WISE: A Content-based Web Image Search Engine[#]

G. Qiu^{*} and R. D. Palmer[♦]

School of Computing
The University of Leeds
Leeds LS2 9JT, United Kingdom

Abstract: This paper describes the development of a prototype of a Web Image Search Engine (WISE), which allows users to search for images on the WWW by image examples, in a similar fashion to current search engines that allow users to find related Web pages using text matching on keywords. The system takes an image specified by the user and finds similar images available on the WWW by comparing the image contents using low level image features. The current version of the WISE system consists of a graphical user interface (GUI), an autonomous Web agent, an image comparison program and a query processing program. The users specify the URL of a target image and the URL of the starting Web page from where the program will “crawl” the Web, finding images along the way and retrieve those satisfying a certain constraints. The program then computes the visual features of the retrieved images and performs content-based comparison with the target image. The results of the comparison are then sorted according to a certain similarity measure, which along with thumbnails and information associated with the images, such as the URLs; image size, etc. are then written to an HTML page. The resultant page is stored on a Web server and is outputted onto the user’s Web browser once the search process is complete. A unique feature of the current version of WISE is its image content comparison algorithm. It is based on the comparison of image palettes [6] and it therefore very efficient in retrieving one of the two universally accepted image formats on the Web, “gif”. In gif images, the colour palette is contained in its header and therefore it is only necessary to retrieve the header information rather than the whole image, thus making it very efficient.

Keywords: Search engines, content-based image indexing and retrieval (CBIR), image database, query by example.

1. INTRODUCTION

Traditionally, images have been gathered into particular types of collections, which enables particular indexing methods suitable for the specific collections to be developed. Users can then query the collection using a standardised approach. With the advent of the World Wide Web, a large unordered collection of images have become available to many different types of users, along with a wide variety of image formats, image quality, image descriptions, etc. As such, trying to apply normal specific search methods to this would be unfeasible, as it would be likely to encompass only a small sub-set of the images available on the World Wide Web (e.g. only a small amount of images have a text description provided by the image author, so a text based search method would exclude all images without this description). Therefore a method that allows for the retrieval of images by matching any set of criteria chosen by the user, independent of image category or intended usage, would be beneficial to all users, as it would not be reliant on any additional information added by the image author.

One method is to query images based on their contents, whereby the contents of the images are automatically indexed in some way by a program. A combination of low level features, such as colour, shape, texture, etc., and high level descriptions of the image content can be used for comparing images [5]

[#] To view images appear in the paper in colour, a pdf version is available online at

<http://www.cs.nott.ac.uk/~qiu/Online/Publications.html>. WISE Website: www.comp.leeds.ac.uk/wise

^{*} Current affiliation: School of Computer Science, University of Nottingham, Jubilee Campus, Nottingham NG8 1BB, United Kingdom. Email: qiu@cs.nott.ac.uk

[♦] Current affiliation: Thus plc, 310 Vincent Street, Glasgow, G2 5BB, UK. Email: richard@magicality.org

Search engines are the main method of finding out information about a subject on the millions of un-catalogued pages on the World Wide Web. However, they are limited, both in size (as new web pages are constantly added and old web pages are updated, it is impossible to store the entire Web) and the method they allow users to search with. The majority of search engines use a basic technique of searching through their database of web pages to find matches with text keywords entered by the user. There are few search engines that allow searching on things other than text, such as images/video/sound etc., although some search engines are experimenting with image search methods. In the following, we briefly review some of these search engines

AltaVista [1]: Using the Virage image search engine, this indexes image available on the World Wide Web by text based indexing. Queries are entered by the user by entering keywords related to the image they wish to find. Images stored in a database by altavista that match the keywords are returned. This search method suffers from language limitations i.e. different meanings of words in different contexts giving wrong results to some queries, is limited to the language the description is written in and is reliant on the web page author giving relevant descriptions to their images.

Corbis Images [2]: This website contains a large number of photographs that have been manually annotated; users then search using keywords and images whose description match these keywords in some part are returned to the user. The database of images is not taken from the World Wide Web; instead it is the gathering of several separate image collections.

WebSEEK [3]: Composed of two main program, an autonomous software agent that searches the World Wide Web for images/video and collects meta-data (such as surrounding textual content and visual features) about the content, but not the *image data* itself. A separate system is then used to search this database in response to user queries regarding images. This search can take the form of text based searching or searching using visual features.

ImageRover [4]: This search engine combines textual and visual information to index its image database. A user searches the database by initially entering some keywords relevant to their query. This returns images whose textual information matches the keywords entered in some way. The search is then refined by the user selecting those images in the list of initial matching images that are most relevant to their query. From these images colour information, orientation, and semantic is taken and applied back to the database to perform another query. This process can be repeated by the user for as long as needed to perform a specific search for the necessary type of image. The interface to the search engine is a plain HTML based page, with a server being forked off to deal with each query. The database is gathered by running several autonomous robots, which parse pages and return textual and visual information about images on the page, as well as a thumbnail of the image. Links are also parsed and followed to collect more images.

Currently available search engines all take the approach of creating a database to store information about the images found on the World Wide Web. This database is then looked at when a query is received so that a reply to the query can be returned within a few seconds. The interface to all of the search engines is very similar to many text based search engines, even though new options could be added to an image search engine that could improve the search results.

In this paper, we present the design of WISE, an image search/retrieval program running on the WWW. The prototype design and implementation is similar to other search engines and web-crawlers, what is unique in our system is its image comparison algorithm. It is well known that images on the Web are either in GIF or JPEG format. The image comparison algorithm [6] developed by the first author and his colleagues can be directly used to compare GIF images. One of the purposes of the current work is to provide a platform to test the algorithm developed in [6] on the WWW.

The organisation of the paper is as follows. Section 2 describes the overview of the WISE system. Section 3 discusses issues concerning the design and implementation of autonomous Web agent. Section 4 discusses the design and implementation of the user interface. Section 5 briefly reviews a content-based image matching method developed by one of the authors and is implemented in WISE. Section 6 describes the overall system implementation. Section 7 presents testing results. Section 8 discusses possible future improvement of the system.

2. WISE SYSTEM OVERVIEW

The schematic diagram of our Web Image Search Engine (WISE) is shown in Figure 1. It consists of a graphical user interface (GUI) which will receive user commands and output results. Although not explicitly shown, the system also contains an autonomous Web agent, which is a set of programs that retrieve Web pages and extract information by parsing the retrieved pages. The user specifies the URL of a target image; the program will retrieve the target image and compute the visual features to be used for content-based comparison. The user also specifies the URL of the starting Web page from where the program will “crawl” the Web, finding images along the way and retrieve those satisfy the CONSTRAINTs inputted by the user. The CONSTRAINTs are a set of keywords and other information such as image dimensions, contained in the HTML text. The program then computes the visual features from the retrieved image and performed content-based comparison with the target image. The results of the comparison are then sorted according to a certain similarity measure.

At this initial stage, it is decided that a database will be created once a query is started. Once the query is completed the image data itself will be removed, and the search result will only contain the thumbnails and the URLs of those retrieved images along with other related content information, e.g., semantic descriptions available from the Web pages contained the images. The user can then study the result and retrieve those contain certain descriptions of interests.

Figure 1 A high level view of the WISE system

3 AUTONOMOUS WEB AGENTS

More commonly referred to as "spiders", "crawlers", "robots" or even "bots", autonomous Web agents are programs that retrieve web pages and extract information by parsing the retrieved page, including such information as links to other Web pages. The robots can then gradually build up a tree of pages to retrieve; adding more links as each page is parsed. Other information can also be extracted such as links to images in the pages, and keywords within the web pages. Some of the issues and problems related to writing a Web crawler are looked at below.

3.1 Page retrieval

The web pages can be retrieved by the program using the standard Hypertext Transfer Protocol (HTTP) used by web browsers to retrieve and display pages. Images can also be retrieved using this protocol. HTTP also allow for information about the file size and file creation/modification time to be sent before the file data is

transferred, meaning that we could limit the images to be retrieved to dependent on their size, without having to download the file to find the image size out. We can also use the modification time if we choose to keep a local database of images, we can check that we have the most up to date copy of the file available, if not we will retrieve the new copy. Once we have retrieved copies of web pages we then need some method of looking through them to extract the relevant data, which we will do by parsing the retrieved page.

3.2 Page Parsing

The program will obviously have no understanding about the page it has retrieved. We need some way to tell the program what parts of the page we are interested in, such as the links to other pages, links to images, etc. As we will mainly be concerned with pages written using the HyperText Markup Language (HTML) we can tell the program to look in the page for links written in the fixed format given by the HTML specification. Using regular expressions we can then tell the program to recognise these patterns within the page and remember them to be used later. For example, a web page may contain the following HTML code

```
..... <h1><a href="http://www.oranges.com/images/index.html">Link to Images</a></h1> .....
```

We would tell our program to look through this HTML code for any text that starts with '<a href="' and save the URL that follows that text. So in the example above our program would save the URL "http://www.oranges.com/images/index.html" for use later on.

However, there are variations on this format that are allowed within web pages, which we must also be able to deal with. Relative URLs are often used, as in the following example,

In the page at <http://www.oranges.com/drink/index.html> we may find the following HTML code

```
..... <h1><a href=" ../food/index.html">Link to Food Index</a></h1> .....
```

Our parsing program would see this code, recognize the text '<a href="' and remember the URL "../food/index.html" . However this is not a valid URL by itself, and as such could not be used to request the web page. Instead it must be taken relative to the URL of the page it was on (in this case <http://www.oranges.com/drink/index.html>), with the "." indicating to move up one directory. So the correct combined URL would be <http://www.oranges.com/food/index.html> . The program must be able to deal with generating the correct URL in cases like this.

So we now can find and retrieve web pages and images , but we need some way to deal with storage of this data once we retrieve it. The next section looks at the best ways to cope with this.

3.3 Storage

We need to choose whether or not to store the images and pages we retrieve in a local database. This would mean that once the data has been retrieved the first time, if it is needed later on we can use the local copy of the data rather than having to retrieve the data over a network again, reducing the time taken to receive the data.

However, this storage does have problems. Web pages and images do not always remain static, they are modified over time by their author. As these changes are not propagated over to our local copy of the pages or images, we would need to check before we used a local copy that it is the most up-to-date version of that data. As mentioned before, HTTP allows for checks of this sort to be performed, so we could use our local copy of the data if it was the most up-to-date version, and update our local copy with the latest version if we do not have the most up date version.

So keeping local copies of data would possibly speed up a query if the data from the same location were looked at more than once. However, once a query has finished, we have to decide whether we will keep this local copy of the data or whether to delete it. If we do keep all of the data we may be able to improve other query speeds if they happen to require data that we already have a local copy of. However, it will need a large amount of storage space to hold all of the data. If we only kept a local copy of the data for a particular query, we would need only limited storage space to hold the data for that query.

The autonomous robot has to perform the three tasks described above, retrieval, parsing and storage. This will allow for searching for any images on any site, without having to keep a local database of these pages or images,

even though it will mean there is a delay before a response can be returned to the user. Because of the large number of pages available on the Web, we also need to have some limiting factor for the agent, so that it will return the results in a reasonable time. One method we can use to do this is to limit the number of links to follow from the initial URL, called a depth. This depth parameter will be set by the user, so they can control how long the query will take.

4. INTERFACE

Because of the long delay between the search being started and the final results being output, it was decided that as the search progressed information should be given to the user to provide them with the current results of the search, the best match so far, number of web sites looked at, number of images looked at, number on images still to-do, etc. To implement this interface, the following methods are available.

Java Applet. One possible implementation is to launch a java applet on the client machine where the user can enter their search parameters. This would allow for more advanced searches to be entered, for example in the WebSEEK project users could select colours they wanted to search for by moving sliders for each colour component to the wanted value. Another approach may be to have a colour picker, where the user can select a specific colour they wish to find in images. The advantage of using a Java application for this is that it gives immediate feedback to the user, as it can communicate with the server as the search continued and output the search progress to the user. But, it would restrict users to those with Java aware browsers, and would place a load on their machines which they may not want have to deal with and would prefer the search to be performed on the server side.

Javascript. Javascript in the page could be used to provide some limited feedback to the user upon making choices for the initial search, for example, Javascript could be used to display the target image the user has entered before the search is started, allowing the user to ensure they are searching using the correct target. However, Javascript cannot communicate with the server, so it would not be able to show search results as they progressed.

Static HTML. Using only HTML on the page will only allow for simple data entry, using the Form interface specified in the HTML standard. However, search results could be updated as the search progressed, as the web page can be updated on the server with the latest results of the search. The user can then look at that page regularly to see the current results and when the search is finished that page can also be written to show the final results.

Although writing a Java application or Javascript would allow for more information to be given to the user, they both have drawbacks. Javascript has no way to update pages as the search progress, and Java applications would limit the search engine to users with Java enhanced browsers. So instead, using only normal HTML pages will be the approach taken, with the pages updated on the server as the search progresses.

5 CONTENT-BASED IMAGE COMPARISON

There are many content based image comparison methods developed in the past decades [5]. In the first stage of the project, we implemented a method developed by one of the authors' [6]. This method is especially suitable for comparing colour quantised colour images. There are two universally accepted image formats on the WWW, gif and jpeg. Because gif image itself contains a colour table in its header, it is particularly suited for fast comparison of this type of images. We plan to implement other more complicated image content matching methods e.g. some of those mentioned in [5] in the future, at the time of writing, the method of [6] is the only content based matching method being integrated into the WISE system. The other image format jpg is first converted to gif via an ImageMagick routine and then perform content-based comparison using the colour table in the gif format.

Colour quantisation techniques find statistically most significant colours of an image as its palette. Matching palettized colour images by their colours can thus be realised by matching their palettes. Let $P_A = \{(R_i, G_i, B_i), i = 1, 2, \dots, N\}$ and $P_B = \{(R_j, G_j, B_j), j = 1, 2, \dots, M\}$ be the colour palettes of images A and B, consisting of N and M colours respectively. To measure the similarity of these two colour palettes, the following modified Hausdorff distance (MHD) measure was used in [6]

$$MHD = \max(mhd(A, B), mhd(B, A)) \quad (1)$$

where $mhd(A, B)$ defined as

$$mhd(A, B) = \frac{1}{N} \sum_i^N \min_{\forall j} \|P_A(i) - P_B(j)\| \quad (2)$$

is the modified Hausdorff distance from A to B, and $mhd(B, A)$ is defined similarity, and $\|\cdot\|$ denotes the Euclidean distance.

As can be seen, instead of taking the maximum of the minimum, as in the original Hausdorff distance, the average of the minimum is used as this proves to be less sensitive to noise.

It should be noted that the actual image dimensions do not affect this matching technique as only the information of colourmap itself is used. Hence, its efficiency does not decrease with increasing image size.

6. SYSTEM IMPLEMENTATION

In order to implement the system, the following software has been written,

1. A program implements the image-matching algorithms and outputs the similarity between the target image and another image.
2. A program retrieves images from websites, which also parses web pages to retrieve image locations and locations of related web pages.
3. A CGI script processes the user query and submits the query to the image searching program.
4. Web pages to provide an interface between the user and the image search program.

The CGI script could be written in many different languages, however because of pre-existing libraries available for writing CGI scripts; Perl has been used to allow for code-reuse. Because of the speed at which the image comparison program needs to work, it has been written using C. This allows for the program to be optimized specifically for use in image matching. The image search program needs to be able to retrieve and parse web pages, for which there are again already existing libraries available for Perl. So this program has also been written in Perl to take advantage of this code. The screen shot of a partially finished system's user interface is shown in Figure 2.

6.1 Optional Constraints

Apart from content-based comparison, some of the possible constraints that may be implemented to make the search more effective are discussed as follows.

Image File Size: We can limit our database to only images that are greater or less than a certain size, which could be used to exclude image below a certain size on the basis that they are too small to contain enough information to match the search image properly. Images over a certain size could also be excluded to reject those images too large for the user to consider.

Image Dimensions: We can limit the database based on how large or small in geometrical terms we wish the image to be. This could be used to exclude images that are too small to be of interest, for example images are often used in web pages to control the layout of the page, usually consisting a size of 1 pixel in one dimension and a unknown length in other dimension, so we could exclude all images that have a size of 1 pixel in one dimension as these images are unlikely to be of interest to the user. We could also use this feature to do exact matches, that is to only find images that have exactly the same dimensions as the target image, which may provide for a more accurate search results.

Site limited URL: We can limit the URLs that we follow from the first page to only those that are on the same host as the first page, for example: If we are given a start URL of <http://www.oranges.com> we would only follow URLs from that start page that were stored on <http://www.oranges.com>, so we would retrieve the page <http://www.oranges.com/growing/maps.html> as this is on the same host, but we would not retrieve the URL <http://www.apples.com> as this is on a different host. This feature can be used to limit search to particular sites of interest, without leaving the initial site to look at images on other sites that may not be of interest to the user.

Figure 2, A screen shot of the WISE Interface

Directory limited URL: A slightly more specialized case of the above, where not only do we limit the URL's to those from the same site, we also limit them to those in or below the same directory as the initial URL, for example: Given a start URL of <http://www.oranges.com/drinks/> we would only follow URLs that start with this URL, so we would look at <http://www.oranges.com/drinks/news/latest.html> as this is in a directory that is a child of the original URL, but we would not look at <http://www.oranges.com/pie.html> as even though it is on the same host, it is in a different directory. This can be used to limit searches to a particular subsection on a website that may contain various information, of which the user is only interested in one part. This way we can exclude all the other sections on the same site.

Image types: Two of the main image types used on websites are GIF and JPEG images. With the GIF image format we can use our algorithm to search through the colourtables stored at the header of the image format, thus being quick to process. JPEG images are more complex, with no colour contents available without reading in the entire image.

Regular Expression in Page: We can restrict the images in our database by only including images that are on web pages that contain a certain regular expression as specified by the user. This allows for even more specific searches, which will concentrate on a topic selected by the user. In this way we can limit images in a way that we cannot do using image recognition at the moment, as it is not capable of understanding high level queries.

We may wish to search for images of apples. The user may enter the regular expression "apple" as well as a target image and a target URL. We would then check every web page that we would normally look at to see if it contained this word, if it did we would then retrieve the images for this page, if not we would not look at the images.

Regular Expression not in Page: Simply the inverse of the above, to provide extra ways to exclude certain pages that are not of interest, e.g. Exclude "pineapple".

Page/Image Creation Date: Using this we can try to include/exclude certain groups of images by only using those that are created after/before a certain date. This can be used to reduce search results for when we have information about an image, such as the date it was taken. This can also be used to look for newest images, for example on a news website it can be used to only look at images created in the last 2/3 hours, so only the very latest images are included in the search.

6.2 Optimisation

We can improve the speed of the search engine by having multiple processes running for each query. For example we can have one process that will retrieve images and compare them against the target image, at the same time another process can be running that will retrieve Web pages and parse them to find URLs for other Web pages and images. We would also need a process that managed both of this two process, letting them communicate their results to each other, and checking if there is any problem. However, this process can probably be merged with the Web Page gathering process.

We could continue to have new processes for each query, for example have 2 processes retrieving images and 2 retrieving Web sites, as in theory this would improve the time taken to perform the query. However, as all of these processes would have to be in communication with each other, the complexity of the code would increase exponentially with each additional process, as it would involve more management of the sub-process. So for the moment it seems best to only have 2 process per query, as this only involves 2-way communication.

6.3 Program Operation

To use the program to search for images on the WWW, the following sequence of event will take place:

1. The user uses their Web browser to point to the image query page¹.
2. The user fills in the required forms with the url of the query image, the url to start searching for images on, and the number of pages to move away from the start page (depth). Additional options may be present on the web page as well.
3. The user submits this data using the standard FORM mechanism in HTML.
4. The data is received by a CGI script on the server, the data is checked to ensure it contains valid parameters, if not an error page is returned to the user. If all the data is correct, the script sends the query to the Image Search program via a Unix internal sockets which the image search program is listening on.
5. The image search program receives new queries from the CGI script, it forks (i.e. launches a identical copy of itself as a separate program) and starts the search for images and pages based on the data passed to it, it also returns a string to the CGI script that sent the original request to confirm whether the request has been started or not and if it has been it returns the location of the page on the server where the results of the request will be outputted. The original image search program goes back to listening for more quires from CGI scripts.
6. The CGI script from part 4 receives the information from the image search program regarding the success or failure of the request. If the request has failed the CGI script generates a error message informing the user why and returns this page back to the user. If the request has succeeded the CGI script uses HTTP headers to send a re-direct to the user, that is it tells the browser to load a new page which will contain information about the users search request.

Once the query has been received by the image search program and it has forked, the following process is undertaken

1. The forked image search program initially retrieves the query image and query URL from the relevant remote Web sites. If it fails to retrieve either of these the search is aborted and an error page is generated informing the user of why the search failed.
2. If both target image and target page has successfully been retrieved the forked image search program then forks again, opening a pipe between itself and its child to communicate over. From now on, all images will

¹ We have an experimental website www.scs.leeds.ac.uk/wise, which is still under construction. Readers can run the query program but is quite slow at this moment. Some of the results we have obtained can be found in the query result page www.scs.leeds.ac.uk/wise/query

be retrieved by one process and all pages by the other. Management of the search will be controlled in the same process as the page retrieval process, which will also store image search results as the search progresses.

3. The current page is parsed to find the URLs contained within it and the image links. If none are found and we are on the target page the search finishes as we have exhausted all the page links, so we go to step 7. Otherwise it means we have finished with this particular branch of the URL parse tree, and we move back up one page and pick the next URL that we have not already visited from the list of URLs on that page
4. The list of images to retrieve for the current page is then passed over to the child process
5. The child process starts to retrieve the images given. If any image takes longer to download then a set time-out limit the download for that image is aborted and that image is ignored. Once all the images have had at least 1 attempt at download, they are compared against the query image. The scores for each image and the success or failure to download each image are then written back to the parent process. The child then waits for the next list of images to download.
6. The parent retrieves the page given in one of the URLs from the list of URLs found in the current page. It then parses this page and gets a new list of images to retrieve and a new list of URLs to follow. The parent then waits for the child process to return the list of results from the previous list of images, if the child does not reply within the time-out limit the parent assumes the child has been stopped working for some reason so it forks itself again, although it does not need to retrieve the initial target page and target image again as it has already saved them. If the child does reply within the time-out limit the results from each image comparison with the query image are compared with the other results already received. If a better match is found a new web page is written out showing the new best match found so far is. This page can be viewed by the user who started the search to monitor the progress.. The search then goes back to step 3.
7. The search has finished, we now look at the results for each image and return a page to the user that shows in ranked order the results of the search, with the score given and a thumbnail picture of the image that achieved this score, along with a link to it and the page that it was on.

7. TESTING

We have successfully tested the program. Figures 3, 4, and 5 show the screen shots of the search results and the top 30 returned images. These results are available on www.scs.leeds.ac.uk/wise/query and readers can also run the live program on www.scs.leeds.ac.uk/wise. (slow at the moment).

8 FUTURE IMPROVEMENTS AND CONCLUDING REMARKS

We have presented the design, implementation and preliminary results of a program, which allows the user to search for image on the WWW, based on image content. The software is still under development, we are trying to improve its features constantly.

There are many extra features that can be added to the software to allow for greater control over the search by the user. Many of the options presented in Section 5 could be implemented to the existing program to give a more powerful search technique.

9 REFERENCES

1. <http://images.altavista.com/>.
2. <http://www.corbis.com/>
3. <http://www.ctr.columbia.edu/webseek/>.
4. <http://www.cs.bu.edu/groups/ivc/ImageRover/demo.html>.
5. Y. Rui, T.S. Huang, T.S. and S. F. Chang, "Image Retrieval: Current Techniques, Promising Directions, and Open Issues", Journal of Visual Communication and Image Representation, Vol. 10, pp. 39-62, 1999
6. G. Scheafer, G Qiu and G. D. Finlayson, "Retrieval of palettized color images", Proc. of SPIE, vol. 3972, Storage and Retrieval for Media Databases 2000, San Jose, CA, USA, January 2000, pp. 483 – 493

Figure 3, Screen shot of the resultant page of a particular searching session and all top 30 returns

Figure 4, Screen shot of the resultant page of a particular searching session and all top 30 returns