounts and variations of web pages are to be processed.

In web page classification, noise content in training web pages makes the features of category very indefinite, and noise content in classified web page makes the categories of pages very ambiguous. Yang [23] and Li [22] presented several noise reduction approaches to improve the accuracy of classification. But those methods focused on more accurately selecting features from whole page instead of extracting the topic content first and selecting features from topic content.

Information extraction (IE) systems aim to extract structural information from raw pages. Wrapper [9] and SoftMealy [7] are two well known IE systems that extract the structural information from HTML pages based on manually generated templates or examples. One common disadvantage of IE systems ([1] [5]) is that they are often time consuming, corresponding to the complexity of templates

and web pages. It's evident that IE systems will become more efficient by extracting structures from topic contents instead of the whole page.

In meta data extraction field, keyword/keyphrase extraction ([16] [21]) is the focus of active research because of the usefulness of keywords. Of all the keyword extraction approaches, statistical approaches are widely used and proven effective. Formulas such as Tf-Idf can measure the importance of a term through some statistical value. But the HTML language has a lot of useful characteristics that are not fully used. Combining traditional weight-computing formula with HTML characteristics can ameliorate the rationality of weight-computing methods. In traditional abstract extraction approaches, the generation of abstract is based on the extraction of key sentences. In this paper, we share the same idea with the traditional approaches. But, we first identify the key paragraphs of the purified Web page according to the layout of the Web page. Then in each paragraph, we select the sentences with higher weight to generate the abstract of the page. By this approach, the abstract can cover all main information and not be simply leaded by several keywords.

2. Preprocessing framework and DocView model

The preprocessing framework proposed in this paper includes three parts: Web page cleaning (local noise reduction), replica removal (global noise reduction) and Web page integration (meta data extraction and transformation to DocView).

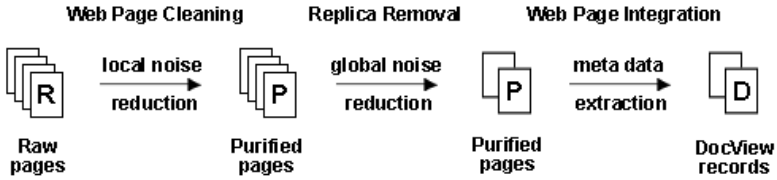

Figure 1 Preprocessing framework overview

From the figure 1, we can see that after Web page cleaning, every page are purified; after replica removal, the replicated pages are removed and Web page collection is refined; after Web page integration, all Web pages in refined collection are integrated into DocView records, which will be more adaptive for various applications.

As the result of the preprocessing stage, DocView model consists of the following eight elements: identifier, type, content classification code, title, keywords, abstract, topic content and relevant hyperlinks. Most of them are metadata, while the latter two are content data. A detailed description of each field is given below.

Identifier is an unambiguous reference to the page within a given context. In our model, we use URL of the page as its identifier.

Type is assigned according to the nature or genre of the content of a page just like the Type element of DC (Dublin Core, [13]). Every page is of one of the following three types: *topic page, hub page and image page*. The so-called *topic page* has a theme that is expressed concretely through text in the page. A news page is a typical topic page. Compared with topic page, the home page of yahoo is a perfect example of another type called *hub page*. Generally, a hub page does not express a topic concretely, but it contains many links to the pages related to this topic. If the main content of a page is

represented by images instead of text, we call this page an *image page*. Pages of different types play different roles on the web and are generally manipulated in different ways. Hence, before any further steps are taken, the type attribute of a page is usually identified first.

Content classification code is the result of classifying a web page semantically, so it always relies on a formal classification scheme and is acquired by a trained classifier. Content classification is one of the most convenient means to obtain the semantic information of a page for computers, thus widely used in text related applications. It is also the recommended expression manner of subject element of DC [13].

Title, keywords and abstract are familiar meta data of documents. They are widely used in information retrieval fields such as digital library.

Topic content is the text expressing the theme of a topic page, the main hyperlinks of a hub page, or the images and their descriptive text in image page.

Relevant hyperlinks are the links to the pages relevant with the topic content.

Topic content and relevant hyperlinks are content data. It's clear that a favourable purified page can be generated by reconstructing topic content into a new page according to the layout of the original page. Taking relevant hyperlinks together is another choice.

3. Page representation

Before extracting DocView from raw page, two abstract representations of the page are required: markup tag tree and quantitative measure.

3.1. Markup tag tree

HTML is a markup language, using tags to structure and describe the texts and images in a page. W3C's Document Object Model (DOM [4]) defines a tree structure for HTML documents, with tags as internal nodes and texts and images as leaf nodes. In our approach, we modify the DOM tree to a tree structure more concentrating on the organization of content. Our modified tag tree focuses on the following two kinds of HTML tags:

Container tags: These tags add structure to documents and partition a page into visual chunks, which may be a tabular structure or a paragraph of text. Based on the assumption that text put in the same paragraph is more content related, we call the chunks "content blocks", and the corresponding tags "container tags". <TABLE>, <TR>, <TD>, <P> and <DIV> are the typical container tags. Content blocks are in one-to-one correspondence with container tag nodes in the HTML tag tree, so in the rest of this paper we will use them interchangeably.

Weighty tags: tags in this category specify display properties in terms of color, font, size or style, such as <B>, <I>, <STRONG>, <H1> and so on. What's more, authors of web pages use tags like these to highlight the plain text embedded in them. In another word, they add weight to words, thus we call them "weighty tags". Due to the nested structure of HTML codes, we can build an HTML tag tree, using only container tags and some in-evitable tags such as <HTML> and <BODY>. Weighty tags, images, text and links are all attached to their corresponding container tag node as attributes. The number of terms, links and images are computed to better describe the content in each node. Moreover, a type field, similar with the type element in DocView model, is assigned to each node (content block). Type is obtained in the following manner. If the proportion of terms in anchor text to all terms in the

block exceeds a threshold, the node is of hub type; if the proportion of images to terms in the block is greater than a threshold, the node is of image type; otherwise, topic type is designated.

3.2. Quantitative measure

In conventional text management fields, a document D is represented by a feature vector W = (w_1, w_2, ..., w_n), where w_i is the weight for the ith feature and n is the total number of features. A feature stands for a word or phrase after deletion of stop words. There are many weighting schemes for determining the weight of a feature. The most generic one is Term Frequency combined with Inverse Document Frequency (TFIDF, [10]):

$$w_i = \frac{tf_i * \log(N / n_i)}{\sqrt{\sum_j (tf_j * \log(N / n_j))^2}} \qquad (1)$$

Where tf_i is the number of times the ith feature occurs in D, N denotes the total number of documents in the collection, and n_i represents the number of documents in the collection in which feature i occurs at least once. In our case, merely counting the occurrences of a feature in the whole page is not enough. As container tags describe the layout of content, they also imply the semantic relation among content blocks. Weighty tags provide information of the importance of the content embedded in them, in our measure, the importance of a weighty tag is scaled by a value named "force" of a tag. Obviously, the force of each weighty tag may be different. For example, the force of <H1> is larger than that of <H3>. To take all factors into account, we introduce block weight into our measure to scale the importance degree of each block. Block weight is initiated and changed by two factors. One is the force of the block, which is the sum of the force of weighty tags in a block; the other is the nesting and neighbourhood relation between blocks.

But before we go into more detail with block weight, we will have to first filter out weighty tags embedding "pseudo weighty" information. Much redundant information such as company logos and advertisement banners are located in weighty tags like <B>, trying to catch more attention, but having nothing to do with the content of the page. We call such redundant information "pseudo-weighty" information. Based on the statistics of 20,000 randomly chosen web pages, 13.2% of the embedded information in weighty tags is noise. Apparently, counting in the influence of noise will harshly mislead the weighting measure. After filtering out weighty tags embedding "pseudo-weighty" information, the block weight of each node can be calculated using algorithm 1.

```
1: for each node n_i in the tag tree do
2:     set the initial block weight bw_i to 1
3: end for
4: for each node n_i in the tag tree do
5:     compute force_i by summing up forces of all weighty-tags in it
6:     if the node contains no weighty tag
7:         set force_i to 1
8:     end if
9: end for
10: for each node n_i in the tag tree do
11:   if n_i is a leaf node
12:       bw_i = bw_i * force_i
13:       parentbw_i = parentbw_i * pforce_i        // parentbw_i is the block weight of parent(n_i)
14:       for each sibling of n_i do
15:           siblingbw_ij = siblingbw_ij * pforce_i  // siblingbw_ij is the block weight of // sibling(n_i)
16:       end for
17:   end if
```

18: end for

Algorithm 1. Computing block weight

Figure 2 gives an example of the process of transferring the force of a block/node and changing weights of related nodes. Here, w_i is the block weight of each node, and f_2 is the force of node 2.

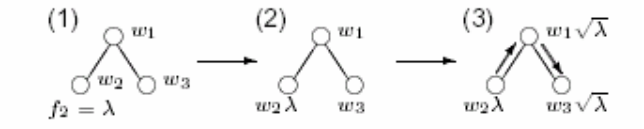

Figure 2. Transferring block force

We state the following claim without proof for lack of space: CLAIM: The distribution of the node force is in one-to-one correspondence with the final distribution of node weight. Based on the block weight generated above, the weight of a feature is now calculated as follows:

$$w_i = \frac{\sum_{j=1}^{BN} BWeight_j * BTf_{ij}}{\sqrt{\sum_{i=1}^{n} \left(\sum_{j=1}^{BN} BWeight_j * BTf_{ij}\right)^2}} \qquad (2)$$

Where BN is the total number of content blocks in a page, n denotes the number of features in a page, $BWeight_j$ is the block weight of node j, and BTf_{ij} the occurrences of feature i in content block j. It can be noticed that the frequency of features is still the chief factor in the formula, while not all high-frequency features are of high importance. So a stop-word list is necessary here, as it is widely used in information retrieval. The generic stop word list is relatively "safe" because it is often much smaller than the vocabularies of real-world document collections, consisting of the most commonly used words which contributes little to algorithms. In our work, from a large web page set randomly crawled from the web, we generate a more aggressive stop-word list by picking up the terms with high term frequency in a lot of documents. We believe that such terms generally have little expressing abilities.

Overall, we can represent an HTML page as a feature vector by the following three steps:

Step1: Compute the block weight of each block (In Algorithm 1)
Step2: Filter out non-meaning terms by stop-word list
Step3: Compute the weight of all features by Formula 2

4. Web page cleaning

Web page cleaning, which is motivated to reduce local noise within a Web page, is also the process of extracting topic content of the page. Based on the markup tag tree, the extraction of topic content and relevant links is a process of pruning or keeping nodes of markup tag tree. As shown in Figure 1, Web page cleaning is the foundation of the preprocessing stage.

As we have mentioned above, every page is of one of the following three types: topic pages, hub pages or image pages. Due to their distinct characters, each should be treated in their own way. In this section, we will first present the method to identify the type of a page, and then concentrate on approaches to realize Web page cleaning automatically.

4.1. Identifying the type of a page

From the description of characters of each type, we can find that the key distinction of the three types rest on the relative amount of terms, hyperlinks and images to each other. As a fact, we also find that if the chunks of the content are different in importance, the important chunks usually occupy the central area of the page while trivial chunks is displayed around them. This layout style also accords with the ordinary browsing manner of a reader. And most trivial content such as advertisement comes up with hyperlinks that will mislead the decision on the type of a page. So, it will be more accurate to identify the page type using central area of page than whole page. Based on the analysis above, we identify the page type with following algorithm that depends on the markup tag tree of the page.

First, the type of each node (see 3.1.) in the markup tag tree is necessary for this algorithm. Then, in the central area of a page, which can be identified by the attributes of the content blocks in most cases, if the proportion of terms in hub blocks (nodes) to terms of all central blocks (nodes) exceeds a threshold, this page is of hub type. The same rule can be applied to identify image page. If the preconditions of the former two rules are not met, it's a topic page.

4.2. Web page cleaning

Given a Web page, the markup tag tree is built first as introduced in section 3.1. In the markup tag tree, every node has a type and the whole Web page also has a type. Based on the markup tag tree, the process of extracting topic content is guided by several heuristics according to the characters of Web page type.

Topic page cleaning: In a topic page, topic content is expressed by paragraphs of text with few hyperlinks and images. So, the topic content is embedded in nodes of topic type. Based on this heuristics, pruning the tag tree of hub node and image node can get a subtree that contains only topic content. We call this subtree "topic content tree". Reconstructing the topic content subtree into a new page according to its original structure thus generates a purified page, a kind of representation of topic content.

Hub page cleaning: In hub pages, topic content is not as clear as that in topic pages. Many hub pages even don't have any topic or have multi-topics. In this condition, extracting the more important hyperlinks in the page is also valuable. Just as what was discussed in Section 4.1, the main content of a page usually occupies the central area of the page while trivial content is displayed around them. So, in a hub page full of hyper-links, we can keep the content in the blocks of the central area as the main hyperlinks of the page.

Image page cleaning: In image pages, the image and the text around image are regarded as the topic content of the page.

To make Web page cleaning result more adaptive to existing systems, we reorganize the topic content into an html page according to its original layout, called topic content page. The topic content extraction algorithm presented in this paper satisfies the following assertion in almost all conditions: If two pages are presented by the same template, their topic content pages also have the same template. This is a prerequisite for DocView to be a general model for preprocessing result of Web applications, especially of applications depending on templates.

The algorithm presented above may leave out some useful information such as the relevant links in the non-topic blocks of a topic page or some useful links in the marginal area of a hub page. So it is

a cleaning with larger granularity. The useful information left out in this step will be picked in the Web page integration process.

5. Replica removal

Generally, replica detection depends on computing similarity of all Web page pairs. Given a large Web page collection, computing similarity of all pairs is too expensive since it would require $O(n^2)$ distance computing. So we first give every Web page a fingerprint according to its content, the size of fingerprint is much smaller than that of the page. Then the algorithm can detect the replicas through comparing all the fingerprints.

Next, we will introduce the algorithm of fingerprint generation. In this algorithm, the fingerprint is generated based on the keywords not on the sentences or paragraphs. As our goal is not only to detect the exact replicas but also to detect the near replicas, the algorithm tries to cover the small difference between two pages by selecting the representative words to generate the fingerprint rather than all the words in the page. In the algorithm, two parameters are defined:

Percentage: the percentage of the words to be selected to generate the fingerprint.
Interval: a tunable unit of the number of selected words. For example, if i*nterval* is 4, the sum of the selected words to generate fingerprints is a multiple of 4.

The algorithm is listed below:

Step1: Represent the page as a feature vector.(the number of the words in the vector is *Sum1*)
Step2: Sort the words in decreasing order of term frequency; the words with the same term frequency are sorted in alphabetic order.
Step3: Select the anterior *Sum2* words from the word list of Step2. (*Sum2*=*Sum1*Percentage*)
Step4: Sort the selected keywords in alphabetic order.
Step5: Select the anterior *Sum3* words from the words list of Step4. (*Sum3*= *Sum2* DIV *Interval* * *Interval*, so *Sum3* is a multiple of *Interval*)
Step6: Call the MD5 algorithm (a message-digest algorithm [17]) to the words selected in Step5 to generate the fingerprint.

During the replica detection process, generating fingerprints for every page is a time-consuming task. Fortunately, in this process, it would require O(n) fingerprint computing. After the generation of fingerprint for all Web pages, we can sort all fingerprints to find out the identical ones. The computational complexity is O(nlogn). What's important is that the fingerprint is only 16 bytes long, so the time spent on the sorting process is much shorter than that on computing similarity of all page pairs.

In fact, the actual system can generate fingerprints for 22 pages per second with single process under the following environment: CPU:PIII 733MHz, Hard disk: Barracuda180 (ST1181677LWV), Memory: PC133 1GB, Mainboard: SuperMacro DL370. The raw pages and fingerprints are all stored using Berkeley DB. The bottleneck of the system is segmenting a page into a feature vector, but segmenting a page into words is an inevitable task for Chinese Web pages.

It can also be seen that the algorithm of near-replica page detection is very sensitive to keywords and their term frequency. However, the noise content in templates not only adds "noisy" keywords, but also changes the term frequency of keywords in topic content. As we have pointed out, in the process of near-replica page detection, confusions brought by the noise content in templates is one of the most serious problems, which misleads the detection algorithm to make a wrong decision. The noise content in templates affects near-replica detection in two ways: on the one hand, replica pages are not identified because the same topic content is embedded in different templates; on the other hand, non-

replica pages are regarded as duplicates because different topic content are embedded in the same template. In order to avoid the negative effects by templates, we use topic content instead of raw page as the input of the detection algorithm.

6. Web page integration

Web page integration is a process of extracting common required meta data from Web page based on the result of Web page cleaning and integrating the extraction result into a general model called DocView. Next we'll illustrate the extraction approach for each element respectively.

Keywords: After grasping the topic content of the page, we can measure the topic content with a feature vector by the method introduced in Section 3.2. Generally, in the feature vector, features with a higher weight have a greater importance. A user tunable parameter (λ) is needed to specify the number of keywords desired. Then, λ features with the highest weights are picked as keywords of the page.

Content category: Content category is acquired by classifying the topic content of the page using a trained classifier. In the process of classification, the feature vector of topic content can be used directly, which is generated in the former phase. It not only enhances the classification accuracy by reducing noise content but also saves time spent on quantitative measure of the page. This is one of the advantages of extracting the whole model at one time.

Title: In HTML pages, title of a page is embedded in the <title> tag. Statistically, about 6 percent of titles are generated automatically by edit tools. The most typical example is "Untitled Document" or "New Page". Obviously, such titles have no meaning; so replacing such a title by one or two the highest weighted keyword is a better choice.

Abstract: In the process of abstract extraction, two heuristics are used. First, the content of a page is organized semantically by paragraphs. Second, readers can quickly catch the ideas of the page with only a small quantity of sentences, and the sentences are generally located by key-words. Based on these heuristics, we can acquire an algorithm that will simulate the browsing manner of a reader to extract the abstract using the keywords obtained in the previous phase. The algorithm first divides the topic content into several paragraphs according to the tag tree, and then the sentences with high weights are picked from each paragraph to make up the abstract of the page. Following, two important subprocesses are illustrated in detail.

1. Identify the paragraphs of the content

Fortunately, the structure of container tags describes the layout of HTML pages, which makes it possible to identify the paragraphs of the content. In the topic content tree, first locate the node that is the lowest common ancestor of all leaf nodes (we call it topic root). Topic root corresponds with the tag that exactly embeds all topic content nodes. Then, the son nodes of the topic root correspond with the paragraphs of the topic content. Figure 3 shows the process of identifying paragraphs. In (3) of Figure 3, each p block is a paragraph.

2. Extract key sentences from each paragraph

First, two user tunable parameters are required: δ and θ. δ indicates the number of sentences to be extracted for the abstract. θ indicates the number of keywords that will be used to compute the weight of the sentences. Keywords with higher weight will be selected preferentially because of their greater expression ability.

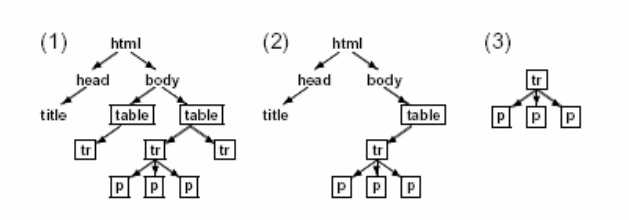

Figure 3. Process of identifying paragraphs

Then according to the size of each paragraph, we can obtain the number of sentences to be extracted in each paragraph (δ_i). In every effective paragraph ($\delta_i > 0$), each sentence is assigned a weight to evaluate its importance according to the weights of selected keywords in it. At last, the δ_i sentences with the highest weight are extracted. These high-weighted sentences make up the abstract.

Relevant hyperlinks: The method of relevant hyperlinks extraction relies on following the two heuristics: First, anchor text of a hyperlink expresses the content of the referenced page concisely. Second, the related hyperlinks in a page are generally listed adjacently, furthermore, they are usually embedded in one content block.

Correspondingly, we can acquire the following two rules: First, we can measure the pertinence of hyperlinks using their anchor text. Second, we can use content block as a unit to accept or reject hyperlinks.

In a page of topic type, the process of relevant hyperlinks extraction is also a process of accepting or rejecting hub type blocks. We use the standard cosine similarity [18] (3) between the text in a hub block and the topic text to measure the pertinence of hyperlinks in this hub block.

$$d(X, X') = \frac{\sum_{i=1}^{n} x_i x'_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} x'^2_i}} \tag{3}$$

The relevant hyperlinks extraction algorithm is given in following.

```
1: the feature vector of the topic content is φ
2: for every leaf block CB_i do
3:     if CB_i is of hub type then
4:         compute the feature vector of CB_i (let it be φ_i)
5:         compute the similarity between φ and φ_i (let it be similarity_i)
6:         if similarity_i > β then (β is a threshold for similarity)
7:             accept the hyperlinks in CB_i
8:         else      reject the hyperlinks in CB_i
9:         end if
10:    end if
11: end for
```

In conclusion, the process of the preprocessing stage can be illustrated by Figure 4.

7. Applications and experiment study

In this section, we perform three applications and experiments. First, we apply Web page cleaning approach to our web page classification system to verify the validity of the approach. Then we design a experiment to measure the accuracy of replica removal approach. At last, we apply the abstract

element of DocView to our search engine (Tianwang, see [15]), through the improvement of quality and efficiency we can see the accuracy of abstract extraction. Because the extraction of abstract is based on the extraction of keyword, we also can verify the validity of extraction of keyword. Next, we illustrate them in detail.

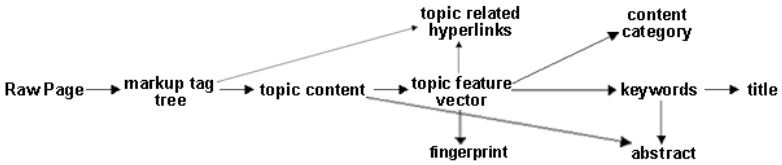

Figure 4. the process of preprocessing stage

7.1. Application in web page classification

We design the experiment based on such logic: reducing noise content in training web pages makes the feature of each category more definite, and reducing noise content in classified web page makes the category of the page more evident. So, local noise reduction can improve the performance of classification.

First, we introduce the original classification system. The original classification system relies on a topic taxonomy having 739 topics arranged in a tree hierarchy with depth at most 3. The top hierarchy includes 12 top categories. There are 12456 pages in the training set and 3114pages in the test set. Our classification system uses χ^2 [25] as its feature selection algorithm and KNN [24] as its classification algorithm. In this application, we make no change to the algorithms of feature selection and classification, but only substitute the topic content for the raw page in training process and classifying process. Hence, we train the classification system on the new training set by the same algorithm and test it using the new test set.

We apply three measures widely used in evaluating the performance of IR systems to the experiments and verify the quality of our proposed methods. They are precision rate (4), recall rate (5) and F1 value (6), which is a general evaluation based on recall and precision.

$$precision_i = \frac{\#of\ pages\ assigned\ to\ category\ \text{i}\ correctly}{\#of\ all\ pages\ assigned\ to\ category\ \text{i}} \tag{4}$$

$$recall_i = \frac{\#of\ pages\ assigned\ to\ category\ \text{i}\ correctly}{\#of\ pages\ that\ belongs\ to\ category\ \text{i}} \tag{5}$$

$$F_{1i} = \frac{2 * precision_i * recall_i}{precision_i + recall_i} \tag{6}$$

From the result shown in Figure 5, we can see that the F1 value of every category has increased. This indicates that the noise content is reduced successfully and substituting topic content for full text is reasonable. Because we make no change to the algorithms of feature selection and classification, the improvement of classification performance is the contribution of noise reduction.

In practice, the effect of noise reduction is greater than what is shown in Figure 5, because pages with little noise content is preferred when we select training page and test page manually.

ID	Category name
01	Art
02	News and Media
03	Business and Economy
04	Entertainment
05	Computer and Internet
06	Education
07	Region
08	Natural Science
09	Government and Politics
10	Social Science
11	Medical and Health
12	Society and Culture

Table 1. Category Id and Name

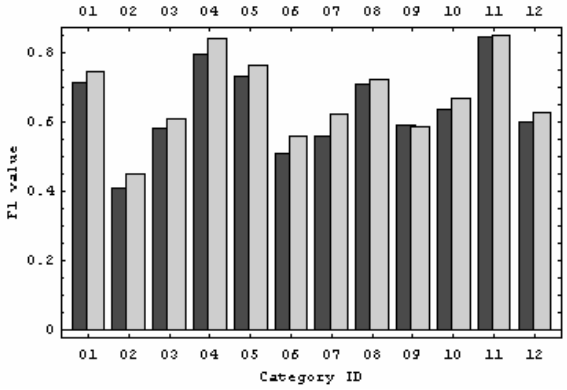

Figure 5. Classification Comparison

7.2. *Experiment of replica removal*

According to the replica removal approach presented in section 5, we can know that the exact copies can be easily detected without exception. So, the detection of near-replicas is the key issue of any replica detection approach. In this experiment, we focus on the detection of near-replicas, and measure the accuracy of the detection approach based on three manually labelled Web page sets: base set, near-replica set and non-replica set. Base set contains 125 randomly chosen web pages, each of which has a topic content. Near-replica set contains the near-replicas of every page in the base set, but the topic content in these replicas are presented by different templates compared to their corresponding pages in the base set. Every page in non-replica set shares the same template with a page in the base set, but its topic content is basically different. The near-replica set is used to evaluate the ability of the replica detection approach to find near-replicas presented by different templates. The non-replica set is used to show whether the approach is deceived by non-replicated pages with identical templates.

From the algorithm introduced in Section 5, we can see that the quality of the algorithm is controlled by two parameters: *Percentage* and *interval*. The experiment result is shown in Table2. For every element in the Table2, the value before '/' is the number of pages in the near-replica set detected as near replica, named *hit rate*; the value after '/' is the number of pages in the non-replica set regarded as replica wrongly, named *error rate*. Through the experiment result shown in Table2, we can get the following two rules about the two parameters.

	P=10%	P=20%	P=30%	P=40%	P=50%	P=60%	P=70%	P=80%	P=90%	P=100%
I=1	82/29	61/6	47/1	35/0	33/0	25/0	23/0	23/0	16/0	16/0
I =2	107/9	95/6	64/3	46/0	33/0	28/0	28/0	24/0	19/0	17/0
I =3	109/9	99/7	62/3	55/1	49/0	41/0	41/0	30/0	21/0	17/0
I =4	112/10	99/7	72/4	67/2	51/1	46/0	45/0	31/0	27/0	28/0
I =5	125/10	102/7	75/5	72/2	66/1	53/0	52/0	48/0	41/0	32/0
I =6	112/10	93/7	70/4	61/1	53/1	47/0	42/0	42/0	31/0	28/0
I =7	110/10	93/6	70/4	59/1	53/1	46/0	40/0	41/0	31/0	27/0
I =8	101/9	92/6	69/3	51/1	43/0	29/0	28/0	24/0	24/0	17/0
I =9	97/8	85/5	63/3	59/1	49/0	46/0	40/0	35/0	26/0	26/0
I =10	82/8	83/5	60/2	47/0	43/0	42/0	37/0	33/0	29/0	26/0

Table 2. relation between replica detection quality and two parameters

Rule1: For every *interval* value, the *hit rate* and *error rate* all decrease with the rising of *percentage*.

Rule2: For every *percentage* value, the *hit rate* and *error rate* increase first and then decrease with the rising of *interval*. The hit rate reaches the maximum when *interval*=5.

In fact, most of the replicas on the Web are exact copies rather than near-replicas. The exact copies can be detected by our algorithm easily. So in the actual system, we set *interval*=5 and *percentage*=0.4. In such a condition, of the 125 near-replicas in near-replica set, 72 near-replicas are detected successfully. The *hit rate* of the approach is 58%. And of the 125 non-replicas in non-replica set, 2 pages are regarded to be replicas wrongly. So the *error rate* of the approach is 1.5%. Through the service provided by our search engine, we can see that the quality of replica detection can meet the general use.

7.3. Application in search engine

We apply the abstract element of DocView to the indexing stage of our search engine. In the indexing stage, we create an index of the abstract element of DocView, in addition to the original index of full text. After providing service using the two indexes, we get the precision of the top 10 results returned by each.

In the process of abstract extraction, two parameters are required to tune the size of the abstract: the number of sentences making up the abstract (δ) and the number of key-words used to affect the weight of sentences (θ). In our prototype system, δ=15, θ=15. Our index was physically organized as an inverted file. In this condition, two important values about the indexes are shown in Table 3.

The sum of terms in all abstracts is 35.28% of that of full text, while the size of abstract index is 41.4% compared to full text.

	Abstract	Full text	ratio（abstract/full text）
Sum of terms	8.3641e+08	2.37055e+09	0.3528
Size of index file	3370054685	8139460811	0.4140

Table 3. Important values about index

In order to evaluate the disk access load of the two indexes, we randomly picked 6,000 query terms from the log file, and used every 100 queries terms as a group to simulate query requests. The ratio of disk access loads of the two indexes is displayed in Figure 6.

From Figure 6 we can see that the average disk access load of abstract index is approximately 45% of that of full text.

At last, the precision of top ten results of the two indexes is evaluated using TREC-inspired methods [6] and a set of 50 queries taken from the log of our search engine. The judgment of our volunteers shows that compared with a 41.2% precision of full text index, the precision of abstract index is 62.5%.

Above results tell us, using the index of abstract not only enhances the efficiency of the system, but also improves the precision of results. As a cost of the advantages, the number of query results has decreased. The results also verify the accuracy of the extraction of abstract. Based on abstract index, we can not only provide service for abstract retrieval to improve precision of result, but also create a

hierarchical index, with abstract and full text indexes laid in different hierarchies, to enhance the efficiency.

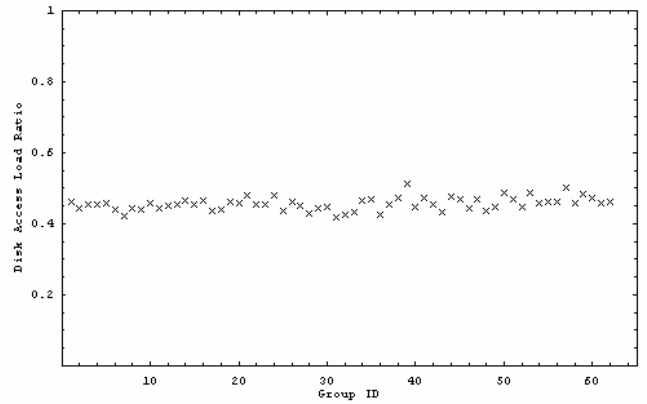

Figure 6. Comparison of Disk Access Load (abstract/full text)

8. Conclusions and Future Work

In this paper, we proposed a preprocessing framework and approach to meet the common requirements of several typical web applications. After such a preprocessing, the Web page collection is refined and the Web pages are integrated into a general model, which is more adaptive for database operation. So, using the preprocessing approach given in this paper, we can set up a well formed, purified, easily manipulated information layer on top of any Web page collection (including WWW). We carried out 3 independent applications and experiments to evaluate our approach. From the results, both validity and practicability of the approach was proven. Besides the framework and approach, we made some significant modifications to several basic intermediate phases. All of the enhanced methods can be adopted in related research. For instance, our quantitative measure for HTML documents can be employed in web page classification and web information retrieval. The improved HTML tag tree is more adaptive to web page content analysis.

As the development of the web continues, new problems may be introduced to applications, thus a rational preprocessing stage may change its task and framework. How to extract more common required information without sacrificing efficiency is our next pursuit. Besides, how to represent the result of preprocessing stage with high generalization and scalability is another problem to address.

References

1. N. Ashish and C. A. Knoblock. Wrapper generation for semi-structured Internet sources. In Proceedings of the Workshop on Management of Semistructured Data, Tucson, 1997.

2. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems, 30(1-7):107-117, 1998.

3. Lan Yi, Bing Liu, Xiaoli Li. Eliminating noise information in Web pages for data mining. SIGKDD, 2003.

4. DOM. http://www.w3.org/dom/.

5. J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting semistructured information from the web. In Proceedings of the Workshop on Management of Semistructured Data, pages 18-25, May 1997.

6. D. Hawking, N. Craswell, P. Bailey, and K. Griffihs. Measuring search engine quality. Information Retrieval, 4(1):33-59, 2001.

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

8. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. Journal of the ACM, 46(5):604-632, 1999.

9. N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In Intl. Joint Conference on Artificial Intelligence (IJCAI), pages 729-737, 1997.

10. D. D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka. Training algorithms for linear text classifiers. In H.-P. Frei, D. Harman, P. Sch"auble, and R. Wilkinson, editors, Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval, pages 298-306, Z"urich, CH, 1996. ACM Press, New York, US.

11. S.-H. Lin and J.-M. Ho. Discovering informative content blocks from web documents. SIGKDD, 2002.

12. U. Manber. Finding similar files in a large file system. In Proceedings of the USENIX Winter 1994 Technical Conference, pages 1-10, San Fransisco, CA, USA, 1994.

13. Dublin Core. http://dublincore.org/documents/dces/.

14. Encoded Archival Description. http://lcweb.loc.gov/ead/.

15. Networks Lab, Peking University. http://e.pku.edu.cn/.

16. T.-H. Ong and H. Chen. Updateable pat-tree approach to chinese key phrase extraction using mutual information: A linguistic foundation for knowledge management. In Proceedings of the Second International Conference of Asian Digital Library, pages 63-84, Taipei, Taiwan, November 1999.

17. Rfc1321. http://www.faqs.org/rfcs/rfc1321.html.

18. G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. Information Processing and Management, 24(5):513-523, 1988.

19. N. Shivakumar and H. Garc'ia-Molina. SCAM: A copy detection mechanism for digital documents. In Proceedings of the Second Annual Conference on the Theory and Practice of Digital Libraries, 1995.

20. N. Shivakumar and H. Garcia-Molina. Finding near-replicas of documents on the web. In WEBDB: International Workshop on the World Wide Web and Databases, WebDB. LNCS, 1999.

21. I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin, and C. G. Nevill-Manning. KEA: Practical automatic keyphrase extraction. In ACM DL, pages 254-255, 1999.

22. L. Xiaoli and S. Zhongzhi. Innovating web page classification through reducing noise. Journal of Computer Science and Technology, 17(1), January 2002.

23. Y. Yang. Noise reduction in a statistical approach to text categorization. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, Proceedings of SIGIR-95, 18th ACM International Conference on

Research and Development in Information Retrieval, pages 256-263, Seattle, US, 1995. ACM Press, New York, US.

24. Y. Yang and X. Liu. A re-examination of text categorization methods. In M. A. Hearst, F. Gey, and R. Tong, editors, Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval, pages 42-49, Berkeley, US, 1999. ACM Press, New York, US.

25. Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In D. H. Fisher, editor, Proceedings of ICML-97, 14th International Conference on Machine Learning, pages 412-420, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.