An Implementation of Web Image Search Engines

Zhiguo Gong, Leong Hou U, and Chan Wa Cheang

Faculty of Science and Technology, University of Macau,
P.O.Box 3001 Macao, PRC
{zggong, ma36575, ma36600}@umac.mo

Abstract. This paper presents our implementation techniques for an intelligent Web image search engine. A reference architecture of the system is provided and addressed in this paper. The system includes several components such as a crawler, a preprocessor, a semantic extractor, an indexer, a knowledge learner and a query engine. The crawler traverses web sites in multithread accesses model. And it can dynamically control its access load to a Web server based on the corresponding capacity of the local system. The preprocessor is used to clean and normalize the information resource downloaded from Web sites. In this process, stop-word removing and word stemming are applied to the raw resources. The semantic extractor derives Web image semantics by partitioning combining the associated text. The indexer of the system creates and maintains inverted indices with relational model. Our knowledge learner is designed to automatically acquire knowledge from users' query activities. Finally, the query engine delivers search results in two phases in order to mine out the users' feedbacks.

1 Introduction

In recent years, huge amount of information is published on WWW and it continues to increase with an explosive speed. However, we cannot access to the information or use it efficiently and effectively unless it is well organized and indexed. Only after that, people can be possible to perform efficient browsing, searching and retrieving on the resources. Nowadays, we know that "Google" is accepted as the biggest and fastest search engine on WWW. Web users are becoming more and more reliable on this tool. Although "Google" has also provided image-searching function based on its existed technology, such as indexing and ranking, it still has some space for improvement. For instance, a Macau resident wants to find some images about the University of Macau with a keyword--"umac" in his image searching of "Google", we can find a "Luna" at the first page of the result, and it is not relevant to the query. Therefore, there may be much improvement space for web image retrieval technologies.

Image Retrieval has been a very active research area ever since 1970's, with the thrust from two major research communities, Database Management and Computer Vision. These two areas address image retrieval techniques in different ways. One is text-based and the other is visual-based. On the web, most of popular Image Search Engines (For example: Google) are text-based. The reason is due to the fact that visual-based image retrieval can only work fine in some specific application domains. That means the web images are needed to be classified with respect to different domains before visual processing. Two methods can be used for the classification—manually or automatically. The former requires vast amount of labor and even impossible since the huge size of the web. Our system adopts the latter model.

In this paper, we are going to describe the implementation techniques in our text-based image retrieval system. We provide a reference architecture for web image retrieval systems, which includes a crawler, preprocessor, semantic extractor, indexer, knowledge learner and a query engine. We implement the crawler in a multithread model, therefore it can dynamically control its access load to a Web server based on the corresponding capacity of the local system. The preprocessor is used to clean and normalize the information resource downloaded from Web sites. In this process, stop-word removing and word stemming are applied to the raw resources. The semantic extractor derives Web image semantics by partitioning then combining the associated text. The indexer of the system creates and maintains inverted indices with relational model. Our knowledge learner is designed to automatically acquire knowledge from users' query activities. Finally, the query engine delivers search results in two phases in order to mine out the users' feedbacks.

In section 2, we describe our reference architecture of the system. And in section 3 to section 7, we will address our implementation technologies on the crawler/preprocessor, semantic extractor, indexer, knowledge learner and search engine respectively. And finally, we conclude this paper in section 8.

2 System Architecture

Figure 1 shows the fundamental working principles of our system. Firstly, web pages are collected by the crawler and delivered as a stream to the preprocessor of the system. After parsed by the preprocessor, the intermediate results, including all the web images, associated text, links and their relationships, are loaded into the document database. And the new links are passed back to the crawler for recursively web page gathering.

The semantic extractor takes the associated texts as input and represents them into DOM tree format. Then, it tries to partition the associated texts formally into a sequence of semantic blocks based on the element structures as well as their distances to the embedded images. In our system, the semantic relevant factor of each semantic block is measured with respect to its relative position to the corresponding image. For each pair of analytic image and its associated text, the semantic extractor parses the document and produces a set of postings, where each posting is an entry which shows the term's semantic relevance to the web image. The indexer sorts all the postings, then, creates an inverted index to support fast accesses to the images.

In terms of the knowledge base in our system, we consider users search activities as important resources to improve the retrieval performances of the system. Most of current web image retrieval systems can only provide static schemas, which are based only on limited sample data or experts' experiences, in their indexing the web images. Thus, such systems can not evolve to catch the dynamic changing of the WWW.

From users' query log, the knowledge learner in our system tries to mine out the semantic relevances between web images and query terms. In this way, retrieval performance of the system can be well improved. Furthermore, the system can recalculate the semantic distribution schemas of the web images by using extended sample space.

Our knowledge base also captures and maintains frequently used phrases or concepts in the user queries. And they can be used in the future semantic extraction processing and indexing.

For the query engine, it provides two phase deliveries of the query results. In the first phase, only indexing images (the small thumb images generated from the original images) are delivered to the users. And the second phase presents the original images or its owner web pages with users' further requests. In this way, users' searching activities can be captured and passed to knowledge learner. We will introduce each process in detail in the following sections.

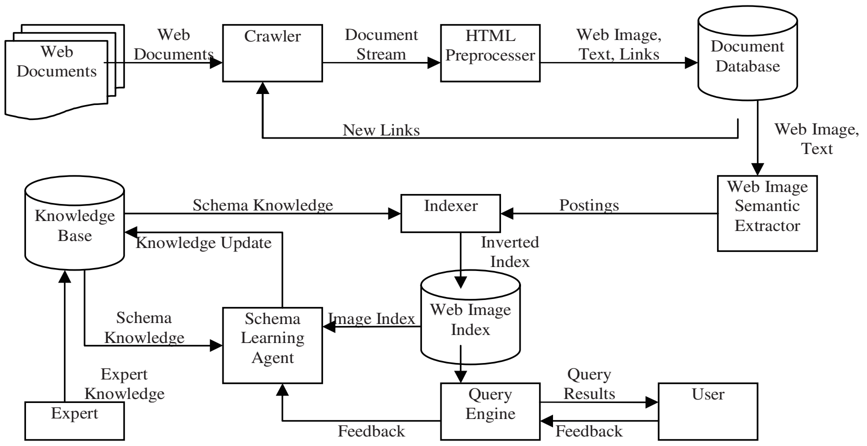

Fig. 1. The System Architecture Overview

3 The Crawler and HTML Preprocessor

Crawler is one of the important components in web information retrieval system. It gathers web pages from WWW and recursively fetch the new pages with the new URLs which are contained in the load documents. Generally, two strategies exist for web page traversing—BFS(breath first search) and DFS(depth first search). And BFS is exploited by our crawler.

In order to work, the crawler needs to maintain a queue of URLs, and it always selects URL from the top of the queue to start a new page traversing. And the newly gathered URLs are appended to the end of the queue. Because the web sites are autonomous and independent in nature, they may have different workloads and response capabilities to the crawler. For example, if the accesses received are more than the limitation of the host being accepted, the host may shut down (just like the host is attacked by Distributed Denial of Service attack).Therefore, it is not wise if the crawler only work on one site at one time. To solve this problem, our crawler is designed to employ I/O multiplexing or multithreading to allow web pages to be fetched from multiple sites at the same time. It is obvious that more threads being run at the same time can raise the system throughput. And this model can also leave more processing capabilities of the local sites for their local requests.

For this reason, the crawler needs to control the thread number at each accessed host. To reach this destination, our system separates the global URL queue into subqueues according to different hosts. Each subqueue of a individual host has a running count variable that records number of the running threads at the site. If the running count of a queue is less than the threshold N (For example: N=10), the crawler will pop up the top URL from this subqueue and goes on fetching state. Otherwise, it will try the next host's queue. The worst case occurs when all the hosts' running counts are more than the limitation, then crawler must switch to sleep state.

Because the network traffic keeps dynamic changing, the access threshold of each host can not be fixed and it is hard to be determined manually. Our solution assumes that all the thresholds of each host are the same at the initial state. Based on the change of the transmission time, crawler can adjust the value dynamically. One situation is that the transmission time increase continuously. That means the performance of this host is low and is busier than before. Then, crawler reduces the limitation value of N.

All the crawling URLs are converted into 32 bytes MD5 string first, then, written into database. To prevent any page from being reloaded, any new URL is compared with the existing MD5 string in the database. Then the URL is inserted into the queue. The web images and their associated pages are stored and managed with file system. We allocate web images and pages by the value of a harshing function of its URL. And the relationship with its parent page is also maintained in the database.

The Preprocessor contain two components: stemming processor and stopword processor. Stemming processor transforms terms or words in the texts into normal form. In our system, we uses a stemming algorithm by Porter, M.F [4], which is a popular algorithm used by many systems. This algorithm uses five steps to convert an informal English word into a formal one. For more detail, refer to [4].

Stop word processor is used to remove meaningless words, such as 'the', 'a', 'and', from the documents. In our implementation, we use the list provided by [2] as our set of stopwords. Of course, this list may not be the optimal one and we may extend it in the future.

Crawler is the fundamental component for web information retrieval systems. It takes charge of the document gathering. Besides universal crawlers, there are also many specific crawlers available in both the commercial market and academic communities [6,12]. For such kind of crawlers, topic relevance of each URL should also be taken into account when scheduling.

4 The Semantic Extractor

The semantic extractor takes the associated text of an image as input and tries to mine out the semantics of the image. The retrieval performance of the system is closely related to the correct extracting work of the semantic extractor. In this section, we address our implementation techniques for our extractor in detail.

4.1 The Problem

Images on the web can be classified into different types according to their purposes, such as (1) icon, (2) site logo, (3) realistic photo, and (4) cartoon pictures.

To simplify the problems, our system tries to removes stop-images (meaningless images) such as icon, web site logo before performing the image semantic extractions. In our approach, we use some measurements, such as the sizes of the images and the ratios between the width and the height of the images, as the criteria to discriminate whether an image is a stop –image or not.

Even though other solutions, such as link based, visual feature based, are also used for web image semantic extraction, text based methodology is the most fundamental one. Our semantic extractor is based on the associated texts of the images. Many TBIR researches [3, 6] use ***title***, ***alt of the analytic image***, ***filename***, and ***surrounding text*** as the sources for the semantic extractions. However, we should pay attention to some problems in using the texts.

- How to determine the optimal range of the surrounding text?
- In general, we know that the terms which has the smaller distance to the image have higher relevances to the image. But this linearization distance method shows some defect as that the linearization distance inside the HTML document is different from what we can view with web browser.

4.2 Text Fragmentation

We know that many HTML documents often present information with a lot of HTML tags, however the traditional HTML document model is hard to be used to mine out such layout structures automatically. We need to use other document object model to replace the traditional one. One favorite model for representing the HTML structure is document object model tree (DOM tree) [5]. Based on the result by [9], we build the DOM tree using <TABLE> template tags: <TABLE>, <TR> and <TD>. After this transformation, the HTML structure becomes more easily to process. In general, the distance between the blocks and the analytic image rendered by the browser well match with that organized in the DOM tree. So we assume that the nearer blocks are more related to the analytic image. To formalize the distance between the analytic image and the associated text blocks, the DOM tree will be partitioned properly (Fig. 2).

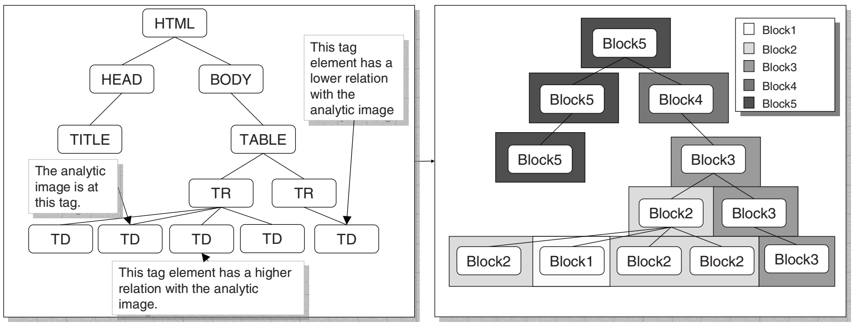
HTML
HEAD
BODY
TITLE
TABLE
TR
TR
TD
TD
TD
TD
TD
The analytic image is at this tag.
This tag element has a lower relation with the analytic image
This tag element has a higher relation with the analytic image.
Block5
Block5
Block4
Block5
Block3
Block2
Block3
Block2
Block1
Block2
Block2
Block3
Block1
Block2
Block3
Block4
Block5

Fig. 2. DOM tree split into different blocks

Our Semantic Extractor uses MSHTML library, which is one of the Microsoft COM interface, to parse the HTML document into DOM tree. It builds the tree only by using these three tags: <TD>, <TR> and <TABLE>. Each tag contains its inner text and the relationship with its parent and child elements. For the characteristic of HTML document, we use Depth First Search (DFS) to build the tree. It scans the document from start to end. The inner text which contains between two of these tags is determined by this scan. Each node and its inner text will be grouped into difference blocks later. The following is the pseudo code for this processing:

```
STRING Trace_Dom_Tree (ParentNode)
{
  ExtractInnerText=ParentNode.GetInnerText();
  ForEach (NewChildNode) in ParentNode
  {
    If (NewChildNode.TagName == (<TABLE>||<TR>||<TD>))
    {
      ParentNode->ChildNode.Add(NewChildNode);
      ExtractInnerText.Remove(
            Trace_Dom_Tree(ParentNode->ChildNode));
    }
  }
  ParentNode.SetInnerText(ExtractInnerText);
  Return ExtractInnerText;
}
```

Our system uses a set of sequence numbers to determine the distance between the analytic image and the associated text elements. The sequence number is set by Semantic Extractor during building the tree. (Fig. 3) Each sequence number of the nodes is based on its parent and its sibling order under the same parent tag. In Fig. 3, the sequence number of analytic image's parent is 00 and its own order is 1, so the sequence number of analytic image is 001. After assigning the sequence number for each node, our system compares the analytic image sequence with others. The comparison checks the difference of the position of sequence number only. All the situations are showed as follows:

- The position of different sequence number (*P*) is equal to or less than the length of the associated text sequence number (L_{at}). Then the distance (D) is

$$D = L_{at} - P + 1 \tag{1}$$

- No any different sequence number find, but the length of associated text sequence number (L_{at}) is less than the analytic image (L_{ai}). Then the distance (*D*) is

$$D = L_{ai} - L_{at} \tag{2}$$

- No any different sequence number is found and the length of associated text sequence number (L_{at}) is equal to or longer than the analytic image (L_{ai}). Then, the distance (*D*) is

$$D = 0 \tag{3}$$

Fig. 3 shows the distance for each node by the above calculation. Based on this, we partition the document into different semantic blocks.

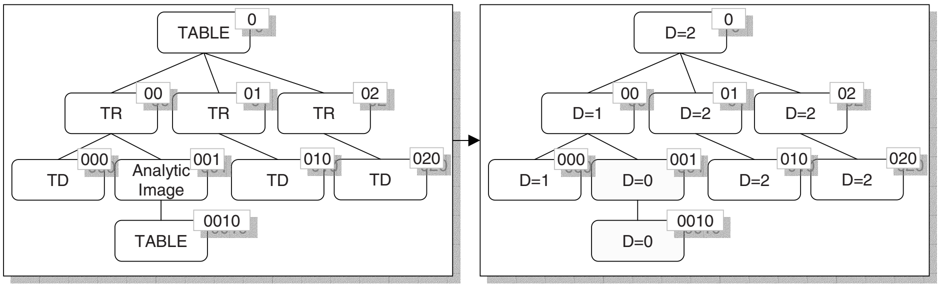

Fig. 3. Giving Sequence Number to each node

4.3 Distance Adjustment of Semantic Blocks

If the partitioning is only based on the absolute distance captured from the DOM tree as in last subsection, we find that the fragmentation can not properly reflect the visual rendering in many cases. As a matter of the fact, the same visual distances may tagged in the file differently by different authors. For instance, one author may like to write the nearest associated text within the same block as the analytic image block but others may not. Using the absolute approach as in section 4.2, the associated text of the first one is more related than the second one. However, they look the same when rendering in the browser. Since this confusion, in our approach we measure the position by relative distance (Fig. 4). We assume that the associated text appeared at the first block is the nearest block. So the above example will consider both of the associated texts to be the most related text.

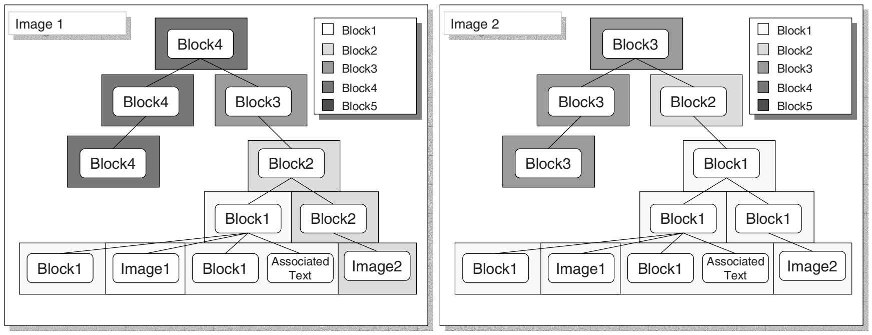

Fig. 4. DOM tree split into different blocks based on relative position

The above approach only works fine if the document only contains one image. However, there are quite a lot o documents containing more than one images. If the above images are in the same document, our approach considers that the importance between the associated text and the two analytic images is the same. But we know that the first image is closer to the associated text than the second one. It means that the relationship of the first image and the associated text is higher than the second one. So we need to adjust the blocks number to solve the problem. As doing this amendment, we need to analyze all useful images in the same page.

Suppose we have a set of analytic images $\{I_1, I_2 \ldots\ldots I_n\}$ in the HTML document. Each text block $\{T_1, T_2 \ldots\ldots T_m\}$ has a distance $\{D_1, D_2 \ldots\ldots D_n\}$ with each analytic image. So each analytic image I_i is corresponded to a set of distances $\{D_{i1}, D_{i2} \ldots\ldots D_{im}\}$. Based on the order of the distance, each analytic image I_i has a sequence block number $\{BN_{i1}, BN_{i2} \ldots\ldots BN_{im}\}$.

We define the semantic blocks with the following property in the same document:

$$(BN_{ik} == BN_{jk} \text{ iff } D_{ik} == D_{jk}) \text{ AND } (BN_{ik} < BN_{jk} \text{ iff } D_{ik} < D_{jk})$$

In the same document, the blocks number is the same if and only if these blocks have the same value of the distance. And the smaller blocks number also have a smaller value of the distance. In Fig. 4, we know that the block number between the associated text and the two analytic images is the same, but their distances to the associated text are difference. Because of this property, we need to make following adjustment.

The adjustment has three steps: (1) Sort the sequence block number first. That means BN_{ik} is smaller than or equal to $BN_{i(k+1)}$ (2) If ($BN_{ik} == BN_{jk}$ AND $D_{ik} > D_{jk}$), then we adjust BN_{ik} to $BN_{ik} + 1$. (3) Then, we continue to find all BN_{il} ($l>k$) and adjust all BN_{il} to $BN_{il} + 1$. Go back step (2) until no any ($BN_{ik} == BN_{jk}$ AND $D_{ik} > D_{jk}$). After these adjustments, we get a more reasonable distance measurement in splitting the DOM tree. The result shows on the Fig. 5.

With the approach above, we partition the inner text of HTML document into different blocks successfully. We combine these blocks with other semantic blocks: ***title***, ***alt of the analytic image*** and ***filename*** to generate a set of semantic blocks $\{SB_1, SB_2, \ldots\ldots, SB_n, SB_{title}, SB_{alt}, SB_{filename}\}$. Now, we need to know how important for each block to a embedded image. We will use the statistical result to calculate the weight of each block. And it will be discussed in detail later.

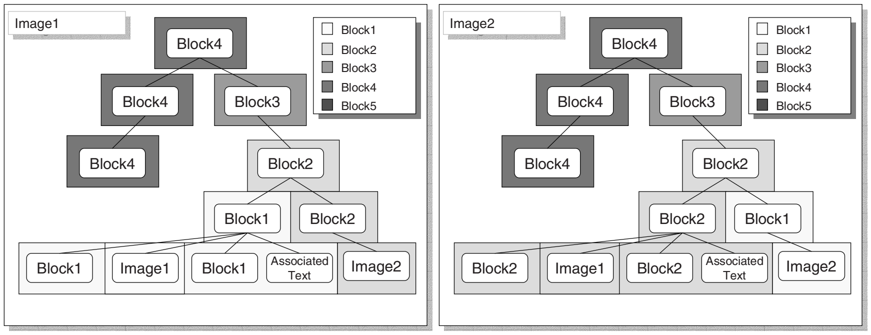

Fig. 5. The modified method for splitting the DOM tree

4.4 Term Weight Calculation

After we split the document into semantic blocks, Semantic Extractor finds out the weight of each term. In TFIDF model, the term of t's semantic relevance to page p is inverted document frequency of term *t*. In our system, we use terms (or concepts) of *p* to derive semantics of the Web image *i*. However, the above TFIDF approach can not be directly applied to index the embedded image.

In this paper, we modify TFIDF model as the following equation:

$$ntf(t)|_{SB_j} = \frac{tf(t)|_{SB_j}}{|SB_j|} \tag{4}$$

where $ntf(t)|_{SU_j}$ is called normalized frequency of term *t* over SB_j, $tf(t)|_{SB_j}$ is the frequency of term *t* over SB_j, and $|SB_i|$ is the size of SB_i. Thus, the total term frequency over the whole *p* is obtained as

$$ttf(t)|_p = \sum_{1 \le i \le N-1} w_j * ntf(t)|_{SB_j} \tag{5}$$

where *N* is the total number of the semantic blocks, and w_j is the weight of SB_j in implying the semantics of the embedded image *i*. Without loss of generality, we suppose w_j's are normalized, such that $\sum_{1 \le i \le N-1} w_j = 1$.

In our approach, $ttf(t)|_\mathrm{p}$ indicates the semantic relevance level of term *t* to the embedded image *i* of page *p*. Then the indexer creates an inverted index based the values of $ttf(t)|_\mathrm{p}$. We will discuss how to determine the values for w_j later.

5 The Indexer

Semantic Extractor passes the relation between the analytic image, the relationship of associated text and all the term frequency to Indexer. Like many other information retrieval systems, we use inverted index in our system. For performance, the table is built of a one-to-many relationship. Like the following table:

Term	Index
TERM1	DOC_ID_1, DOC _ID_2,., DOC_ID_N
TERM1	DOC _ID_N+1, DOC_ID_N+2
TERM2	DOC_ID_1, DOC_ID _2, DOC_ID _3

To increase the database performance, the length of the index has an optimal upper bound – N. If one term is corresponding to more than an upper bond documents, it will use more than one row to store the document ids. In our system, the index does not store only the document id but also stores some related information, such as the frequency of the term for each semantic block ($ntf(t)|_{SB_j}$).

For performance, we know that the cost of I/O access is higher than the cost of memory access, so our system stores the inverted index into memory first. When the index number is up to upper bound value, it writes the index from memory to database and clears the memory content. In order to reduce the total disk capacity, it stores inverted index to database in binary format. (Fig. 6)

6 The Knowledge Base

Knowledge Base is a subsystem which tries to capture the knowledge from experts and user feedbacks. In section 4, we have not determined the values of w_j yet. As we discuss before, our system use the statistical result to determine their values.

In the area of information retrieval, precision/recall is well accepted evaluation method which indicates the performance of the systems. In order to calculate the precision/recall value, our system needs the expert to provide some training set. Our system provides a user-friendly interface, to let the experts define the corresponded meanings for each image easily by using the mouse.

Since the result of a retrieval are usually long in size, especially in the World Wide Web environment, a figure of precision versus recall changing is commonly used as a performance measurement for a retrieval algorithm. However, this metric can not be used as an objective function in determining those weight values. We use the average precision concept instead of the above metric. The average precision is defined as

$$AP_{jk} = \frac{1}{R_{jk}} \sum_{k=1}^{R_{jk}} \frac{k}{N_k} \tag{6}$$

where R_{jk} is the total number of all relevant results with respect to $ntf(q_i)|_{SB_j}$ and N_k is the number of results up to the k-th relevant result in the image result list. As a matter of the fact, AP_{jk} is the single value metric which indicates the performance of querying q_k by only referencing semantic block SB_j. The optimal value of w_j is determined by the overall average precision objective defined as:

$$AP = \frac{1}{N} \sum_{i=1}^{N} AP_i(w) \tag{7}$$

where $AP_i(W)$ is the average precision for each semantic block. Because the higher value of AP represents the higher quality of the result. So we try to adjust the value of w_j for each semantic block to get a higher AP value. After this step, we get the optimal value of w_j. This is an iterative improvement. After we add some new record to our result set, we calculate the new optimal value of w_j based on the old one.

Knowledge base also handle the user's feedbacks. It uses a two steps model to handle about it. Like most of the current image retrieval systems, we display the search results in two steps: (1) indexing thumbs of the original images are delivered to the users (2) original images or the web pages which contain the original images are sent to users with users' further request command. The promise of using our users' feedbacks is that the second activity is performed by a user implies the result is likely to be the correct answer of the query. We store the users' feedbacks into Knowledge Base and display the result with the new rank function:

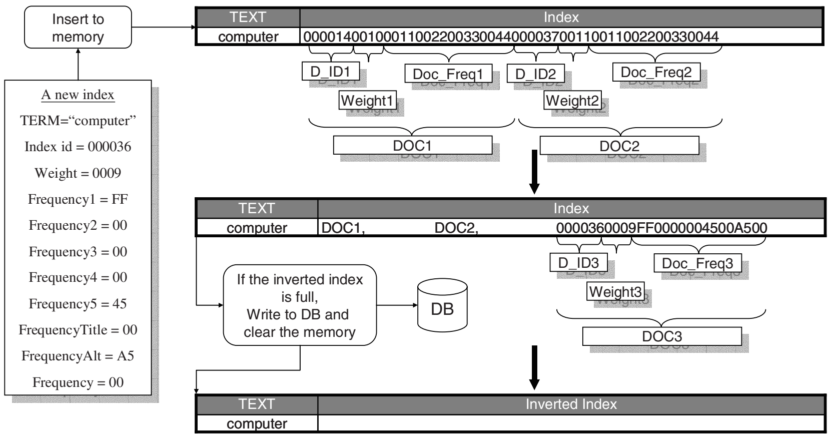

Fig. 6. Example of indexing

$$adj(t,i)=w_{d*}ttf(t,i)+w_{u*}srl(t,i) \tag{8}$$

where *ttf*(*t,i*) is the term frequency for each semantic block which is discussed in equation 5, *srl*(*t,i*) is term-image relevant values obtained from user query activities (its details addressed in our other papers, w_d and w_u are weights for objective function *ttf* and *srl* respectively, and w_d+w_u=1. We can use the same method as to calculate the w_j by average precision to determine these two values.

7 The Query Engine

Query Engine supports concept based searching function to the user. When it accepts a term from the user, it presents the relevant images to the user. For performance, the index is sorted by Indexer first. For one term searching, the system delivers the thumbs to user directly. But this sorting can not be satisfied with two terms or more. For two-term searching, the result set *R* is the intersection of those term result. We need to sort *R* for each searching because *R* is an unsort set. The sorting operation is time consuming even if we have already exploited a good sorting algorithm. So the sorting operation must decrease the system performance. One method to solve the problem is to store the *i*-term concept which may be from 1 to N to database. But we know that the number of two-term concept has already been very large. It is impossible to store all of these into database because since the physical limitation and the performance of the system.

In our approach, the database only stores some important N-term concepts which are captured by the user feedback. Because our system records all user queries, we will know which concept is more likely to search. Our system just stores the N-term concept, which frequency is higher than the predefined value, into database. How to balance the physical storage size and the performance is the problem of database management. But it is out of the range of this paper, we will not discuss in detail. Fig 7 shows an example of our retrieval interface.

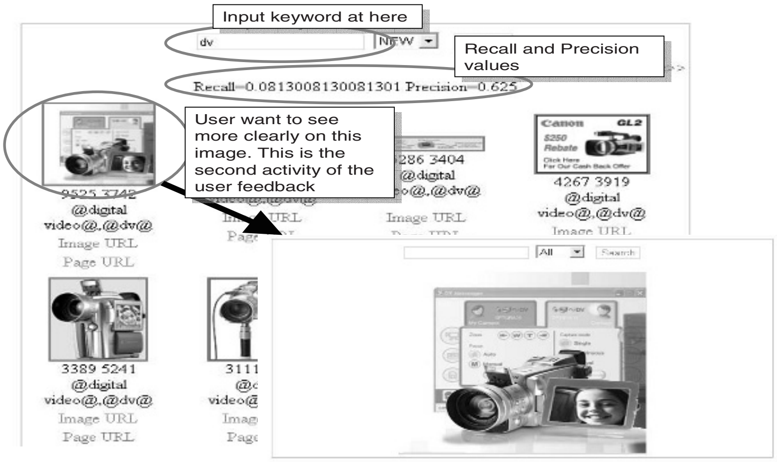

Fig. 7. One example of Search Interface

8 Conclusions and Future Work

In this paper, we provide a comprehensive survey on our implementation technologies of the text based image retrieval system. This paper introduces our methodologies on each essential components of our system, such as Crawler, HTML Preprocessor, Semantic Extractor, Indexer, Knowledge Learner and Query Engine in detail.

In our system, we partition the HTML document into a set of semantic blocks. It means our method only focus on analyzing the text which is inside one HTML document. We consider this model as a single level analysis of link structure. In web environment, the parents and the child nodes of HMTL document may have significant meanings to the extract the semantics of the analytic image. For this objective, we can try to expand the analysis to multiple levels.

Some link structure algorithms have developed on web environment, such as Hyperlink Induced Topics Search (HITS) or Google's Pagerank. And there existed some excellent surveys on link structure image retrievals already[7]. In our future work, we will try to combine our existed method with the link structure to get a better result.

The other improvement of our system is to use an electronic thesaurus (WordNet^{TM}) to expand both the user queries and the meta-data associated with the images. We want to get a model based on objects, attributes and relationships to allow us to use WordNet^{TM} in a comprehensive automatic manner and to improve retrieval effectively.

References

1. S. Chakrabarti, M. V. D. Berg and B. Dom: Focused Crawling: a New Approach to Topic-Specific Web Resource Discovery. *Computer Networks* 31(11-16), 1999, pp. 1623-1640.

2. S. Chakrabarti, M. V. D. Berg and B. Dom: Focused Crawling: a New Approach to Topic-Specific Web Resource Discovery. *Computer Networks* 31(11-16), 1999, pp. 1623-1640.
3. Shi-Kuo Chang and Arding Hsu: Image information systems: Where do we go from here?. *IEEE Trans. on Knowledge and Data Eng.*, 4(5), Oct. 1992, pp. 431-442.
4. Chen, Z. et al.: Web Mining for Web Image Retrieval. To appear in *the special issue of Journal of the American Society for Information Science on Visual Based Retrieval Systems and Web Mining*.
5. DOM: http://www.w3.org/DOM/
6. Zhiguo Gong, Leong Hou U and Chan Wa Cheang: Web Image Semantic Extractions from its Associated Texts. *The 8th IASTED International Conference on Internet & Multimedia Systems & Applications*, Kauai, Hawaii, USA, August 16-18, 2004.
7. V. Harmandas, M. Sanderson, M.D. Dunlop: Image Retrieval By Hypertext Links. In: *Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval* 1997.
8. Mingjing Li, Zheng Chen, Hongjiang Zhang: Statistical Correlation Analysis in Image Retrieval.*, Pattern Recognition* 35, 2002, pp. 2687-2693.
9. Lin, Ho: Discovering Informative Content Blocks from Web Documents. *ACM SIGKDD 2002*, Edmonton, Alberta, Canada, July 23 - 26, 2002.
10. Porter, M.F.: An Algorithm For Suffix Stripping. *Program 14 (3),* July 1980, pp. 130-137.
11. Stop Word List: http://www.searchengineworld.com/spy/stopwords.htm
12. Hideruki Tamura and Naokazu Yokoya: Image database systems: A survey. *Patt. Recog.*, 17(1), 1984, pp. 29-43.