IJDAR manuscript No.
(will be inserted by the editor)

Harvesting maps on the Web

Aman Goel · Matthew Michelson · Craig A. Knoblock

Received: date / Accepted: date

Abstract Maps are one of the most valuable documents for gathering geospatial information about a region. Yet, finding a collection of diverse, high quality maps is a significant challenge because there is a dearth of content specific metadata available to identify them from among other images on the Web. For this reason, it is desirous to analyze the content of each image. The problem is further complicated by the variations between different types of maps, such as street maps and contour maps, and also by the fact that many high quality maps are embedded within other documents such as PDF reports. In this paper, we present an automatic method to find high quality maps for a given geographical region. Not only does our method find documents that are maps, but also those that are embedded within other documents. We have developed a Content Based Image Retrieval (CBIR) approach that uses a new set of features for classification in order to capture the defining characteristics of a map. This approach is able to identify all types of maps irrespective of their subject, scale and color in a highly scalable and accurate way. Our classifier achieves an F1-measure of 74%, which is an 18% improvement over the previous work in the area.

A. Goel · C. A. Knoblock
Department of Computer Science and Information Sciences Institute
University of Southern California, Marina del Rey, CA 90292 USA
E-mail: amangoel@isi.edu
E-mail: knoblock@isi.edu

M. Michelson
Fetch Technologies, 841 Apollo St. Suite 400, El Segundo, CA 90245 USA
E-mail: mmichelson@fetch.com

Keywords Map identification · K-Nearest Neighbor Classifier · Water-filling features · Content Based Image Retrieval (CBIR) · PDF documents

1 Introduction

Maps are one of the most important documents containing geospatial data about any region in the world. They provide information about a wide variety of subjects including road networks, transportation routes, natural terrain, buildings and other infrastructure, weather, water and gas pipelines, etc. The Web is a great source of information of all kinds, including maps devoted to the aforementioned subjects. The capability to find and process these maps automatically aids in the construction of useful geospatial knowledge bases. This in turn permits queries similar in nature to those applied to satellite image data, for example queries related to region, latitude/longitude, street name, etc. Researchers have dealt with various problems in automatically processing map documents, such as extraction of road layers [5], conflation of maps with satellite imagery [1] [2] (see Figure 1), etc., but most of them assume that they have a corpus of map images to process. In this work, we tackle the problem of building that corpus by harvesting maps from the Web. In addition to the maps that exist independently as images, we also parse and extract suitable examples from documents devoted to geospatial subject matter, such as PDF reports, etc.

There are a couple of challenges involved in this task. First, we cannot rely on only one repository of maps since those maps would describe only a particular subset of properties of an area, such as its infrastructure or topological characteristics. Despite the wealth of map data available via the Web, said data is highly

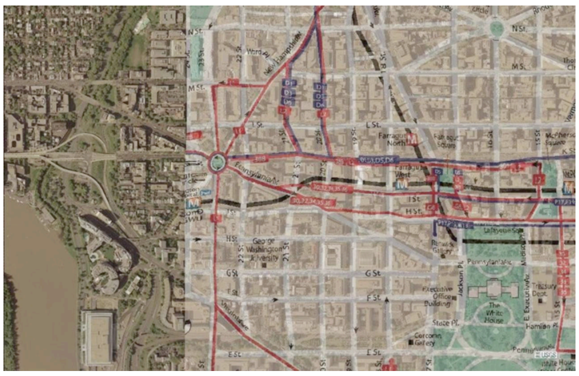


Fig. 1: A Washington D.C. map conflated with the satellite imagery of the region

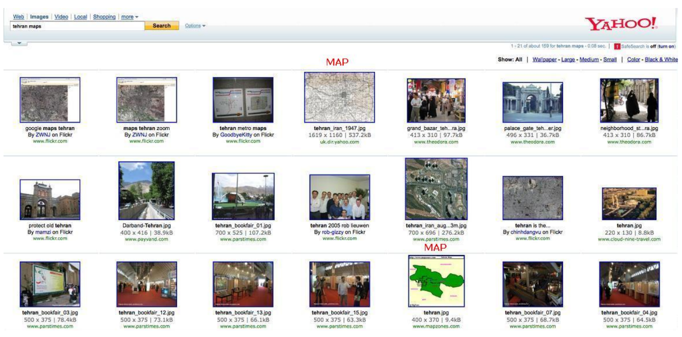


Fig. 2: A snapshot of the first search result page for "tehran maps" on Yahoo Image Search

distributed, in various formats and frequently embedded in other objects such as PDF documents. Identifying these maps among the multitude of images is hard, since for most of them there is no metadata to indicate that they indeed are maps. This would suggest that we need to look at the actual content of each image to determine whether it is a map or not.

Web based image search engines, such as Yahoo Image Search[1] are inadequate when tasked with finding specific maps. Figure 2 shows a snapshot of the first result page returned by this service for the query – "tehran maps". Only two of the 21 displayed images are maps, namely, the 4th on the first row and the 5th on the last row. The rest of the images are either satellite images of Tehran or pictures of different places or events. The reason for this is that a search engine indexes an image based on the name of the file and the text surrounding it in the webpage where it appears. It does not look at the actual content of the image. Therefore, if someone wishes to collect a sufficient number of maps for Tehran to cover most of its area using this search engine, he or she will have to browse through all the results pages, hundreds of them, to select the few good maps which appear on each page. If one wants to find documents that contain maps using a search engine, this manual process will be even costlier since it would involve downloading each such file and looking through the whole document. On the other hand our system can make this search not only convenient, but also fast and more accurate.

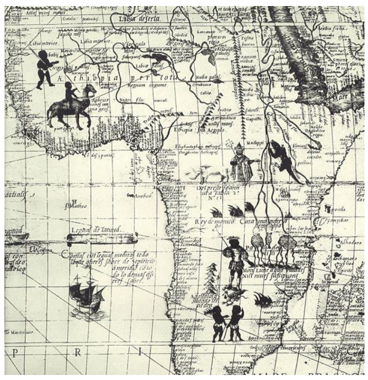
Since map is a very generic word, we would like to define precisely what we mean by it in this paper when we talk about identifying or harvesting maps. A map is a high quality image which depicts a region of earth with sufficient clarity so as to be useful for extracting information from it. Therefore a very poor quality scan of a paper map (Figure 3a) or a hand drawn map (Figure 3b) or a photograph of a paper map in bad light (Figure 3c) or a picture with a tiny part of it displaying a map (Figure 3d) is not considered a map. A computer vision system might be interested in identifying any kind of map or even a photograph of a map, but our system is meant for identifying those maps which can be usefully processed to extract road intersections and other features of the region for alignment with satellite imagery. Figure 4 shows some of the images which we want to identify as maps. As is evident from the collection, we consider software generated maps (Figure 4a and 4b) and scans of paper maps (Figure 4c) as good maps which can be processed successfully.

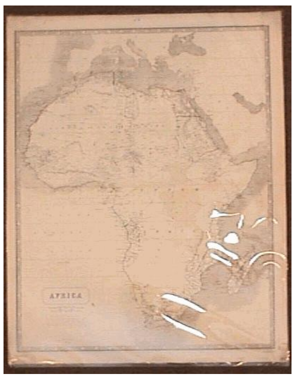
In this paper, we describe a method to automatically collect high quality maps for any region, from the Web. Figure 5 shows the architecture of our end-to-end system. The rectangles in the diagram represent collections of data (e.g. documents, images). The quadrilaterals with slanting sides depict a process. Since search engines have the most exhaustive index of the Internet, we use them as the starting point. We query the search engines for documents and images related to the area. All the images embedded in the gathered documents are then extracted and put together with the independent images. This combined set contains maps of the area as well as other nonmap images like pictures of people, places etc.

Each image is classified by applying a K-Nearest Neighbor classifier [10] based on Content Based Image Retrieval (CBIR), which we call CBIR classifier in this paper. Our CBIR approach finds the most similar images to each query image from a repository of pre-labeled map and nonmap images based on a set of similarity features. These features are derived from the Water-filling algorithm described by Zhou et al. [12]. The use of this particular feature set in this work is due to their efficacy in capturing characteristics germane to

[1] http://images.search.yahoo.com/

(a)


(b)


(c)

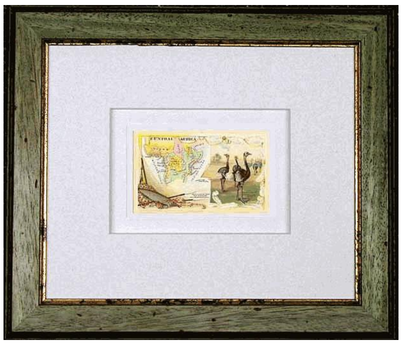
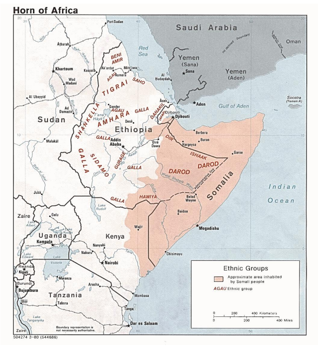

(d)

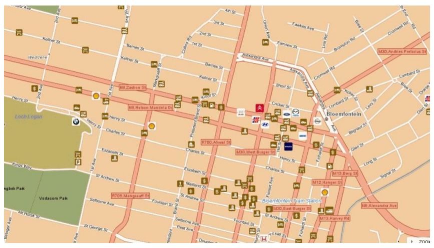
Fig. 3: Images not considered as maps. Poor quality scans (3a), Hand drawn (3b), Bad lighting (3c), Very small portion of the whole image (3d).

maps, for example, sharp boundaries, high branching of lines, etc. The image is then classified as belonging to the same category to which the majority of the most similar images belong.

We tested our system on real images from the Internet which represent the actual distribution of various map types for different regions. Our experiments show that our classifier performs better than the classifier used in the previous work by Desai et al. [3] by almost 20% in F1-measure. We attribute this result largely to the use of the previously mentioned feature set, versus the use of Laws' textures [9]. Also, our experiments show that K-Nearest neighbor classifier is much better at identifying maps than a Support Vector Machine (SVM) [13] when the training data set is small in size.

We described our map classification algorithm in a previous paper [4]. Further work documented in this article has been dedicated to modifying the Water-filling algorithm in order to capture the properties of maps more accurately. Also, we have ignored some of the properties suggested by Zhou et al. and adjusted the


(a)


(b)

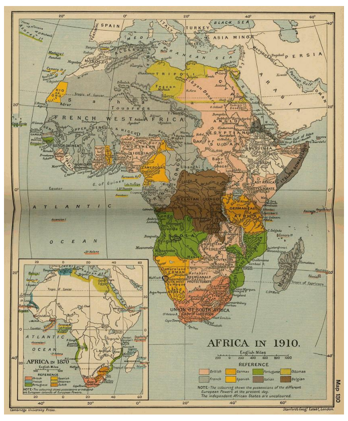
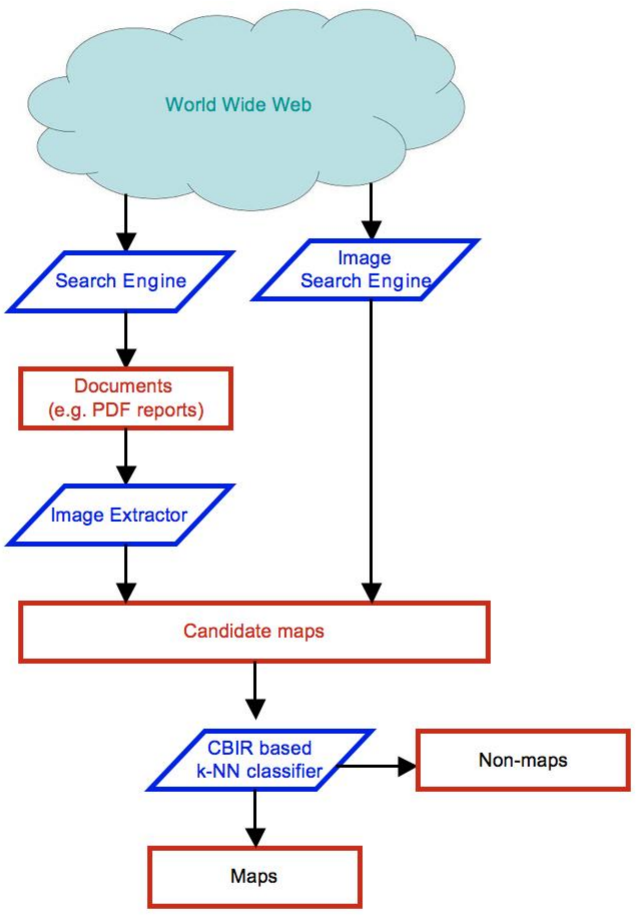

(c)

Fig. 4: Images to be identified correctly as maps

weights of others to make the algorithm more specific to the task of identifying maps. We describe all these changes in Section 4.1 and 4.2. We have also conducted an elaborate set of experiments on a much larger and diverse data set, with a focus on classifying maps which are totally different from the labeled data in terms of scale, locations etc. (e.g. classifying non-US maps based on a repository of maps of cities in United States).

The major contributions of our paper are the following.

1) We have developed a set of features based on the Water-filling algorithm which can capture the characteristics of maps more accurately as well as more generically, independent of color, scale, origin, size etc.



(a)

Fig. 5: The architecture of our system

2) Our paper shows that CBIR based k-NN is more suited to classification of maps as compared to Support Vector Machines when the training data set is small in size. When the data set size increases, our classifier scales very well because it doesn't require retraining on the labeled data, unlike SVMs. Therefore the incorporation of new map data may take place with relative ease.

3) We present an approach to map classification based on the previously mentioned feature and classifier which not only outperforms the existing state-of-the-art by almost 20% in F1-measure but also requires less time and space for processing by a couple of orders of magnitude.

4) Our paper describes an approach to finding documents containing maps related to a particular geographical location, focusing specifically on PDF files.

5) Finally, our paper describes an end-to-end system, MapFinder, for harvesting maps from the Web which is accurate, exhaustive and automatic. Given only the name of a location, this system searches for images on the Internet, downloads and extracts candidate maps, classifies them using the classifier mentioned above, and then presents a collection of maps ready to be used for geo-referencing (among other applications). Our system is very scalable in terms of increasing the repository size, as well as including new map types for identification.

The structure of this paper is as follows; we discuss related work in Section 2. We describe the technique by which we collect potential maps of a region in Section 3. We then lay out our technique for classifying these images into maps or nonmaps in Section 4. We explain the methodology of our experiments and present the results in Section 5, which also compares the performance of our system with the current state-of-the-art. We discuss future work in Section 6, and conclude with a discussion in Section 7.

2 Related Work

Previous work on harvesting maps from the Internet was done by Desai et al. [3]. They used Laws' Textures [9] as the representative feature set to differentiate between maps and nonmaps. These features capture the existence of various regular shapes, such as circles and rectangles in images. We believe that such features are not specific to maps in general, and therefore are not very effective in differentiating them from nonmaps. For classification, the authors trained a two-class SVM on the 3,840 element feature vector based on this texture. We, on the other hand, use Water-filling features with CBIR based k-NN classifier. As demonstrated in the experiments section, both, our feature set and classifier performs better than their respective choices for this application, and combining them together, we get a performance improvement of almost 20% in F1-measure in identifying maps. Also, our approach is more efficient in terms of both time and memory requirement, since we use only a 24 element feature vector and our method takes about 3 seconds to process an average size image (1000x1000 pixels), compared to 30 seconds, that is required to extract Laws' Texture. Further, we also present a method to collect maps embedded in other documents on the Internet. Our experiments demonstrate that general documents like PDF files are indeed useful sources for harvesting high-quality maps.

CBIR methods have been applied to various scientific disciplines ranging from astronomy [17] to botany [18]. Much attention has been given to CBIR methods in medicine, where they can have tremendous impact [19] [20]. For example, in one medical system [20], the authors use a CBIR-based k-Nearest Neighbor approach to classify medical images. This work also uses Water-filling features for the CBIR component. However, this combination performs the worst among the five different classification systems they have tested because Water-filling algorithm does not work well with images having amorphous boundaries and gradual color gradients, which is typical of medical images such as X-ray images (Figure 6). On the other hand, maps generally have sharp boundaries and no color gradient, properties that we have exploited to our benefit in this application. Also, the context in which they apply these algorithms is very different from ours. Whereas our system is geared toward automatically harvesting maps from the Web, their system is used to classify images so that they can be queried categorically.

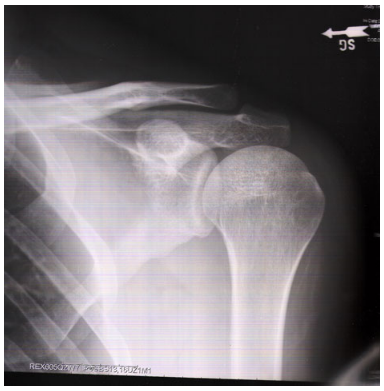


Fig. 6: Medical images such as X-Ray photographs have fuzzy boundaries which makes Water-filling features inappropriate for their classification.

Tan et al. [24] recently proposed a method for identifying maps embedded in Web documents by analyzing captions of images, references to them that are embedded in text, and their size relative to the font. They do not look at the actual content of the image. It might be possible to combine the two approaches together to take advantage of evidence about the content as well as the text surrounding it to achieve more accurate results. Yet, unlike them, we do not assume that a highly relevant library of documents is already provided to us, but discover the Web documents that are likely to contain useful maps ourselves. This also lets us gather more diverse types of maps.

3 Finding candidate maps

The first step in harvesting maps is collecting images from the web. This includes finding PDFs from which we will extract the images. We use the Yahoo Image Search Engine API[2] for the purpose of collecting the independent images. The choice of engine is mostly influenced by convenience and does not affect the performance of the process. Any reasonable image search engine is sufficient. To collect maps for a region, we send two queries to the search engine. One query is formed with the word "map" appended to the name of the region (e.g. "Los Angeles map"). The other query is formed by appending the word "maps" in place of "map" (e.g. "Los Angeles maps"). We know that the relevance of images to search queries falls sharply after only a few result pages. As we go further in the list of URLs returned, sorted by relevance, the existence of maps becomes more and more sparse. Therefore, we use the top 2,500 URLs returned for both the searches for our purposes. We remove the duplicates from both the lists and then merge them together.

The fact that the search engines fail to identify and provide maps correctly, becomes evident when we look at the images which are common between the two lists returned by the search engine. Although there is little semantic difference between the two search queries when it comes to images, there is less than 10% overlap between the two lists. Therefore, a person searching for maps on these search engines using only one of the queries will miss most of the maps which can be obtained by the other. The maps, among the images returned, are of varied types and scales. This is one of the advantages of using a search engine for harvesting the maps. Limiting the search to a particular database restricts access to all kinds of maps available freely on the Internet. Also, new content is constantly added to the Internet and indexed by the crawler of these engines. This guarantees a supply of up-to-date maps and greater detail about every region. Figure 7 shows some of the different kinds of maps returned by the search engine. There are physical maps (7a), street maps (7b), political maps (7c), highway/freeway maps (7d), commercial maps (7e), transportation maps (7f), etc.

Yet, all the good quality maps are not available as separate images on the Internet. Many high quality maps exist embedded in documents such as PDF reports. Therefore, in order to exploit these maps, we query the Google Search Engine[3] for URLs of PDF files related to the region. Again, the choice of the search engine is not fundamental to the performance of the system, as the major difference between the various engines is the relevance they attach to each document. Since, we do not restrict ourselves to only the top few URLs, the system effectively has access to the same

[2] http://images.search.yahoo.com/
[3] http://www.google.com

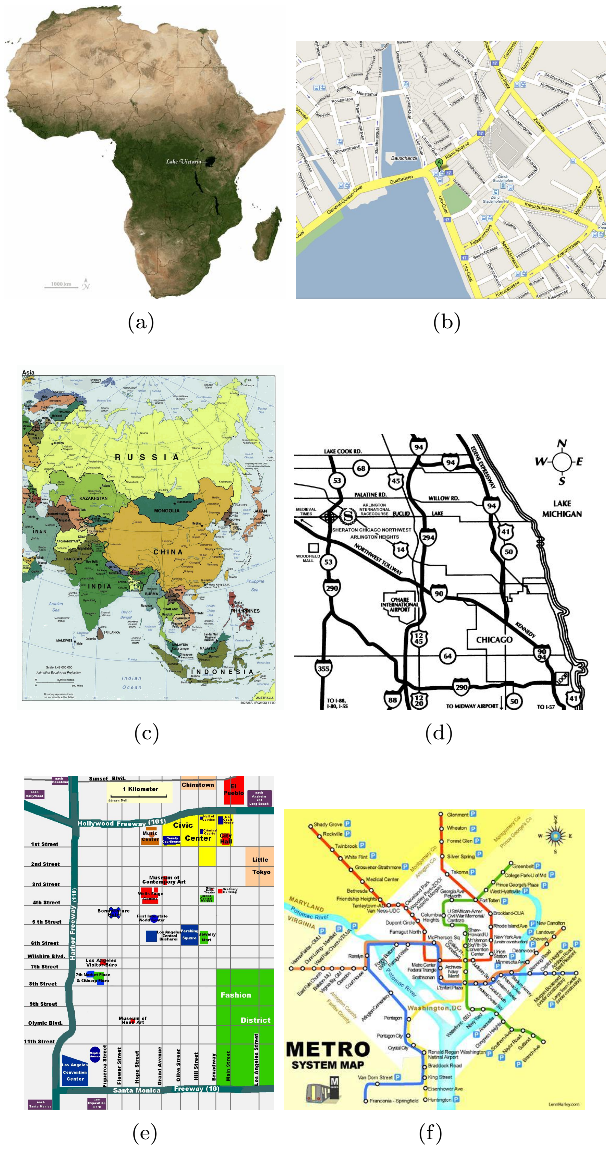


Fig. 7: Different kinds of maps available on the Internet. Physical maps (7a), Street maps (7b), Political maps (7c), Freeway maps (7d), Commercial maps (7e), Transportation maps (7f).

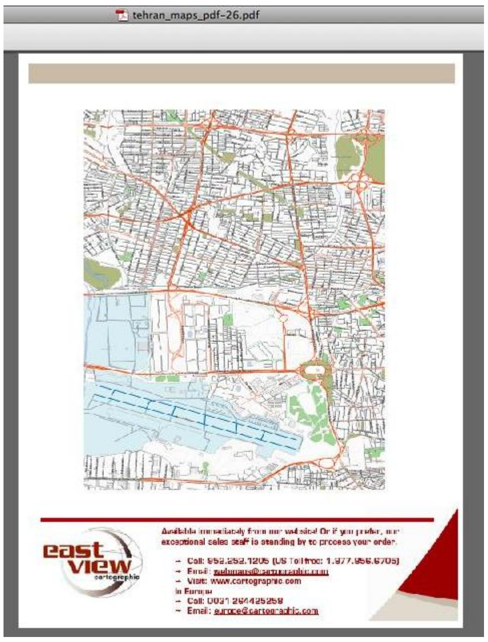
files that would be returned by any other search engine. We use the same query terms as explained earlier for the images (e.g. “Chicago map” and “Chicago maps”), but we also append the file-type qualifier to restrict the results to PDF files only, which we focus on for this study, based on the availability of geographic reports. Therefore an example query for Chicago would be “Chicago map filetype:pdf”. A PDF file generally has text included in it along with the images. Hence a search engine would return the PDFs with greater accuracy based on the word “map(s)” included in the query.

We have developed a PDF parser to extract the embedded images out of the downloaded files. According to the official PDF specification[4] by Adobe, a PDF file stores images, text and all other information in the form of objects, which reference each other. The primary object for images defines the width, height and type of the image and the compression algorithm used to store the image data. The actual image stream and optional color maps are stored in other objects which are referenced from the main object. Our parser reads through the entire file to rebuild the objects in memory. Then it replaces the static references with memory pointers. Depending on the kind of compression, type of image, color map, etc., the parser decodes the image stream and then stores it to the hard disk. The parser is capable of decoding most of the standard methods of storing images in PDFs including those based on ICC[5] profiles and JPEG formats.

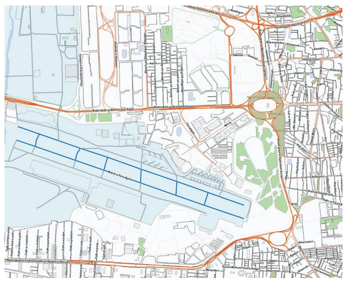
As shown in the experiments section, PDFs indeed contain a huge number of images embedded in them. Although most of these are too small (e.g. company logos, map legends), it turns out that we still get about one image per PDF file downloaded, which can be useful (i.e. big enough for processing). The percentage of maps among these useful images is higher (≈60%) than among the independent images (≈50%) downloaded using the method described earlier. Also, the quality of the maps is much higher. Figure 8a shows a sample PDF returned when Google was searched for maps of Tehran. We can clearly see a very high resolution and detailed map embedded in it. Figure 8b and 8c show two high quality images extracted from the PDFs that we downloaded. The image in Figure 8c is of size 3675 pixels by 4575 pixels, which is quite difficult to find as a standalone image on the web.

It should be noted that the process of collection of independent images has almost no overhead. The entire time taken is that required by the search engine to return the list of URLs and to download the images from the Internet. In order to speed it up further, we use parallel threads to download these images. It also prevents a very slow website from holding up the download of images from other websites. We employ the same trick to download PDFs from the URLs returned by Google. But in this case we also process these PDFs to extract the embedded images. Since our parser has been built in-house for experimental purposes, it is not very optimized. Despite this, on an average, we require less than a minute per PDF file for the parsing, extraction and storage. This is not a very large time considering

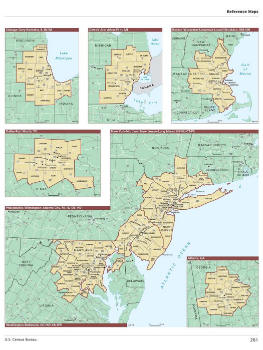
[4] http://www.adobe.com/devnet/pdf/pdf_reference.html
[5] http://www.color.org



(a)



(b)



(c)

Fig. 8: Sample PDF file (8a). Sample images (8b, 8c) extracted from some of the downloaded PDF documents.

the fact that there are fewer PDFs (≈50) than images (≈1000) corresponding to a query for the maps of a city.

4 Classifying images

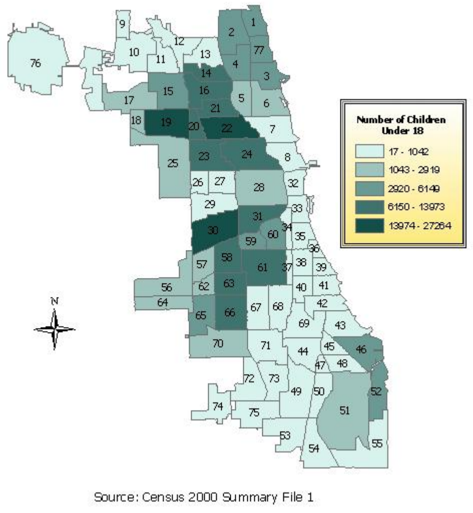
In this section we describe our approach used to separate the actual maps from all the images collected by the process described in the previous section. From each image, we extract a set of features to be made explicit shortly, based on the Water-filling features introduced by Zhou et al. [12]. The Water-filling algorithm works on the edge maps of the images instead of the original images. An edge map is a binary image produced from the original image by marking the sharp color gradients (edges) in black. Figure 9b shows an edge map for a colored image (9a). Using the edge maps makes the Water-filling features color invariant. Maps vary widely in their colors schemes, which is mostly controlled by the source of the maps. Previous work [6] dedicated to identifying various features in a map based on color have acknowledged the fact that variation in the colors in a map makes any algorithm utilizing them applicable only to a small percentage of maps, which are mostly from the same source. Therefore, we ignore the color scheme totally in favor of a color invariant strategy. We explain the Water-filling features along with the modifications we have made to them in detail in Section 4.1.

Having extracted the features, we find nine images from a repository of images that are most similar to the image under consideration. The images in the repository have already been manually labeled as maps or nonmaps and their features have been extracted and indexed with them. The number of maps and nonmaps among those nine images is calculated and the query image is classified as belonging to the class that is in majority. The process of finding the most similar images based on the content of the images (represented by some features) is called Content Based Image Retrieval (CBIR). The technique of classifying an image (or more generally, any object) based on the majority among the most similar images (objects) is called k-Nearest Neighbor classification (k-NN). We describe this process in further detail in Section 4.2.

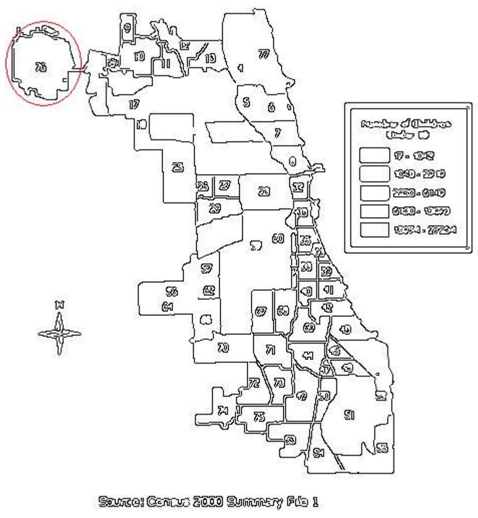
4.1 Extracting features that differentiate between maps and nonmaps

The Water-filling algorithm quantifies the general complexity of an image by the number of branches and lengths in the edge map. It does this by simulating the flow of water along connected canals which are represented by the connected edges in the edge map. We use the standard Canny edge detector [16] for generating the edge maps since it is more generally applicable to the wide variety of maps and is immune to the variations of hue and saturation in the original images. Figure 9c shows a part of the edge map (Figure 9b), which is circled, enlarged for clarity. The edge map consists of several disjoint segments of lines like the one shown in the center of Figure 9c. An average size map of dimensions 1000 x 1000 pixels can have any number of segments between 50 and 3000 depending on its scale and sparseness. For each of the disjoint segments, the water starts flowing from a point along the lines. When it reaches a fork-point, where the line is dividing into two or more lines, the water starts flowing along all of these lines simultaneously, in the same way as water flowing from one canal into multiple canals will flow along each of them. The algorithm notes the number of such fork-points (represented by “Fork Count”) as well as the time required to fill the entire segment (“Filling Time”). It also records the total amount of water stored in the segment (“Water Amount”), which is the same as the number of pixels in the segment.

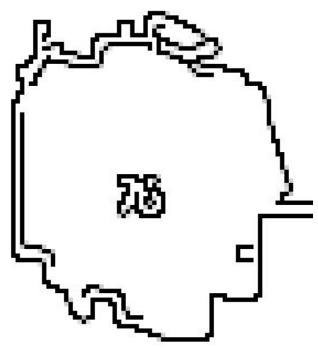
We explain the Water-filling algorithm in detail in Figure 10. Suppose that the top-left segment is one of the many segments in an image that is being processed. The three values below each instance of the segment show the calculated parameters up to that time in the course of the Water-filling algorithm tracing the segment. We show the Filling Time (“FT” in figure), Fork Count (“FC”) and Water Amount (“WA”), which all start with a zero value. The light pixels visible in other instances of the segment represent water, filling up the segment pixel by pixel. The progression of the algorithm is from left to right in each row, starting with the top row. The second instance of the segment (figure in the middle in the top row) shows the situation when the processing has just begun, with only one pixel (top-most) having been filled up. Accordingly, the Filling Time and Water Amount increase to one each. When water reaches a fork-point (in the figure on the right in the top row), indicated by the crossed pixel, where it begins to flow in two or more directions simultaneously, the Fork Count value increases by one. At the end, all the pixels are filled and the final values are shown below the last instance. The Fork Count is two because the water forked at two places, marked by the crossed pixels. The Water Amount is same as the number of pixels in the segment i.e. 16.


Number of Children Under 18
17 - 1042
1043 - 2919
2920 - 6149
6150 - 13973
13974 - 27264
N
Source: Census 2000 Summary File 1

(a)

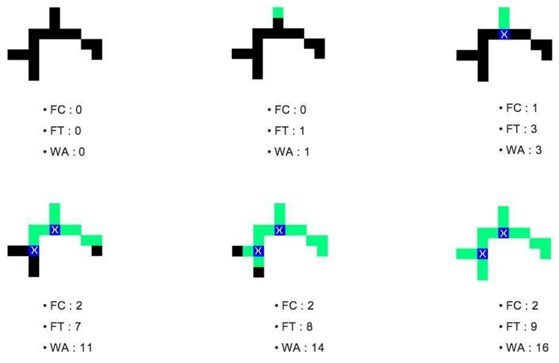


(b)



(c)

Fig. 9: (9a) A map in color, (9b) Its Canny edge map, (9c) A part of it enlarged to show the disjoint segments.



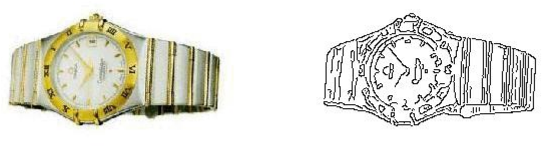
(a)

Fig. 10: Water-filling algorithm

The Filling Time of a segment indicates the approximate length of the longest path in the segment from one end to another. Intuitively, a map will have longer lines which represent streets, freeways and geographical or land/ocean boundaries, depending on the scale of the map. On the other hand, a typical nonmap will not have such long lines. For example, a picture of nature or vegetation will have many short segments for edges of leaves on trees and other growth. In an urban landspace, the edge map will have several short segments for windows of building, or faces of people in a crowd. Therefore, a map, in general, will have larger number of long segments as compared to a nonmap image.

The number of Fork Counts, on the other hand, represents the complexity of the segments. Again, a map will generally have more Fork Counts because of the branching of roads at intersection repeatedly or continuous borders branching in different directions to represent neighboring states, countries, etc. Also, by virtue of having longer connected lines in the edge map, the Fork Count per segment will be higher for the maps. In contrast, a nonmap with discrete windows or short leaves will have Fork Counts of mostly zero or one.

The Water Amount in each segment represents the the size of the segment. In a map, the large lengths of the lines and high connectivity, as indicated by the Fork Count, makes average size of segments large. In the case of nonmaps, besides the fact that they have lower Filling Time and Fork Count, the color gradient causes the various connected lines to break in the middle. This causes the average segment size to be much smaller. Figure 11 shows how the color gradient causes even clear object boundaries to break up into smaller segments. The edge of the dial of the watch breaks up at several places even though a human eye can clearly see the entire boundary in the original image.

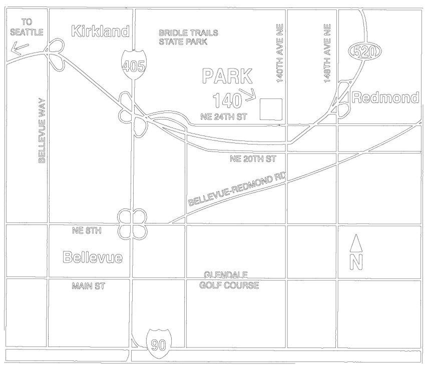


(a) (b)

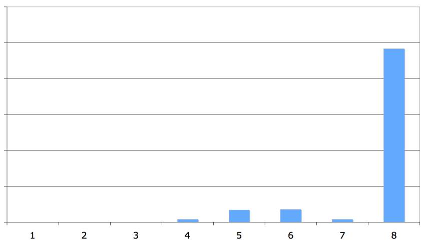
Fig. 11: Segments get broken because of the color gradient. Even though the edge of the dial in the original image (11a) is a full circle, it gets broken in the edge map (11b).

To capture the differences in distribution of Filling Time, we distribute all the segments into 8 bins depending on their individual Filling Times. The range for the 8 bins is as follows : segments with filing time of 1-2, segments with filling time 3-5, 6-9, 10-14, 15-19, 20-24, 25-29, and finally segments with filling time equal to or greater than 30. Therefore, we put all the segments with Filling Time of 1 or 2 into the first bin, all the segments with Filling Time between 3 and 5 in the second bin, and so on. This gives us a histogram for Filling Time values. Figure 12b shows the Filling Time histogram of the edge map of a highway map (12a). It is clear that the size of bins with longer filling times is much larger. Similarly, Figure 13b shows that the distribution of Filling Time in a nonmap (Figure 13a) is very different, with the smaller filling times having slightly larger portion of all the segments. As with Filling Time, we divide the segments into 8 bins for Fork Count as well. The bins for Fork Count are as follows : segments with Fork Count of 0-1, segments with Fork Count of 2, 3, 4, 5, 6, 7, and finally segments with Fork Count equal to or greater than 8. Figure 12c shows the Fork Count histogram for the map. As we can see, there are large number of segments with high Fork Counts. Figure 13c on the other hand, shows the Fork Count histogram of the nonmap. This image does not have as many complex segments; but there are a large number of segments with low fork counts which is representative of the highly decomposed structure of a nonmap. We also divide the segments based on Water Amount. The bins for Water Amount are as follows : 0-2, 3-6, 7-11, 12-18, 19-24, 25-30, 31-37, 38- . We experimented with various bin sizes for all three features. In order to discriminate between the images in the best possible way, we have adjusted the boundaries of each bin in such as way that, for a property, their sizes are roughly equal when averaged over a large set of images that we used in our experiments. This implies that the variations in histograms among images will be most pronounced. Any other bin size configuration would tend to underestimate the difference between images since the variation would be smaller on average. This is similar in spirit to the concept of maximizing entropy in probabilistic systems. Therefore, we can say that range of bins has been chosen so as to separate segments such that change in the number of segments in any bin reflects a characteristic change in the nature of the image.

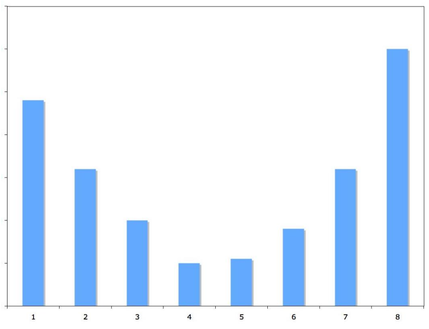
In this way, we generate eight values (corresponding to the eight bins) for each property (i.e. Filling Time, Fork Count and Water Amount) which combined together give us the 24 element feature vector for an image in the Water-filling feature space. If t1, t2, ... , t8 are the bin values for the Filling Time, c1, c2, ... , c8



(a)

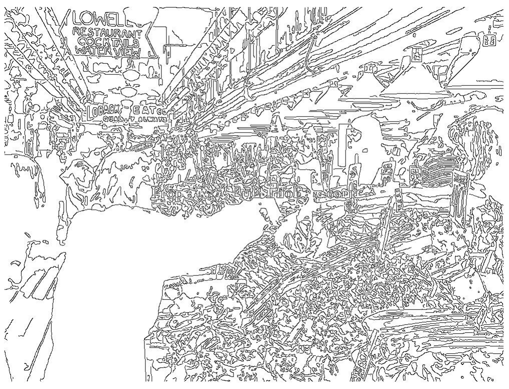


(b)

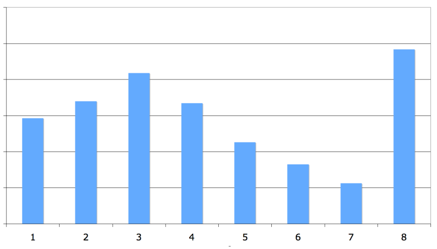


(c)

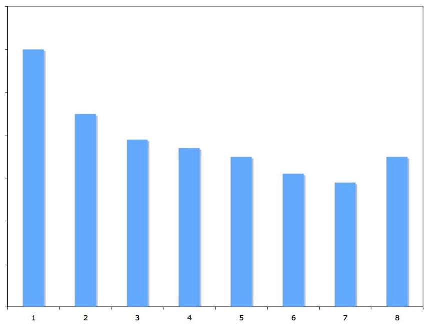
Fig. 12: A typical example of Filling Time histograms and Fork Count histograms for maps (left column, 12b, 12c) and nonmaps (right column, 13b, 13c).



(a)



(b)



(c)

Fig. 13: A typical example of Filling Time histograms and Fork Count histograms for maps (left column, 12b, 12c) and nonmaps (right column, 13b, 13c).

are the bin values for the Fork Count and a1, a2, ... , a8 are the bin values for the Water Amount, then the feature vector F = [t1 t2 t3 t4 t5 t6 t7 t8 c1 c2 c3 c4 c5 c6 c7 c8 a1 a2 a3 a4 a5 a6 a7 a8]

We implemented the feature extraction code in Java. Zhou et al. suggest that in order to find the next pixel to fill with water, we should look for all pixels that are 4-m neighbors of it. 4-m neighbors of a pixel are the four pixels located to the north, south, east and west of it. Due to the discrete nature of the pixels on the screen, occasionally, connected pixels occur diagonally (Figure 14). This causes the flowing water to reach a dead end at a point (circled in the figure) even though the line actually goes on further. As a result, features found by using this algorithm do not represent the characteristics of a segment to the fullest possible extent. The average Filling Time, and segment size is much lower according to this algorithm. This is why in addition to the 4-m neighbors, we consider the diagonally located pixels as well, in searching for the next connected neighboring pixels to fill. As shown in the experiments section, this small extension to Water-filling gives a much better estimate of the features that we intend to measure and improves the performance of the system considerably.

The size of images on the Internet varies widely. Our hypotheses is that for maps of a particular type, the ratio of number of segments of a particular Fork Count, Filling Time or Water Amount to the total number of

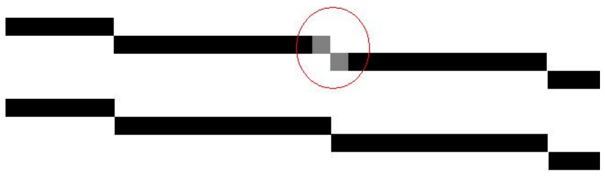
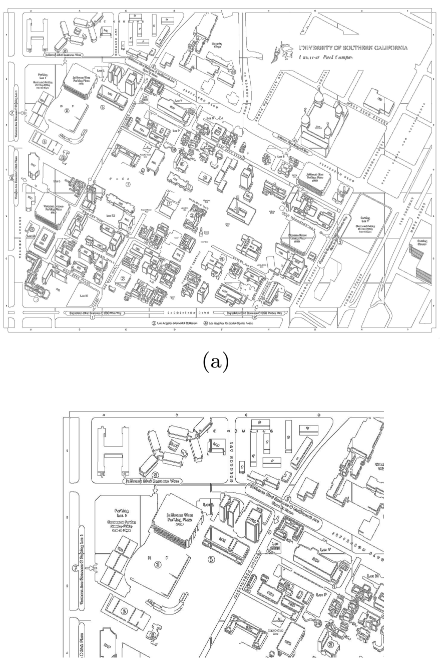


Fig. 14: Original WaterFilling algorithm terminates at points without 4-m neighbors even if the line logically extends further



(a)

(b)

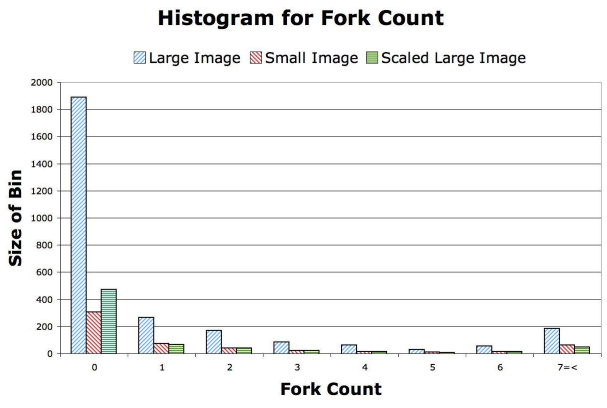
Fig. 15: The small map 15b is extracted from the larger map 15a

segments will remain roughly constant. The absolute number of segments will increase with the size of the image though. For example, consider the map shown in Figure 15a. We took the top left quarter from this image to create a new image Figure 15b. Figure 16a shows the bin sizes for both the images for various Fork Counts. As expected, the values for Figure 15b are smaller than those for the entire image. Yet, the relative distribution is identical in both images as in evident from the scaled down values of the larger image obtained by dividing them by four. In order to be able to match maps of the same type in our nearest neighbor classifier, we need to normalize the total number of segments in each image, so that the number of segment in each bin also becomes identical for similar maps. We have chosen to normalize the total number of segments in each image to 1000. We do this on each image in the following way. First, we calculate the actual sizes of bins for Filling Time, Fork Count and Water Amount. The sum of bin sizes for each property will be equal to the total number of segments in the image. Then we scale all the values so that this sum becomes equal to 1000. These resultant bin values are used to construct the feature vector of the image. The nearest neighbor classifier also uses the same values to calculate the distance between images in the feature space. Figure 16b shows the values of Fork Count bins after the normalization. Looking at the values, it becomes clear that both the histograms are identical and hence describe similar types of maps. When the nearest neighbor algorithm searches for similar images for one of the maps, the other map is very likely to be found as a close match, even though their sizes are very different.

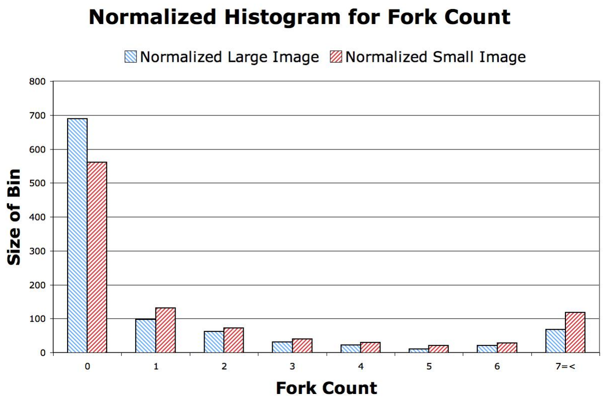
We would like to note here that the extraction of these features is very fast as compared to that of Laws' Textures which was used previously [3]. The system described herein takes less than 3 seconds to process a medium size image (1000 x 1000 pixels), versus a method relying on Laws' Texture, which takes about 30 seconds for the same image. Also, our feature vector has only 24 elements in it. This is a huge saving in space considering the fact that Laws' Texture generates more than 100 times this many elements in its feature vector, i.e. 3840. Therefore we gain an order of magnitude improvement in speed, combined with two orders of magnitude improvement in space. Further, as shown in our experiments, our features are more accurate as well. Thus we have built an efficient and accurate solution.

4.2 CBIR based k-Nearest Neighbor Classifier

There exist a wide variety of useful maps on the Web and the features of these maps are very different from each other. For example, a physical map doesn't have clear boundaries, whereas a street map has sharp lines for roads and blocks. Also, the same kind of map shows different densities at various scales. A street map at a low scale (Figure 17a) has large white space in between the lines of roads and other structures. On the other hand, a street map displaying the entire city (Figure 17b) will have a very grid-like structure with roads forming very small blocks and major freeways depicted

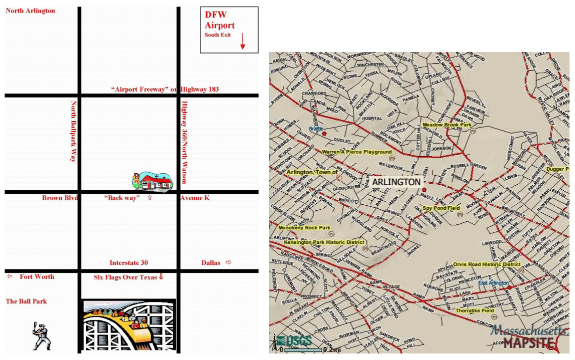


(a)



(b)

Fig. 16: Normalization for size invariance



(a) (b)

Fig. 17: Maps of Arlington, Virginia at different scales have very different structure. Zoomed-in (17a), zoomed-out (17b).

by bold lines. Therefore, it is not possible to find a common set of features among all these different maps which can reliably represent any map in general. These characteristics motivate the use of the CBIR-based k-NN classifier. As alluded to previously, this system finds the most similar images given a query image, from a repository of pre-labeled images. This action is in turn based on similarity of content (Figure 18).

A CBIR system finds the most similar images to a given query image from a repository of pre-labeled images based on the similarity in content between those images. Therefore, when we supply a particular map as query, the CBIR system finds those images that are closest to this map in features. If there are sufficient number of maps of this kind in the repository, then most of the similar images returned would be maps of the same type, since they would have the most similar features. CBIR is essentially the image equivalent to traditional text based Web search. Instead of finding the most similar Web pages for a given set of query terms, in CBIR the most similar images from a set are returned for a given query image. Figure 18 provides a visual representation of this concept. In this figure, the query image comes into the system and the most similar images are returned by it. This is the basis of our classifier.

To exploit CBIR for classification, we use a voting, k-Nearest Neighbor classifier [10]. We first use CBIR to find the nine most similar images (neighbors) from the combined set of images in the map and nonmap repositories based on the features mentioned above. We then employ a simple majority voting mechanism. If the majority of returned images are maps, we classify the query image as a map. If the majority of returned images are nonmaps, we classify the query image as a nonmap. Therefore, although it may be the case that other images on the Web will have clear edge structures (such as diagrams), since our technique relies not only on the edge features themselves, but also on the similarity to the edge features of the images in our map repository, such images will be filtered out. The accuracy of our experimental results indeed show this to be the case. We observed the performance of our system for different number of nearest neighbors chosen. We tried the following five different values: 5, 7, 9, 11 and 13, and found the variation to not be statistically significant for alpha = 0.5. Therefore, we decided to use the value of nine to maintain a balance between the complexity of finding the nearest neighbors and accuracy. If the number of nearest neighbors is too small, then, in case that a few nonmap images exist too close to some maps in a map cluster, they would swing the majority in the favor of nonmaps leading to error in classification. However, if the number of nearest neighbors used is big enough, then the other map images in the cluster will also get included and outweigh those nonmap images.

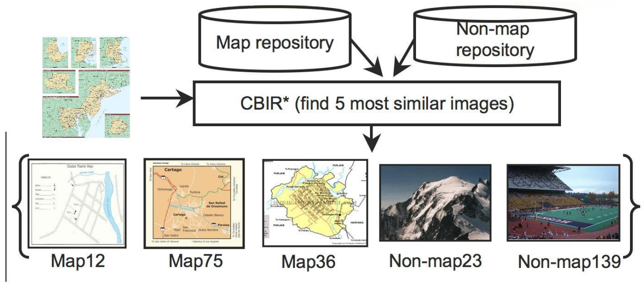


Fig. 18: A schematic of the CBIR method

Practically, we represent the repository of images by an index of feature values for each image and its name. We use the open-source CBIR system, Lire[6] to build and access this index quickly, though we have modified it to support our features. The CBIR system goes through the index calculating the similarity score of each image in the repository with the query image. The similarity score is calculated as follows: for each histogram, we add up the absolute difference between the corresponding bin values. Then we multiply each of these three values with their corresponding weights and add them together. The Lire system finds nine images with the least similarity score. These are the images which are most similar to the query image. In our implementation we have assigned equal weights to all the properties after trying a few combinations of weights, such as assigning weights of 1.0 to two properties and 0.5 to the third one. We also tried setting the weights of two properties to zero and of one property to 1.0. In all these cases, we found the performance to be lower than the case when the weights are equal.

We choose a CBIR based k-Nearest Neighbor classifier over other traditional machine learning methods for the following reasons. CBIR similarity methods allow us to exploit image similarities without explicitly modeling them. For instance, hydrography maps are similar to other hydrography maps and urban city maps are more similar to urban city maps but these types of maps may be quite different from each other. By using CBIR methods, these similarities can be exploited without modeling them explicitly because the returned images encompass the image similarities implicitly (that is why they are returned as similar).

In contrast, if we use traditional machine learning methods, such as Support Vector Machines, we would train a model for each class of map. Then if an incoming image matches any of these map classes, we know it is a map (since each class composes the image similarities). That is, we can take the hydography maps and learn a model of what a hydrography map should be. We can then take the urban city map and learn an urban city map model.

[6] http://www.semanticmetadata.net/lire/

There are several problems with trying to learn a model for each type of map. For one, the number of such classes is not known in advance. Therefore, a user will have to make a decision as to which maps constitute a class and hope he or she made the correct choices to lead to accurate classification. Along these lines, the user must make decisions as to the granularity of the classes (Is an urban-hydrographical map its own class ?), which can further complicate the creation of classes. Also, learning a new model may be a non-trivial process both in the work and time required. So, if a new class is discovered or needed, this can become a prohibitively costly problem since we need to train all the SVM models again with one more class to discriminate from. The other option is to train a two-class SVM as was done by Desai et al [3]. Such an SVM draws a hyperplane in the feature space to separate instances of maps from nonmaps. Yet, as explained earlier, since maps vary significantly in their properties, it is difficult to find a hyperplane that can clearly separate the two classes with minimal error since many nonmaps will have properties similar to a particular kind of map and lie close to them and maps of one kind might lie very far from maps of other kinds in the feature space.

The other reason we chose CBIR based k-NN instead of SVM has to do with robustness and scalability. As we show in Section 5, our classifier is able to learn models of different kinds of maps from a small set of labeled images. On the same data set, SVM performs relatively poorly since it requires more training data to learn a reliable model of maps from all the different kinds of maps shown to it. As we increase the training data size, SVM performs comparably to our classifier, with the caveat that SVM training becomes prohibitively expensive with increasing data set size. However CBIR techniques are built on methods from information retrieval which the major search engines have shown to be fast and robust in very large, practical settings. Further, we can freely tweak the size and composition of our repository to test the effect (something we do in the experiments to test this idea). Using machine learning, we have to retrain a model each time we tweak the training data set. In situations where training is costly, this is not a good solution. Therefore, by using CBIR methods we can grow the repository over time without retraining which allows for a scalable and incremental solution to classifying maps and building good map repositories.

4.3 Improvements over our previous work

We have done some preliminary work on harvesting maps from the Web [4]. Since then we have made several changes and additions to our methodology that we elaborate here. The first major difference concerns the Water-filling algorithm. In the previous approach we did not consider the diagonal neighbors while tracing the pixels to calculate various properties of each segment. In the present work we have modified the Water-filling algorithm to consider them as well leading to much more accurate measurements of these properties. The second change concerns the normalization of histogram values to fit the size of images. This makes our approach more size-invariant. We believe that both these changes have made the feature vectors become a better representative of the images.

In our earlier methodology, we considered two more properties suggested by Zhou et. al., namely, Maximum Filling Time and Maximum Fork Count, that we have done away with now. These represented the maximum values for Filling Time and Fork Count respectively in the whole image. We found that these properties are more relevant in an image of one solid object with a uniform background. In general maps and nonmaps, these properties have very random values due to the irregularity of these images. For example, the longest line in an edge map can be broken into half just by removing one pixel from the middle, thereby cutting the Filling Time by 50%. Similarly, two highly branching segments located close to each other can be joined by just a small line, which could be a random stroke. Yet, this will result in a segment with a very high Fork Count due to the addition of individual fork counts of the two segments. Therefore, these two features add noise to the CBIR matching process, reducing the effectiveness of other features. We also have made the weights of the histograms equal, in distinction to previous implementation where the Water Amount histogram was given lower weights.

These changes have contributed to significant improvement in both precision and recall of the system as illustrated in Section 5.

5 Experiments and Results

We use the Yahoo Image Search Engine API[7] to discover the images and the Google Search Engine to discover PDF documents from which the images can be extracted out. We implement the Water-filling [12] algorithm in Java. The images in the repository and their extracted features are indexed and retrieved using the open-source CBIR system Lire[8], which is in turn based on the popular text-based indexing and retrieval system Lucene[9] [15]. We extend Lire to support retrieval based on Water-filling features. The PDF parser is based on the official specification by Adobe[10].

The two major components of our classification process are the extraction of Water-filling features from the image and then its classification based on these features using CBIR based k-NN. To test the contribution of each component in our approach, we use four different experimental configurations, each of which isolates the different components of our approach. The "CBIR/WF" configuration is our full approach, combing Water-Filling (WF) features with our CBIR based k-NN classifier (CBIR). We also use our CBIR classifier with Laws' Texture [9] (LT) features, which we call "CBIR/LT". We also combine the binary SVM classifier with each of the feature sets, which we call "SVM/WF" for the case that uses Water-Filling and "SVM/LT" for that which uses Laws' Textures. By combining each of the classifiers with each of the feature choices we can isolate the impact of the choice of features and the choice of classifier. The different configurations with their individual components are summarized in Table 1.

Table 1: Four different configurations to isolate the effects of the features and the classifiers

Name of configuration	**Feature**	**Classifier**
CBIR/WF (MapFinder)	Water-filling	CBIR based k-NN
CBIR/LT	Laws' Texture	CBIR based k-NN
SVM/WF	Water-filling	SVM
SVM/LT	Laws' Texture	SVM

Our experimental methodology is as follows. We download images located at the URLs obtained from the Yahoo Image Search Engine for 16 different regions of the world. In order to maintain an equal distribution between different regions, we randomly pick 500 images from each of these collections of images for the purpose of our experiments, except for four cities for which we could not find enough unique images. We label each image as a map or a nonmap based on the criteria mentioned before. The distribution of the images among different regions and the number of maps and nonmaps for each is depicted in Table 2. The "non-

[7] http://developer.yahoo.com/search/image/V1/imageSearch.html

[8] http://www.semanticmetadata.net/lire/

[9] http://lucene.apache.org/

[10] http://www.adobe.com

map" images mentioned in the table are taken from the CALTECH 101 data set [14].

Table 2: Distribution of images by source

Source of image (Keyword used)	# images	# maps	# nonmaps
africa	500	263	237
arlington	500	296	204
asia	500	298	202
baghdad	500	274	226
bangalore	450	241	209
bellevue	500	329	171
chicago	450	354	96
citymap	500	490	10
frankfurt	500	244	256
los angeles	500	256	244
new delhi	350	232	118
new york	500	145	355
nonmap	1000	0	1000
pittsburgh	500	284	216
seattle	500	245	255
shanghai	500	308	192
tehran	330	91	239
ALL	*8,580*	*4,350*	*4,230*

For our experiments, we randomly selected 2000 maps and 2000 nonmaps from the entire set of images to build the repository for the CBIR method. Since the repository acts as the set of labeled samples for the CBIR method, we used this same set to train the binary SVM. Then we tested each method on 2000 randomly chosen maps and 2000 randomly chosen nonmaps that were different from the images in the repository/training set. We also wanted to see if our classifier retains the lead over other systems for smaller repository sizes. To test this, we carried out the same experiment as described above, but with repeatedly smaller repository/training sets. The test set was kept fixed in size and content. The performance of each configuration was measured in terms of Precision[11], Recall[12] and F1-measure[13]. Both the training and testing data set were then returned to the entire set of images. This procedure was repeated 10 times, each time selecting a fresh training and testing data set, sampled randomly from the entire set. The average performance, over 10 runs, of each configuration on the different repository sizes is shown in Table 3.

This table supports three hypotheses. First, it shows that Water-filling is very good at capturing the general characteristics of a map and is therefore a much better feature in discriminating between maps and nonmaps than Laws' Texture used in the previous work by Desai et al. [3]. With regard to both classifiers, namely CBIR based k-NN and Support Vector Machine, the precision, recall and F1-measure for systems using Water-filling features is higher than those using Laws' Texture. When we look at the first two systems, CBIR/WF and CBIR/LT, in Table 3, both of which use CBIR based k-NN, the F1-measure is almost 6% higher with the use of Water-filling features. In case of the next two systems, SVM/WF and SVM/LT, the difference is as high as 17%. This clearly shows that just changing the feature to Water-filling enhances the accuracy of classification.

Table 3: Performance of all four configurations with varying repository/training set sizes. (P=Precision, R=Recall, F1=F1-measure)

Type of configuration	P	R	F1
4000 images in training set			
CBIR/WF (MapFinder)	77.39	71.20	**74.17**
CBIR/LT	70.23	67.77	68.20
SVM/WF	81.41	67.00	73.51
SVM/LT	69.23	47.62	56.43
3000 images in training set			
CBIR/WF (MapFinder)	76.47	72.00	**74.17**
CBIR/LT	69.34	66.83	68.06
SVM/WF	81.12	65.95	72.75
SVM/LT	69.10	47.08	56.00
2000 images in training set			
CBIR/WF (MapFinder)	76.67	70.40	**73.39**
CBIR/LT	68.59	67.19	67.89
SVM/WF	80.93	63.65	71.25
SVM/LT	68.85	45.36	54.69
1000 images in training set			
CBIR/WF (MapFinder)	75.87	69.80	**72.71**
CBIR/LT	67.47	66.29	66.90
SVM/WF	80.36	60.75	69.16
SVM/LT	69.27	41.14	51.62
500 images in training set			
CBIR/WF (MapFinder)	75.52	64.83	**69.77**
CBIR/LT	66.69	64.60	65.63
SVM/WF	79.94	55.80	65.72
SVM/LT	69.08	38.12	49.13
250 images in training set			
CBIR/WF (MapFinder)	76.71	58.80	**66.57**
CBIR/LT	64.47	64.67	64.53
SVM/WF	79.32	51.60	62.53
SVM/LT	69.84	33.26	45.06

The second hypothesis is that on a small training set, CBIR based k-NN performs better than the conventional Support Vector Machine. The first and third classifier/feature configuration in each group use Water-filling features, but differ in the type of classifier. The F1-measure for CBIR based system is almost 4% more than that for the SVM when the training set has between 250 and 500 images. In fact, the difference re-

[11] Precision(P) = Number of images correctly identified as maps / Total number of images identified as maps

[12] Recall(R) = Number of map images correctly identified / Total number of map images

[13] F1-measure(F1) = 2 x P x R / (P + R)

mains significant till the repository size grows to 3,000 images after which it becomes less that 1%. Since maps vary a lot in their properties, and yet the SVM learns one label for all kinds of maps, it requires more data to converge to a good hyperplane that separates maps from nonmaps. On the other hand, the CBIR makes use of these differences to identify different kinds of maps and thus has a recall of 59%. The SVM meanwhile has a recall of 52%. When we look at the second and the fourth configurations, both of which use Laws' Texture, the difference is more marked. Whereas the difference in F1-measure is 12% for the larger training sets, the difference becomes as significant as 20% for the smallest sets. This demonstrates the fact that CBIR based classifier is superior to SVM in this application, even with the use of a suboptimal feature set.

The third claim we made in the beginning of the paper was that our classification system improves upon the current state-of-the-art by almost 20% in F1-measure (for the largest training set). The big gain in accuracy is the result of combining a better discriminating feature set (Water-filling) with a more flexible classifier (CBIR based k-NN). In the process, we gain on the speed of classification as well as the maintainability of the system as discussed in detail in section 4.2. All the values of F1-measure reported in the table above are statistically significant.

Figure 19 shows the change in F1-measure of the four configurations with the size of the repository/training set. Our system maintains a consistent advantage over the other systems for all sizes of the repository. This shows the strength of our method. Even with a repository size as small as 250 images, it is able to classify 4000 images correctly with a precision of 77% and F1-measure of 67%. This means that our system is able to capture the most fundamentally unique features of a map which enables it to correctly classify a much more diverse set of images.

Figure 20 shows the precision, recall and F1-meausure of only our system with change in the repository size. The precision of our system remain fairly constant even when the repository size is very small. It is the change in recall which increases the F1-measure for the most part. This implies that one can start collecting maps with a very small group of images in the repository. Since the precision is high, the images classified as maps by this system can then be put back into the repository with minimal human intervention to grow the size of the repository for better performance. This process is also called bootstrapping.

To test the robustness of our classifier on images unlike those in the repository, we divide the images into the repository and test set based on their origin. For

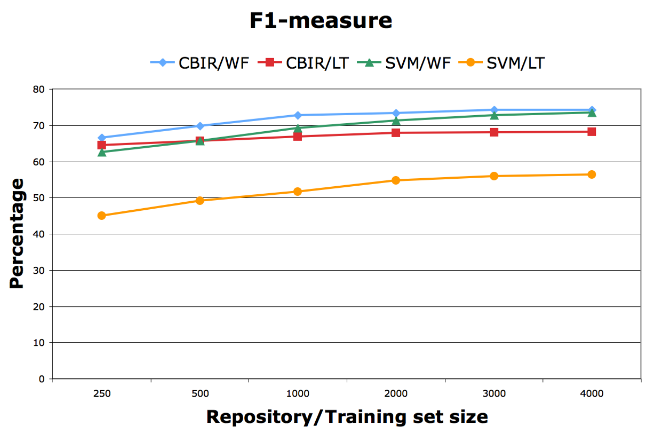


Fig. 19: F1-measure of the four configurations

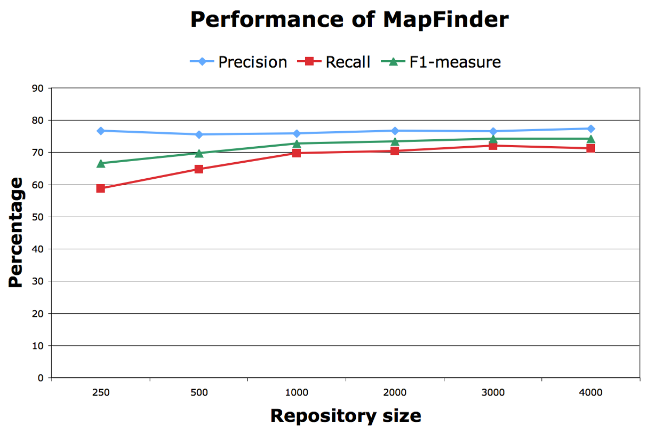


Fig. 20: Performance of MapFinder with different repository sizes

example, US cities in general have a more regular grid based structure to them which is not found in the other parts of the world like India, Iran or China. Therefore, we put all the US city images (for example, Figure 21) in the repository and test the performance of the classifier on the set of images belonging to regions outside US (for example, Figure 22). We also divide the images based on the size of region they represent. In this case, we put all the city images (for example, Figure 21 and Figure 22) in the repository and test the accuracy of classification on maps of continents (for example, Figure 23). We compare the performance on the two distributions of images mentioned above with the standard distribution, in which the repository and test sets are composed by picking uniformly from the entire data set. Both the repository and test sets have 4000 images, 2000 being maps and the other 2000, nonmaps.

The precision, recall and F1-measure of classification are presented in Table 4.

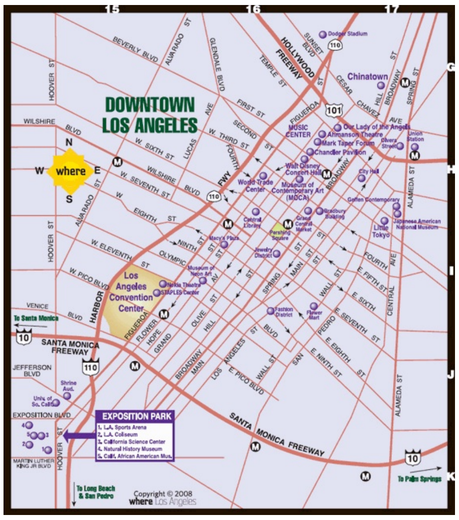


Fig. 21: Example of a US city map. It has regular grid-like structure that is missing in cities of other countries such as India.

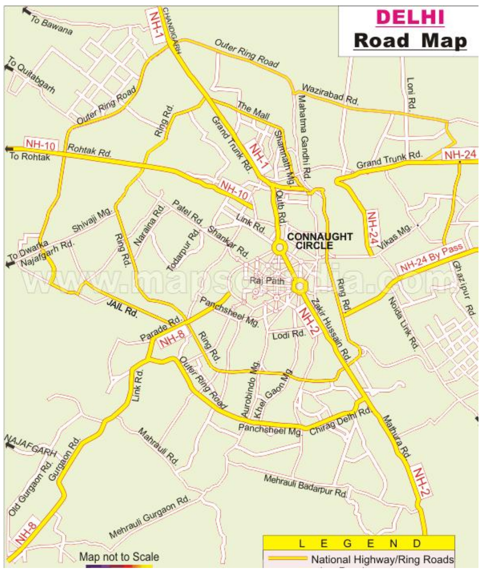


Fig. 22: Example of a map of a city in India.

Fig. 23: Example of a map of a continent.

Table 4: Performance of the classifier tested on images totally different from the repository

Division of images into repository and test set	**Precision**	**Recall**	**F1-measure**
Repository - US cities Test set - non-US cities	76.19	62.55	68.70
Repository - Cities Test set - Continents	75.51	51.80	61.45
Both repository and test set picked uniformly	77.39	71.20	74.17

The results show that the precision remains practically unchanged. From the point of view of a real world implementation of our algorithm, this implies that the user of this system can put the same confidence in the classification of a map from any part of the world even if the classifier has seen that area for the first time. Combining this with the fact that the algorithm maintains its superiority to others even when the repository is very small makes MapFinder the most practical alternative to any existing system for identifying maps.

5.1 Improvement from the previous implementation

Since we introduced MapFinder in our previous paper [4], we have experimented extensively with all the major properties of this system. We have made several major change that are explained in Section 4.3. In order to evaluate how all these changes have translated into improvement in performance of MapFinder, we run the same basic experiment as explained before on its previous implementation as well. Table 5 shows the results.

Table 5: The improvement in performance of MapFinder since previous paper

Implementation	**Precision**	**Recall**	**F1-measure**
Current	77.39	71.20	74.17
Previous paper [4]	67.79	57.98	62.50
Improvement	+9.60	+13.22	+11.67

There is an almost 12% improvement in F1-measure of the system, a direct result of the algorithmic improvements outlined in this paper. The precision and recall have improved by almost 10% and 13% respectively. This demonstrates two facts. First, MapFinder is more able to differentiate between the maps and nonmaps than previously. This is attributed to a better selection of properties (e.g. rejection of Maximum Filling Time and Maximum Fork Count) and the choice of their weights, which lets the CBIR system select the neighbors more accurately. Second, the significant increase in recall shows that MapFinder can now capture the fundamental characteristics of the maps more accurately. This is a direct consequence of improved features, arising from the adaptation of the Water-filling algorithm to our task. The values of precision, recall and F1-measure mentioned here for the previous implementation, based on the new data set, are lower than those reported in our previous paper because the current data set is larger and much more diverse than the previous set and is a more accurate representation of the images in the wild.

5.2 Extracting maps from documents such as PDF files

We believe that PDF documents are a good source of images and maps of a region. To test this, we queried the Google Search Engine, with search queries such as "Los Angeles maps filetype:pdf", "Africa maps filetype:pdf" etc. We downloaded 210 PDFs in this way. We have developed a PDF parser which extracts images embedded in the PDF files in a wide variety of formats and encodings. We extracted 2266 images from all the PDF files. After removing images which were too small (e.g. company logo, legend symbols for maps), we were left with 310 images. The reason for pruning is two-fold. One, maps that are too small are not of much use since they are either too dense or contain too little information for any practical use. Also, small images do not contain enough pixels to let the MapFinder algorithm reliably extract its features. We consider any image that has both its width and its height less than 300 pixels to be too small for use. After manual labeling, we divide these 310 useful images into 182 maps and 128 nonmaps. This shows that the PDFs returned by the search engine indeed contain useful maps (182 maps in 210 PDFs). The performance of MapFinder on these 310 images was measured in terms of Precision, Recall and F1-measure. As before, the repository was created by picking 2000 map and 2000 nonmap images randomly from our entire set of images. We measured the performance three times by creating fresh repositories each time. Table 6 shows the distribution of map and nonmap images in the downloaded PDFs along with the average values of Precision, Recall and F1-measure observed.

Table 6: Harvesting maps from PDFs

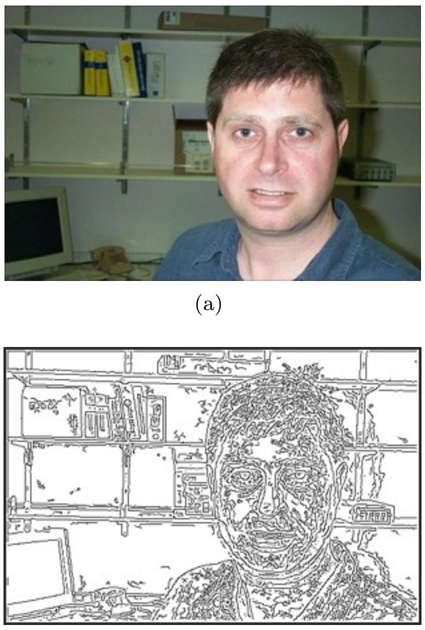
Number of PDFs	210
Total extracted images	2266
Useful images	310
Actual maps	182
Actual nonmaps	128
Classification by MapFinder	
Precision	67.07
Recall	60.44
F1-measure	63.58

This table supports our claim that it is possible to harvest maps from documents like PDF files, over and above the independently available images on the Internet. Although MapFinder is still able to classify these images reliably, the performance for the embedded images is lower than that for the individual images. As we mentioned earlier, the images embedded in PDFs are generally of a much higher quality. In fact, many of the images are stored in a compressed bitmap format, which when extracted produce a high resolution image. In the process of classification in MapFinder, we first convert the images to the GIF format for implementation reasons. Due to the high quality and richness of colors in the extracted bitmaps, the conversion to GIF format leads to the formation of color edges from the color gradients. These give maps a more nonmap-like property (large number of small edges). It is important to note here that this problem is not inherent in the algorithm, but is a side effect of the choice of implementation.

Overall, these experiments verify and support the efficacy of the different parts of our end-to-end system for harvesting maps from the World Wide Web.

6 Future Work

The most important issue we will address in the future is that some maps are misclassified by our system when the majority vote is borderline, for example, when the number of maps is four and number of nonmaps is five. We have found that the one image that sways the classification is often among a very small set of nonmaps ($\approx$10), which repeatedly get selected by the CBIR system because they have a map like quality to them. Figure 24a, shows one such nonmap. The edge map 24b, has many squares in the background, which is similar to the block structure found in cities. We can deal with this ambiguity by employing relevance feedback techniques from information retrieval. Such relevance feedback could help us to identify and prune away such "culprit" images that consistently sway the vote in misclassifications.


(a)

(b)

Fig. 24: A typical culprit image which consistently sways the vote towards misclassification because of its map-like properties.

Even though we have changed the weights given to the features from our previous work, so that we ignore two of them and make the remaining three equally important, we have not tried changing the weights for each bin of the histograms individually. It does seem possible that some of the bins could have a more significant role in differentiating maps from nonmaps than others. Experiments can be carried out to determine the strength of each of these bins by extensively changing weights for them in steps. We think that it will be possible to achieve even higher precision through these experiments.

Our results point to the necessity of using the correct features for CBIR. While we chose Water-filling, which is good for images such as maps with strong edge maps, other methods could perhaps work as well. For instance, the "salient point" features based on wavelets [21] and another set of features based on shape similarity [22] could be well suited to our task, since maps seems to share certain shapes within them. Lastly, methods have been proposed to more efficiently store color information [23], which makes retrieval more efficient. Although we use textures based on edge maps to make our method color invariant, color information might help in discriminating maps. It will be interesting future work to compare the various features and their efficiency and accuracy for map classification.

7 Conclusion

In this paper we present an autonomous, accurate and scalable method for harvesting maps from the World Wide Web. Our method first scours the Web to find both stand-alone images and those embedded within documents. Then, our classification algorithm separates these discovered images into a set of maps and nonmaps for the query region. We find that Water-filling features are much better at capturing the features of maps that differentiate them from nonmaps. We also find that a CBIR based classification method outperforms an SVM based method when trained on fewer labeled samples. We show the robustness of our classifier by using a repository of US city maps to identify maps of other regions outside the US that do not even have a similar grid-like structure. Finally, it is demonstrated that the combination of a superior feature set and classifier contributes to an overall system which outperforms the current state-of-the-art.

By extracting images from PDFs, we collect new high quality maps that are generally not available as independent files. In fact, we find that the percentage of images that are maps, is higher in PDFs than among the independent image files obtained from image search engines.

In addition to being very accurate, our system is much more scalable as well, since adding more images to the repository for better performance requires only extracting their features and adding them to the index. On the contrary, an SVM classifier needs to be retrained every time a new image is added to the training set, which for a sufficiently large set requires significant time. Further, our classifier needs a fraction of the time and memory required by previous work [3].

We have suggested some work which can further improve the performance of the system. Yet, despite the future work proposed, our technique provides an automatic, accurate, practical and scalable solution to the problem of creating useful map repositories from the Web. More importantly, by plugging our method in with methods for aligning raster maps with satellite images we can build effective GIS systems by scouring the freely available images on the Web to build map collections for any given region in the world.

8 Acknowledgements

We thank the anonymous reviewers for their helpful comments and suggestions on improving this paper.

This research is based upon work supported in part by the National Science Foundation under award number IIS-0324955, in part by the United States Air Force under contract number FA9550-08-C-0010, and in part by a gift from Microsoft. The U.S. Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

References

1. Chen, C.C., Knoblock, C.A., Shahabi, C.: Automatically conflating road vector data with orthoimagery. GeoInformatica 10(4) (2006), pp. 495-530
2. Chen, C.C., Knoblock, C.A., Shahabi, C.: Automatically and accurately conflating raster maps with orthoimagery. GeoInformatica 12(3) (2008), pp. 377-410
3. Desai, S., Knoblock, C.A., Chiang, Y.Y., Desai, K., Chen, C.C.: Automatically identifying and georeferencing street maps on the web. In Proceedings of the 2nd International Workshop on Geographic Information Retrieval (2005), pp. 35-38
4. Michelson, M., Goel, A., and Knoblock, C.A.: Identifying Maps on the World Wide Web. In Proceedings of the 5th International Conference on Geographic Information Science (2008), pp. 249-260
5. Chiang, Y.Y., Knoblock, C.A., Shahabi, C., Chen, C.C.: Accurate and automatic extraction of road intersections from raster maps. Geoinformatica 13(2) (2009), pp. 121-157
6. Chiang, Y.Y., Knoblock, C.A: Classification of line and character pixels on raster maps using discrete cosine transformation coeffients and support vector machines. In Proceedings of the 18th International Conference on Pattern Recognition (2006), pp. 1034-1037
7. Chiang, Y.Y., Knoblock, C.A., Chen, C.C.: Automatic extraction of road intersections from raster maps. In Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems (2005), pp. 267-276
8. Smeulders, A.W.M., Worring, M., Santini, S., Gupta, A., Jain, R.: Content-based image retrieval at the end of the early years. IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (2000), pp 1349-1380
9. Laws, K.: Textured Image Segmentation. Ph.D. Dissertation, University of Southern California, January 1980.
10. Dasarathy, B.V.: Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques ISBN 0-8186-8930-7 (1991)
11. Fix, E., Hodges, J.L.: Discriminatory analysis, nonparametric discrimination: Consistency properties. Technical report 4, USAF School of Aviation Medicine, Randolph Field, TX (1951)
12. Zhou, X.S., Rui, Y., Huang, T.S.: Water-filling: A novel way for image structural feature extraction. In Proceedings of the International Conference on Image Processing (1999), pp. 570-574
13. Vapnik, V.: The Nature of Statistical Learning Theory. Springer-Verlag, 1995. ISBN 0-387-98780-0
14. Fei-Fei, L., Fergus, R., Perona, P.: Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In Proceedings of IEEE CVPR Workshop on Generative-Model Based Vision (2004)
15. Lux, M., Becker, J., Krottmaier, H.: Caliph emir: Semantic annotation and retrieval in personal digital photo libraries. In Proceedings of CAiSE '03 Forum at 15th Conference on Advanced Information Systems Engineering (2003), pp. 85-89
16. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Analysis and Machine Intelligence 8 (1986) pp. 679-714
17. Csillaghy, A., Hinterberger, H., Benz, A.O.: Content based image retrieval in astronomy. Information Retrieval 3(3) (2000), pp. 229-241
18. Wang, Z., Chi, Z., Feng, D.: Fuzzy integral for leaf image retrieval. In Proceedings of IEEE Intl. Conference on Fuzzy Systems (2002), pp. 372-377
19. Müller, H., Michoux, N., Bandon, D., Geissbuhler, A.: A review of content-based image retrieval systems in medical applications. Clinical benefits and future directions. International Journal of Medical Informatics 73 (2004), pp. 1-23
20. Lehmann, T.M., Guld, M.O., Deselaers, T., Keysers, D., Schubert, H., Spitzer, K., Ney, H., Wein, B.B.: Automatic categorization of medical images for content-based retrieval and data mining. Computerized Medical Imaging and Graphics 29 (2005), pp. 143-155
21. Tian, Q., Sebe, N., Lew, M.S., Loupias, E., Huang, T.S.: Image retrieval using wavelet-based salient points. Journal of Electronic Imaging 10(4) (2001), pp. 835-849
22. Latecki, L.J., Lakamper, R.: Shape similarity measure based on correspondence of visual parts. IEEE Trans. Pattern Analysis and Machine Intelligence 22(10) (2000), pp. 1185-1190
23. Deng, Y., Manjunath, B.S., Kenney, C., Moore, M.S., Shin, H.: An efficient color representation for image retrieval. IEEE Trans. Image Processing 10(1) (2001), pp. 140-147
24. Qingzhao Tan, Prasenjit Mitra, C. Lee Giles: Effectively Searching Maps in Web Documents. In Proceedings of European Conference on Information Retrieval (2009), pp. 162-176