A Personal System for Web Image Retrieval

Noriyuki Fujimoto and Kenichi Hagihara
Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka, 560-8531, Japan
{fujimoto,hagihara}@ist.osaka-u.ac.jp

Abstract

This paper describes the design, implementation, and evaluation of a novel Web image search engine software. The hardware resources required by our engine are only one commodity personal computer (PC for short) and only one Internet connection with a few Mbps. For a given query and a given time interval, our engine retrieves Web pages relevant to the query during the time interval. Then, our engine analyzes the collected Web pages and finally enumerates images in the Web pages in the descending order of score for the query. We evaluated our engine on 2.4GHz Intel Pentium 4 PC with 512MB RAM and 4.28Mbps effective bandwidth to the Internet. Our experiments show that ,due to our sophisticated Web crawling algorithm, image discard algorithm, and image scoring algorithm, our engine can typically select 349 images of 2205 images in 192 Web pages in 33.38 seconds.

1 Introduction

Web image retrieval is important because it allows the user to find images of interest. Existing Web image retrieval systems can be classified into two categories, i.e., content-based and text-based. This paper focuses on text-based Web image retrieval. There are many such text-based systems, e.g., Google Image Search [4], Altavista Image Search [12], Lycos Multimedia Search [8], Picsearch [14], and alltheweb [11]. In order to respond quickly for user's unpredictable query from all over the world, an existing system keeps a huge image database. The database records which Web page includes what images and what keywords. The database also records the strength of the relation between each image and each keyword. So, existing image search engines require expensive hardwares, e.g., huge disk space for image database, many PCs to build and maintain the image database, and another many PCs to quickly respond users' queries.

On the other hand, Moore's Law [9] states that the transistor density on integrated circuits doubles every couple of years. The growth of the transistor density is connected with the growth of computing power of a microprocessor. Also, Gilder's Law [3] states that the communication bandwidth of optical fiber networks grows at least three times faster than the growth of the computing power. These exponential growths result in increased performance and decreased cost of both a PC and Internet connection in the home. Now, many people have a few GFLOPS PC and at least a few Mbps Internet connection in their home. So, a single commodity PC has the prospect to be a cost effective Web image retrieval system if effort to realize such a system is done.

This paper describes the design, implementation, and evaluation of a novel Web image search engine software. The hardware resources required by our engine are only one commodity PC and only one Internet connection with a few Mbps.

The remainder of this paper is organized as follows. First, we give an overview of our personal system for Web image retrieval in Section 2. Next, we present our Web crawling algorithm, images discard algorithm, and image scoring algorithm in Section 3, Section 4, and Section 5 respectively. Then, in Section 6, we describe the implementation of our system. Last, in Section 7, we show some experimental results on the performance of our system.

2 An Overview of our System

Fig. 1 shows a screen shot of our system. The text box labeled with 'Query' in upper left corner is given a query by a user. A query is usually a sequence of keywords that represent what images are relevant to user's interest. Then, our system performs 'AND-retrieval' for given keywords. In fact, a query may include several operators besides keywords. Possible operators are '+', '-', 'site:', 'related:', 'link:', '~', and ' '' '(double quotations that surround a phrase). The meanings of these operators are the same as ones of Google Web Search [6]. The text box labeled with 'Collect Time' in upper left corner is given a time interval by a user.

For a given query and a given time interval, our system retrieves Web pages relevant to the query until the time interval is elapsed or the stop button is pushed by a user. Then, our system analyzes the collected Web pages and enumerates the images in the Web pages in the descending order of score for the query. Thus, our system builds keywords-to-images database on demand focusing only on the Web pages relevant to the given query. Therefore, no huge disk space is required by our system.

The retrieved images are displayed page by page. In Fig. 1, one page includes 12 images. A user can move page to page using 'Previous Page' button and 'Next Page' button in the bottom of the window. The tab sheet labeled with 'Search Results' in Fig. 1 includes top 12 images of the search result for query 'Neuschwanstein Castle' and time interval 30 seconds. Notice that these 12 images are all relevant to the query 'Neuschwanstein Castle'. Each image in the tab sheet is shrunken within 200 pixels $\times$ 200 pixels if the original image is larger. In that case, the original large image can be seen in a popup window if a user clicks the shrunken image. If a user clicked a hyperlink below an image labeled with 'included in this Web page', then the Web page that includes the original image is popped up. 'Score :' below an image represents the score of the image for user's query. When retrieval is finished, the status bar in the bottom of the window in Fig. 1 reports the total elapsed time, the total number x of analyzed HTML pages, the total number y of images included in the x HTML pages, and the total number of relevant images of the y images. Fig. 1 shows that, in the case of query 'Neuschwanstein Castle' and time interval 30 seconds, our system on 2.4GHz Intel Pentium 4 PC with 512MB RAM and 4.28Mbps effective bandwidth to the Internet collected 349 relevant images from 2205 images in 192 Web pages in 33.38 seconds.

3 The Proposed Algorithm for Focused Crawling

Consider in what order a crawler should visit the URLs it has seen, in order to obtain more important pages first. For this purpose, PageRank [13] is known as an excellent ordering metric [2]. To compute PageRank, we must complete to crawl all the pages in advance. However, during crawling, we have only a subset of objective pages. So, we cannot know precise PageRank

during crawling. For this problem, Page et al. proposed to guess PageRank using the subset of objective pages [2]. In contrast, this paper adopts quite a different approach. The proposed algorithm simply asks Google Web Search about the PageRank ordering. The following is a pseudo code of the proposed algorithm:

```
Algorithm Focused Crawling
Input: query Q, time interval T
begin
  start = 1; n = 50; clear(url_queue); clear(hot_queue); clear(crawled_pages);
  while (not specified time T elapsed) and(not stop button pushed) do begin
    if (empty(url_queue)) begin
      url_queue = Google(Q, start, n); if (empty(url_queue)) exit;
      start += n;
    end
    top_page = dequeue(url_queue); enqueue(hot_queue, top_page);
    while (not empty(hot_queue)) do begin
      url = dequeue(hot_queue); page = download_page(url); register_image(page);
      enqueue(crawled_pages, url);
      if (keywords in Q appears in page except in anchor texts) begin
        url_list = extract_urls(page);
        foreach u in url_list do
          if (u not in hot_queue) and (u not in url_queue) and(u not in crawled_pages))
            if (u and top_page are in the same site) enqueue(hot_queue, u);
      end
    end
  end
end
```

Function call Google(Q, start, n) asks Google Web Search to retrieve Web pages (not Web images) relevant to query Q. Then, Google Web Search returns URLs of Web pages such that PageRank ordering is from start to start+n-1. Procedure call register_image(page) registers only a subset of the images in page using our image discard algorithm described in Section 4.

Our originality is that our algorithm does not traverse a hyperlink if the HTML page which is currently analyzed does not include any keyword in the area except anchor texts. This is because each anchor text represents the content of not the current page but the destination page. So, an HTML page which includes keywords only in anchor texts tends to be irrelevant to the query. In particular, news sites and blog sites strongly exhibit this tendency. An exception is a page that includes many links to the pages relevant to the query. However, such a page seems to include no relevant images while the page itself is relevant to the query. If such hyperlinks are traversed, an explosive number of irrelevant pages are crawled. Such explosion exhausts the precious bandwidth of the Internet connection to crawl irrelevant pages.

4 The Proposed Algorithm for Image Discard

Our engine does not collect all the images in a crawled Web page. That is, during crawling, our engine discards images which seem to be obviously irrelevant to a given query as follows.

Thumbnail images are commonly used to reduce unnecessary communication load. Usually, the corresponding full size image can be obtained by clicking the thumbnail image. To discard thumbnail images is important because not thumbnail images but the corresponding full size images are of user's interest. Consider an HTML page which includes <A href="url"><IMG>

</A>. If url points out an image in the same site as the HTML page, then <IMG> is discarded because it is regarded as a thumbnail image of the image. If url points out an image in another site, then <IMG> is discarded because it is regarded as a banner advertisement, a non-original image, or a thumbnail image which symbolizes the image.

Another devised point of our algorithm is to discard images smaller than 70 pixels $\times$ 70 pixels and images with aspect ratio less than 1/3 or greater than 3. Most small images are icons or decorations. Even if not so, too small images seem to be not of interest. Most images with too small or too large aspect ratio are decorations or banner advertisements.

5 The Proposed Algorithm for Image Scoring

For a given HTML document H, our image scoring algorithm parses H and builds the parse tree of H. Then, our algorithm gives a score to every collected image according to both the closeness to nearby keywords and weight of every tag that surrounds each keyword. A pseudo code of the proposed algorithm is shown below:

Algorithm Image Scoring
Input: a set H of crawled HTML documents, query Q
Output: score $S(i)$ of every collected image i in H for Q
begin
  foreach HTML page h in H **do begin**
    foreach keyword occurrence k in h **do begin**
      tag_weight(k)=1.0;
      foreach HTML tag t in Table 1 **do**
        if (t includes k) tag_weight(k) += weight of t according to Table 1;
    end;
    foreach image i in h **do**
      foreach keyword k in Q **do** $S(i,k) = 0$;
    foreach keyword occurrence k in h **do**
      foreach image i in h **do**
        $S(i,k)+ =((2.0/(1 + dist(i,k)) \times tag\_weight(k))$
          $+baseScore(h,k)/$(number of collected images in h));
  end
  foreach HTML page h in H **do**
    foreach image i in h **do**
      foreach keyword k in Q **do** Normalize $S(i,k)$ between 0 and 1;
  foreach HTML page h in H **do**
    foreach image i in h **do**
      foreach keyword k in Q **do** $S(i) = (\sum_k S(i,k))/$(number of keywords);
end

Our originality is to measure the closeness between an image and a keyword according to the distance on the parse tree, not the linear distance on the text file (i.e., the HTML file). Function call $dist(i,k)$ in the pseudo code counts the distance between image i and keyword k as follows: The deepest node in a tree that is an ancestor of two given nodes is called the lowest common ancestor (LCA for short) of the two nodes; Let $LCA(i,k)$ be the LCA of i and k; Let d_i (d_k, resp.) be the number of nodes from i (k, resp.) to $LCA(i,k)$; Then, $dist(i,k)$ is defined as $d_i + d_k$. Table 1 shows our weight of tags. Our weight improves La Cascia et al.'s weight [1] in that our algorithm gives weight also to <b>, <i>, , , <font>, , and <ins> HTML tags.

Table 1: Weight of HTML tag

tag	title	h1	h2	h3	h4	h5	h6	b, i, em, strong, font, center, ins	others
weight	5.0	4.0	3.6	3.35	2.4	2.3	2.2	3.0	0

Another our originality is the notion of 'base score'. Function call $baseScore(h, k)$ in the pseudo code computes the base score of HTML page h for keyword k. $baseScore(h, k)$ is computed as follows: If h is in the result of Google(Q, start, n) in the pseudo code in Section 3, then $baseScore(h, k)$ is zero for every k; Otherwise, let h' be the HTML page such that our crawling algorithm arrived at h because h' has a hyperlink to h; $baseScore(h, k)$ is equal to $S(i, k)$ (before normalizing) where i corresponds to a virtual <IMG> at the position of <A> of the hyperlink. The term of the base score has the effect to give a score to an image such that the image is the only contents of the HTML page that includes the image. Such an image is pointed out by the corresponding thumbnail image in another HTML page.

When at least two keywords are given, first of all, our algorithm computes a score for every keyword. Then, our algorithm combines the scores into the final score by normalizing each initial score between 0 and 1 and by summing up them. This is because such a combining method is known to be good in spite of its simplicity [7].

6 Implementation Notes of our Engine

Our Web image retrieval software is implemented on Microsoft Windows XP. Our engine is multi-threaded so as to overlap communication for focused crawling with computation for HTML parsing and image scoring. Such overlap effectively hides communication delay. Our default number of threads is 80. Also, we use asynchronous socket API (i.e., WSA* functions) in Winsock 2.0 API [10] to set an arbitrary timeout period for socket communication. Our default timeout period is 2 seconds.

To detect small images and images with illegal aspect ratio is realized by using 'Range Retrieval Request' function of HTTP protocol version 1.1 to download only the header part of an image file. So, our detection does not require to download an entire image file. Notice that our focused crawling algorithm, image discard algorithm, and image scoring algorithm never need to download an entire image file. To download an entire image file is required only when our system displays the final search results page by page. Therefore, communication load of our system is relatively light.

To know PageRank ordering, we simply ask Google Web Search about it. To implement this, we did not use Google Web API [5]. Instead, we implemented meta-search like code to get Google Web Search results to be parsed using libxml2 [15]. This is because Google Web API has severe limitation that we cannot issue more than 1000 queries per day (A query cannot extract more than 10 URLs).

7 Experiments

We evaluated our engine on 2.4GHz Intel Pentium 4 PC with 512MB RAM and 4.28Mbps effective bandwidth to the Internet using queries described in Table 2. Notice that whether a given image is relevant or not to each of the queries can be told objectively. We did not use

Table 2: The used queries and their relevant images

query	relevant images (i.e., user's interest)
Neuschwanstein Castle	Photos or pictures of the Neuschwanstein Castle. Not only the whole view but also inside views (a specific room, a small building inside the castle wall, and so on).
Siberian husky	Photos, pictures, or toys of Siberian husky dogs.
amber insect	Photos of ambers with insect inclusions. Ambers without insect inclusions are not accepted.
Audrey Hepburn	Photos or pictures of actress Audrey Hepburn.

Table 3: The performance of our system in case of query 'Neuschwanstein Castle'

time interval (sec)	30	60	120	300	600
parsed HTML	192	464	674	1009	1401
total images in the parsed HTML pages	2205	2517	4671	8016	14930
selected images	349	361	699	970	1584
discarded thumbnail images	969	1232	2491	4924	9026
discarded external images	471	473	778	1107	2906
discarded small or illegal aspect ratio images	416	451	703	1015	1414
total time (sec)	33.38	65.96	132.92	329.27	656.97

subjective or ambiguous queries. We gave our system keywords in English, as you see. Since our experiments were made in Japan, most Web pages which were retrieved and analyzed by our system were outside Japan.

Table 3 shows how many images our system could collect in 30, 60, 120, 300, and 600 seconds. The number of threads is 80 and the timeout period of a socket communication is 2 seconds. It turns out that our system can collect enough many images within reasonable amount of time. Notice that our algorithm discards more images than the selected images. Most of discarded images are irrelevant or too small images. This implies that our image discard algorithm is effective.

Table 4 shows how many images of the top 500 images are relevant to the query in case that a given time interval is 600 seconds. For query 'Neuschwanstein Castle', the precision of images with upper ranks is better than the precision of images with lower ranks. This implies that our scoring algorithm tends to push up the ranks of relevant images. This property seems to be desirable as a retrieval system. For the other queries except 'amber insect', the precisions are kept high. This is because the saturation point of the precision is lower than 500. So, this does not contradict with the precisions in case of query 'Neuschwanstein Castle'. The reason why some precisions are lower than neighboring precisions is that a chain of irrelevant images from the same Web page are appeared in the results. This implies that our scoring algorithm should attach much greater importance to the closeness between an image and a keyword.

The precision of the result for query 'amber insect' is relatively low than the other queries. For query 'amber insect', many of the images ranked in the one hundreds are images of insects only or images of amber without insect inclusion. This query is different from the other queries in that each keyword in the query is quite different from each other. This result means that there remains room for improvement in the scoring algorithm in such a case.

Recall that our crawling algorithm does not traverse a hyperlink if the HTML page which is currently analyzed does not include any keyword in the area except anchor texts. Table 5

Table 4: The precision of our system (The time interval is 600 seconds.)

query	Neusch-wanstein Castle	Siberian husky	amber insect	Audrey Hep-burn
precision of 1st - 100th images	70%	93%	54%	91%
precision of 101st - 200th images	40%	80%	34%	77%
precision of 201st - 300th images	39%	75%	43%	95%
precision of 301st - 400th images	18%	86%	57%	59%
precision of 401st - 500th images	8%	95%	62%	80%

Table 5: The decline in the precision in case that all the hyperlinks are traversed (The time interval is 600 seconds.)

query	Neusch-wanstein Castle	Siberian husky	amber insect	Audrey Hep-burn
precision of 1st - 100th images	33%	52%	3%	90%
precision of 101st - 200th images	6%	18%	8%	77%
precision of 201st - 300th images	8%	3%	54%	86%
precision of 301st - 400th images	6%	0%	55%	51%
precision of 401st - 500th images	0%	2%	67%	76%

shows the precision in case that all the hyperlinks are traversed. It turns out that our crawling algorithm significantly improves the precision in many cases.

8 Conclusion

We have verified that our crawling algorithm, image discard algorithm, and image scoring algorithm enable a single commodity PC with a few Mbps Internet connection to build a personal system for Web image retrieval.

References

[1] M. L. Cascia, S. Sethi, and S. Sclaroff. Combining textual and visual cues for content-based image retrieval on the world wide web. In *IEEE Workshop on Content-Based Access cf Image and Video Library*, pages 24–28, 1998.

[2] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through url ordering. In *the 7th International WWW Corference*, 1998.

[3] G. Gilder. *TELECOSM: How Infinite Bandwidth will Revolutionize Our World*. Free Press, 2000.
[4] Google. Google image search. *http://image.google.com/*.
[5] Google. Google web apis (beta). *http://www.google.com/apis/*.
[6] Google. Help document: Google advanced search. *http://www.google.com/help/refinesearch.html*.
[7] J. H. Lee. Analyses of multiple evidence combination. In *20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–276, 1997.
[8] Lycos, Inc. Lycos multimedia search. *http://multimedia.lycos.com/*.
[9] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.
[10] L. Napper. *Winsock 2.0*. John Wiley & Sons Inc, 1997.
[11] Overture Services, Inc. alltheweb. *http://www.alltheweb.com/*.
[12] Overture Services, Inc. Altavista image search. *http://www.altavista.com/image/default*.
[13] L. Page and S. Brin. The anatomy of a search engine. In *the 7th International WWW Conference*, 1998.
[14] Picsearch. picsearch. *http://www.picsearch.com/*.
[15] D. Veillard. The XML C parser and toolkit of Gnome: libxml. *http://www.xmlsoft.org/*.

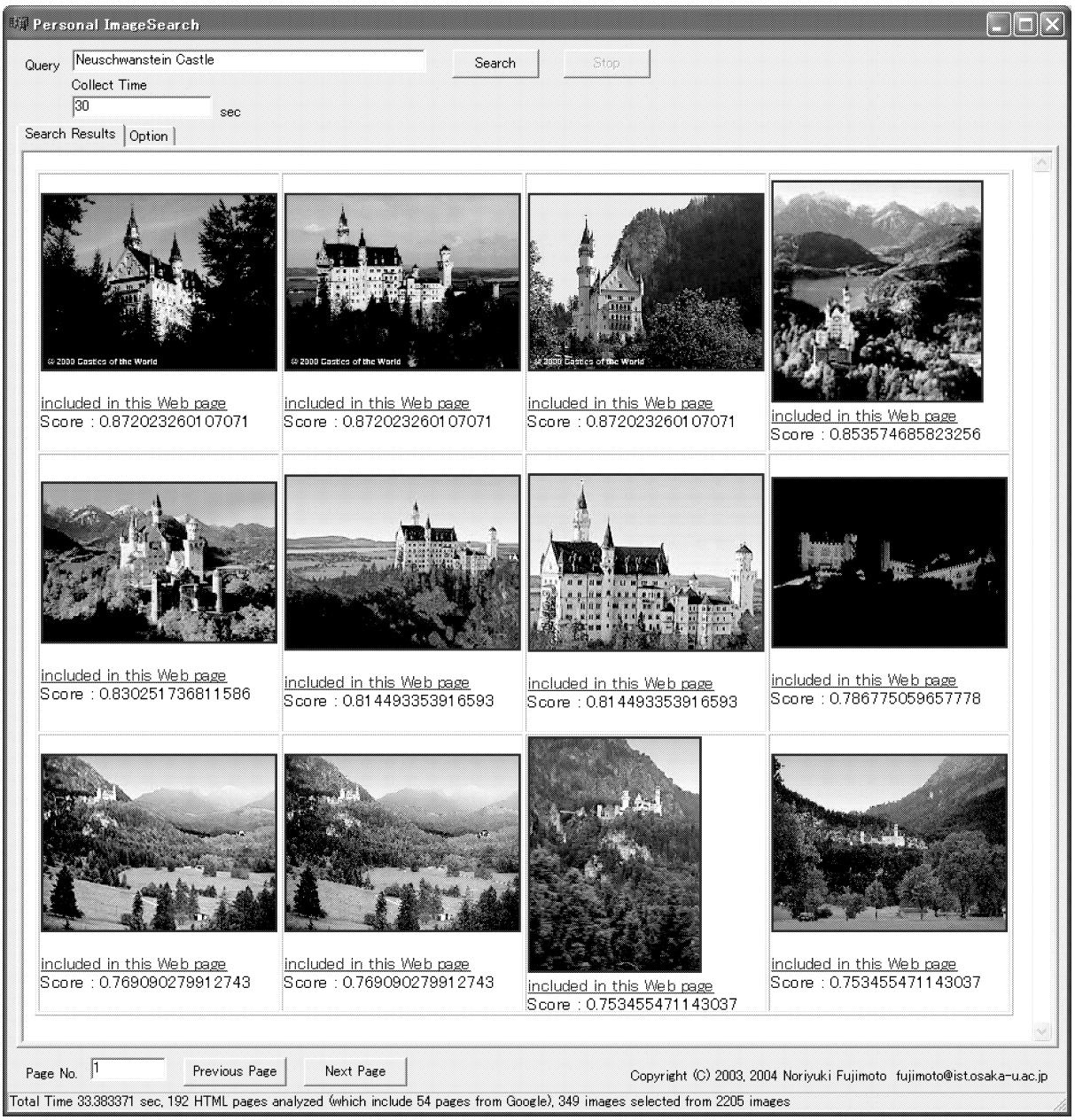

Figure 1: A screen shot of our Web image search system