Mother Fugger: Mining Historical Manuscripts with Local Color Patches

Qiang Zhu
Eamonn Keogh
qzhu@cs.ucr.edu
eamonn@cs.ucr.edu
Dept. of Computer Science & Engineering, University of California, Riverside, CA 92521

***Abstract*— Initiatives such as the Google Print Library Project and the Million Book Project have already archived more than ten million books in digital format, and within the next decade the majority of world's books will be online. Although most of the data will naturally be text, there will also be tens of millions of pages of images, many in color. While there is an active research community pursuing data mining of text from historical manuscripts, there has been very little work that exploits the rich color information which is often present. In this work we introduce a simple color measure which both addresses and exploits typical features of historical manuscripts. To enable the efficient mining of massive archives, we propose a tight lower bound to the measure. Beyond the fast similarity search, we show how this lower bound allows us to build several higher-level data mining tools, including motif discovery and link analyses. We demonstrate our ideas in several data mining tasks on manuscripts dating back to the fifteenth century.**

Keywords-Historical Manuscripts; Color Indexing

I. INTRODUCTION

In the last decade, million of books, maps and historical manuscripts have been digitized and made freely available [12]. It is now clear that within a few years a significant fraction of world's books will be online. While these digitized texts will be an invaluable resource for researchers to browse and search, we feel that the additional step of *mining* these manuscripts will reveal new insights, knowledge and historical context. In a similar vein, technology writer Kevin Kelly has observed, "*the real magic will come in the second act, as each word in each book is cross-linked, clustered, cited, extracted, indexed, analyzed, annotated, remixed, reassembled and woven deeper into the culture than ever before*" [16]. While this remark explicitly singles out text, a similar argument can be made for *images*. Clearly most of the data gleaned from scanned books will be text, but there will also be tens of millions of pages of images. Many of these images will defy automatic annotation for the foreseeable future, and CAPTCHA-based [29] or crowdsourcing efforts [14] are likely to have only limited impacts on these problems. A considerable fraction of the images may, however, be amiable to annotation by algorithms that can exploit the color information which is often available in historical manuscripts.

While there is much work on indexing images by color, most of it assumes that we are indexing atomic images, typically photographs. For example, much work has been done to support queries such as "*find photos of sunsets*" [19] or commercially useful queries such as "*find shoes in this color*" [17]. In contrast, here we are interested not in *global* color queries, but *local* region-of-interest queries, which we call *patches*. In Figure 1 we show some examples of the type of images[1] we are interested in mining.

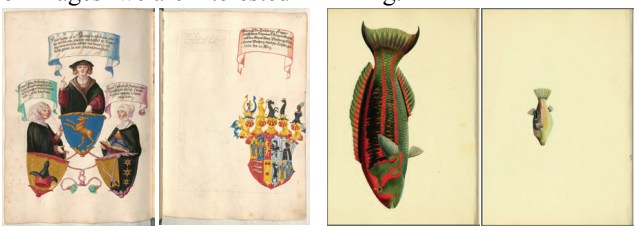


Figure 1. *left*) Two pages from *The secret book of honour of the Fugger* 1545-1548 [4]. *right*) Two pages from an 1834 text on fish [1].

Here we can imagine an historian might be interested in searching a massive archive to discover examples of the coat of arms of Lucas Fugger, which consists of a golden deer on a light blue background. In Figure 1.*left* we have a *local* patch that matches this description, but note that the *global* color of the page is dominated by other colors.

While there is much less work on local color matching, there is some. For example, in [26] the authors proposed a technique called *Histogram Backprojection* to determine the location of a query object in *one* image (provided that the query appears in that image), and in [5] the authors described a scale and translation invariant image retrieval system. However, these works, and their many extensions [20] only consider *query-by-content* for *main memory search*.

[1] Because of the nature of the problems we are considering, many of the figures in this work are best viewed in color. A color version is online at [27].

In contrast, we are interested in more *general data mining* for local color regions, and for datasets which are so large that they must be *disk-resident*. As we shall show, existing color matching measures work well for historical manuscripts, but the existing works are severely limited in their scalability, especially if we look beyond query-by-content and consider more complex data mining operations.

In Figure 1 we show an observation which we plan to exploit to mitigate the scalability problems. Note that while the leftmost page from each manuscript can be considered somewhat "dense", the rightmost pages are relatively "sparse". As we shall show in Sections IV and V, by adaptively exploiting this variability of density, we can prune off a huge fraction of the search space, and we can produce much smaller indices, both of which in turn can allow us to answer queries much more efficiently.

The rest of this paper is organized as follows. In Section II we discuss the necessary background material. In Section III we introduce a lower bound that is at the heart of our search, and algorithms that exploit it are introduced in Section IV. Section V sees extensive empirical evaluation of our ideas, and we offer conclusions and directions for future research in Section VI.

II. NOTATION AND BACKGROUND

Before describing our algorithm, we first give the following definitions.

In the domain of our interest, the natural atomic unit is a single image or *page*:

Definition 1: A *Page* is a color image P, which is represented as a $H \times W \times 3$ matrix where H and W are the height and width of the page, respectively. Each pixel of the page corresponds to a triplet (R, G, B) in the matrix P, denoting the value in the red, green and blue channels, or the pixel may contain *NaN* to indicate transparency.

We use the RGB color space because it is familiar and we found empirically it works very well in our domain. However, all the lower bounds and algorithms introduced in this work can work in any color space. Note that we allow the pixel to have a *NaN* value indicating no color in that pixel. We use this value to replace the background color of the page at a preprocessing step. As we shall see later, removing the background color allows us to avoid the trivial solution of matching backgrounds when we want to find repeated patterns (motifs). Automatically finding the background color of an historical text is an important subroutine in other problems (OCR etc.) and is an essentially solved problem for all but the most pathological texts [8].

As noted in Section I, we are not interested in page-level queries, instead we want to enable local region-of-interest queries, which are more challenging and useful. We define these lower level units as *color patches*:

Definition 2: A *Color Patch* is a rectangular window CP in a page. CP can be specified by four variables: the row r and the column c in P of its top left corner, its height h and width w. Then $CP = P(r:r+h-1, c:c+w-1, :)$. Here $1 \le r \le H-h+1$ and $1 \le c \le W-w+1$.

Note that our basic primitive is a *rectangle*. We had considered allowing more general shaped patches (circles, triangles, etc). However, as we shall show empirically, rectangles allow the effective retrieval of shapes which are

clearly non-rectangular (fish, butterflies, coats-of-arms, etc), so we restrict our definitions to rectangles for simplicity.

Also note that we can consider a *whole* page as a color patch, as a special case. Finally, we define the higher level unit that organizes pages, a *book*:

Definition 3: A *Book* is a set *B* containing *N* pages. Note that due to the scanning process, and the handcrafted nature of some historical manuscripts, the size of pages from the same book may not be exactly the same.

Having this three-level hierarchy (*color patch*, *page* and *book*) in mind, we are now in a position to define the *nearest neighbor* of a color patch.

Definition 4: Given a color patch Q and a book B, the *Nearest Neighbor* of Q in B is a color patch NN with the same size of Q and most similar to Q. More formally, for any color patch CP with the same size of Q in B, $\text{Dist}(Q, NN) \leq \text{Dist}(Q, CP)$.

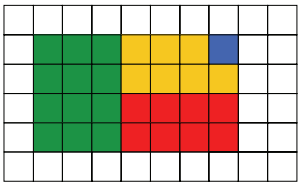
In definition 4, we have not yet given an explicit distance measure between two color patches. While a number of color descriptors (representations) exist, the *Color Histogram* is one of the most widely used. It is easy to calculate, invariant to translation, insensitive to small changes in viewpoint (angle and distance), and has been proven to be effective in several content based image retrieval (CBIR) systems [7] [25] [26].

Definition 5: The *Color Histogram* of a color patch *CP*, denoted as Hist(*CP*), is a vector which counts the frequency of each possible color in a given *color space*.

In a RGB color space which has 8 bits for each channel, the total number of colors is $(2^8)^3$. Maintaining so many colors is impractical and unnecessary. Furthermore, it is likely to *hurt* accuracy, since two colors that are imperceptibly different to the human eye may map to adjacent but different bins, thus accumulating significant distance. One solution to mitigate this problem is to use a "warping-like" measure [15]; however, this will require dramatically more computational power per distance calculation, and make it difficult to design an effective lower bounding search. Therefore, we use *quantization* to alleviate the abundance of colors problem.

In keeping with the majority of the literature, colors are quantized into K *bins*. The value of K ($\ll 2^{24}$) is a user-defined parameter choice, but the exact value does not matter too much from several dozens to several thousands [26].

In Figure 2 we illustrate the notations introduced thus far with a toy example.



0	0	0	0	0	0	0	0	0	0
0	2	2	2	4	4	4	3	0	0
0	2	2	2	4	4	4	4	0	0
0	2	2	2	1	1	1	1	0	0
0	2	2	2	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0

Figure 2. An illustration of some basic notations. *left*) A 6×10 *page* showing a (slightly modified) Benin flag. The 4×3 green window on the left is one of its *color patches*. *right*) A matrix of the same size as the page, with the number indicating the bin_id of corresponding pixels. For simplicity, we set $K = 4$ and assign one bin for each color. Note that for surrounding transparent pixels, we set bin_id = 0. The *color histogram* can be easily obtained from this matrix, which is {**8**,**12**,**1**,**7**}.

Having determined that the color histogram is the most suitable descriptor for color patches, we must consider the most appropriate similarity/distance measure for them. Not surprisingly, lots of measurements for color histograms have been proposed in the last three decades [19][23][24], including simple ones such as *Euclidean Distance* and *Histogram Intersection* [26], and more complex techniques such as *Histogram Quadratic Distance* [7], *Histogram Refinement* [21], and *Color Set Distance* [25], etc. After extensive testing on our domain of interest we chose the *Histogram Intersection*, because of both its (relative) efficiency and accuracy. Note however that the focus of this paper is not to make an argument about the "best" color distance measure in general, but rather to show that we can adapt and augment a popular distance measure to solve data mining problems in a domain that has received surprisingly little attention.

Definition 6: Given a query color patch Q and a candidate color patch C, the *Histogram Intersection* between them is defined as:

$$HI(Hist(Q), Hist(C)) = \sum_{i=1}^{K} \min(Hist(Q)[i], Hist(C)[i]) \quad (1)$$

In the above equation, K is the number of bins in both Hist(Q) and Hist(C). The *HI* value tells how many pixels are in common (falling into the same bins) between Q and C. We can simply use its inverse as a distance measurement[2]:

$$Dist(Q, C) = \frac{1}{HI(Q, C)} \quad (2)$$

It may be noted that our proposed distance measure does not encode the spatial *relationships* of the colors within the query region. As it happens, it is this fact that allows us to create tight lower bounds to the measure, thereby enabling our fast search and mining algorithms. However, this is merely a fortunate side effect. The need for spatial relationship invariance emerged from conversations with domain experts. In Figure 3, we can see two motivating examples: While the parity of left/right orientation typically does have meaning for *some* heraldic shields, it is often ignored, as in the examples from a 16th century book shown in Figure 3 (*left*). Likewise, while most organisms are bilaterally symmetric, they may be illustrated at arbitrary orientations, as in the two examples show in Figure 3 (*right*).

Figure 3. A section of images to demonstrate the need for spatial-location invariant distance measures. From left to right: in page 109 of [4], in page 108 of [4], an image of a Death's Head Moth from 1849 [3], and one from 1860 [28].

[2] For simplicity, Dist(Q,C) is short for Dist(Hist(Q),Hist(C)), the same applies to HI(Q,C) in the equation (2). Without ambiguity, we will use these notations in all the remaining text.

In spite of the above, there are some circumstances where color locality matters; Figure 4 shows some examples.

Figure 4. A section of image pairs that are clearly distinct, but essentially indistinguishable under the distance measure introduced in equation (2).

In the event that spatial sensitivity is necessary in a particular domain, we can achieve this by dividing the query region into (for example) four quadrants, and defining a new measure as the sum of the four local measures. However, because our domain experts, genealogists, an ichthyologist and an historian of science, did not find a compelling example where spatial sensitivity was critical to the success of query-by-content or a higher order data mining query in their domains, we defer a further discussion of this idea to an online appendix [27].

Our color measure is simple, and the domain of interest does feature the complex issues of staining, fading, degradation, and problems where very subtle distinctions are required. The reader may wonder if the proposed measure is powerful enough to be useful. We will forcefully assuage such doubts in the experimental section of this work. In the meantime we content ourselves with a simple demonstration. Color printing of images became possible only in the 19th century. Before that, color images were produced by using copper engravings to print in black ink, and these images were then hand-colored individually, in a process called aquatinting. We obtained two versions of a classic work on marine life by Louis Renard (b. ca. 1678.) [22]. The two versions were published 35 years apart, and almost certainly hand-colored by two different artists. Nevertheless, as we can see by the clustering of a subset of data from both texts in Figure 5, in each case a query image from one book does return the corresponding image from the other.

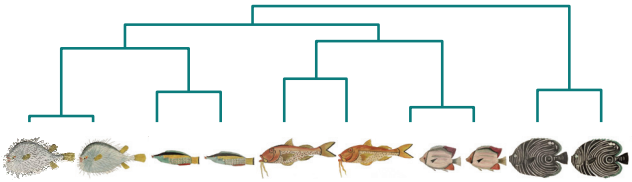


Figure 5. Five hand-colored images from the 1754 version of [22] and five hand-colored images from the 1789 version of the same text. An inspection of a higher resolution color version of this figure in [27] makes it clearer that these images are significantly different in both color and texture.

III. A Lower Bound to the Color Distance

We begin by describing the brute force algorithm to search for the nearest neighbor of a query color patch. This allows us to concretely define the search problem and motivates the need for a more efficient solution.

As shown in Table 1, the brute force algorithm compares the query with all color patches of the same size, and updates the *best-so-far* whenever a better match is found.

Table 1. The brute force algorithm to search the nearest neighbor of a user-given query Q in a book B

```
Algorithm [NN] = NN_BruteForce(Q,B)
1    best-so-far = INF;
2    h = height(Q); w = width(Q);
3    foreach page P in B
4       H = height(P); W = width(P);
5       for row ← 1:H-h+1
6          for col ← 1:W-w+1
7             CP = P(row:row+h-1,col:col+w-1,:);
8             if Dist(Q,CP) < best-so-far
9                best-so-far = Dist(Q,CP);
10               NN = CP;
11            endif
12         endfor
13      endfor
14   endforeach
```

As we can see in the nested for-loop at lines 5 and 6, we need $(H-h+1)\times(W-w+1)$ distance calculations for each page. If a book contains 500 pages with the size 2000×1500, and the size of a query color patch is 400×400 (all typical values), then the total number of color patches we need to check is more than ***881 million***. If each distance calculation takes one microsecond, this translates to about 15 minutes. Note that for some problems this may actually be acceptable. Given that it may take hours to scan a single historical manuscript of value, for some data mining/search queries we may be willing to wait overnight for a gem of information that has evaded scholars for centuries. However, such performance clearly precludes interactive real-time search, the number one requested feature from our domain experts. Furthermore, in many cases similarity search is not an end goal in itself, but a frequently-called subroutine in a higher level data mining algorithm. Thus, we must drastically improve the performance of the brute force search.

Attempts have been made to speed up the individual calculations of the *Histogram Intersection* value with various approximations [26], but the time complexity is still linear to the size of the dataset. However, even if the *Histogram Intersection* value could be *exactly* calculated one hundred times faster, it would still not allow for a real-time interactive search. What is need is a way to eliminate the vast majority of the calculations altogether. We can quickly see how this might be achieved with a simple example. Figure 6.*right* recalls the toy example shown in Figure 2. Consider the two queries shown in Figure 6.*left*; could they have perfect matches in Page-1? It is easy to see that we can answer this question without having to resort to the brute force search, just by examining their *global* histograms. Note that Query-1 has two red, six green, zero blue and one yellow pixels {**2**,**6**,**0**,**1**}. So in order for it to have an exact match, Page-1 must also have at least this number of pixels in its histogram, and it does, it has {**8**,**12**,**1**,**7**}.

Could Query-2 have a perfect match? The answer is clearly no. With a histogram {**0**,**4**,**5**,**0**} it needs at least 5 blue pixels to have a perfect match, but Page-1 can only provide it with 1.

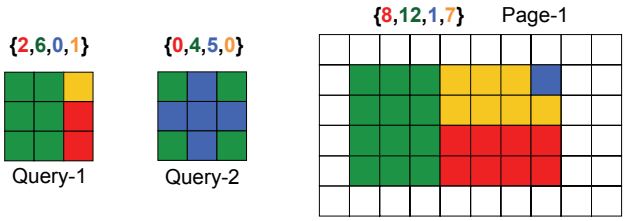


Figure 6. *left*) Two example queries with their color histograms. *right*) A sample page (cf. Figure 2) and its (global) histogram.

At the risk of redundancy we will further illustrate this idea with real world examples. In Figure 7.*left*, a query color patch featuring a shield with a golden deer on a blue background is shown, and in Figure 7.*right*, two candidate pages are shown. Let us first consider Page 1. Assume that we have already encountered a moderately similar object before searching this page, then the *best-so-far* value probably will not decrease after checking all color patches in Page 1. We can make this claim because a global view of the *entire* page reveals that it contains few blue pixels, the dominant color of the query. In other words, even if we perform a histogram intersection between Hist(Query) and Hist(Page 1), the blue bin(s) with a high value in Hist(Query) will have no match. If the *entire page* cannot supply enough pixels to match the query, then clearly no *color patch* inside Page 1 can supply enough pixels to match the query. This idea allows us to prune Page 1 with just a single calculation. As for Page 2, which contains a very similar color patch (only the direction of the deer is different) to the query in its bottom left corner, we cannot prune it after checking the color statistics for the whole page. However, *if* we had recursively divided the page into two halves (an idea we will shortly develop), the "pruning" idea could still be applied to the top half page, since it does not contain enough blue and golden pixels, and we could focus our search on the bottom half page, where a good match exists.

Figure 7. A query color patch and two pages from [4]. Note Page 2 contains a very similar color patch to the query in the bottom left corner.

Now we are ready to formally define the lower bound to the color distance in equation (2). If a color patch $CP(r,c,h,w)$ is *contained* in another color patch $CP'(r',c',h',w')$; in other words, $r \geq r'$, $c \geq c'$, $h \leq h'$, $w \leq w'$, then:

$$Dist(Q,CP') \leq Dist(Q,CP) \qquad (3)$$

Proof:

Since *CP* is *contained* in *CP'*, any pixel in *CP* also belongs to CP', thus we have:

$$\forall i \quad (Hist(CP')[i] \geq Hist(CP)[i]) \quad \Rightarrow$$
$$\forall i \quad (\min(Hist(Q)[i], Hist(CP')[i]) \geq \min(Hist(Q)[i], Hist(CP)[i])) \qquad (4)$$

With (1) and (4), we have:

$$HI(Hist(Q), Hist(CP')) \geq HI(Hist(Q), Hist(CP)) \qquad (5)$$

With (2) and (5), we can obtain (3). □

IV. Mining and Search Algorithms

In the last section we briefly hinted at the utility of the lower bound distance in searching through two sample pages; here we will explore more formally how we can incorporate the lower bound into similarity search and data mining.

As noted in Section I, we need to scale to massive datasets, so a disk-based indexing method is required. We can index each page by its color histogram (*PageHist*), which is a very compact representation of the original color image (When setting the number of bins $K = 6^3$, for a 24-bit color page of size 2000×1500, the space reduction is about 40000:1). We have shown in the last section that by just one comparison with the *PageHist*, an entire page can be pruned (i.e. Page 1 in Figure 7). In addition, as we shall show later, the *PageHist* can provide us with a very good searching/mining ordering by a simple heuristic strategy.

While the *PageHist* information can sometimes help us prune off entire pages, it does not contain any information to guide our search *within* a page (i.e. Page 2 in Figure 7), so we also need to obtain color histograms for lower level patches. To prevent the need for repeated transformations from the color space to histogram, we record a *BinMatrix* for each page (cf. Figure 2.*right*), which provides the bin_id of each pixel, as a preprocessing step. The color histogram of a specific patch in a page can then be easily obtained by a simple summation of its corresponding area in the *BinMatrix*. *BinMatrix* is stored on the disk, and retrieved when a search inside its corresponding page is initiated. As we will show in Section V, such retrieval is only necessary for a tiny portion of all pages and so the resulting I/O overhead is marginal.

A. Nearest Neighbor Search

In the example illustrated in Figure 7, we first compare the query to the whole page, and if the matching value is worse than the *best-so-far*, we can immediately prune the entire page; otherwise, we must continue by comparing the query to lower level patches of this page. Note that this idea lends itself to a classic *divide-and-conquer* framework.

As with all divide-and-conquer algorithms, we must be careful that data on the edge of the "division line" does not get missed. As Figure 8 hints at, we achieve this by having an appropriate amount of overlap between divided sections.

We first show the algorithm (in Table 2) which updates the current nearest neighbor in a color patch, given the height, width and histogram of the query, and *BinMatrix* of the patch. This algorithm will become a subroutine in our algorithm for searching an entire book (see Table 3 later in this section).

Table 2. Divide-and-conquer algorithm to update the current nearest neighbor of a given query in a color patch

```
Algorithm UpdateNN_DC(h,w,Hist_Q,BinMatrix)
1   H = height(BinMatrix); W = width(BinMatrix);
2   if  (h×w)/(H×W) ≥ StopRatio
3     NN_BruteForce(h, w, Hist_Q, BinMatrix);
4     return;
5   endif
6   RowOffsets = (H-h+1)/Sec;
7   ColOffsets = (W-w+1)/Sec;
8   for x = 1:Sec
9    StartCol = ColOffsets×(x-1)+1;
10   EndCol = (x<Sec)?(ColOffset×x+w-1):W;
11   for y = 1:Sec
12    StartRow = RowOffsets×(y-1)+1;
13    EndRow = (y<Sec)?(RowOffset×y+h-1):H;
14    subBinMatrix=BinMatrix(StartRow:EndRow,StartCol:EndCol);
15    Hist = GetHist(subBinMatrix);
16    if Dist(Hist_Q, Hist) < best-so-far
17       UpdateNN_DC(h, w, Hist_Q, subBinMatrix);
18    endif
19   endfor
20  endfor
```

In the above *divide-and-conquer* algorithm, lines 6-15 correspond to the "divide" phase: a patch is divided into *Sec*×*Sec* sub-patches, which are assigned 1/*Sec* of possible offsets in the row-wise and column-wise respectively (line 6 and 7). Lines 2-5 and line 16-18 correspond to the "conquer" phase: if the current patch is small enough, i.e. the area ratio of the query to the patch is above a certain value (line 2), then we simply perform a brute force search in it (line 3[3]); otherwise, we update the current *NN* in its sub-patches (line 17), to which the (lower bound) distance is smaller than *best-so-far* (line 16). In the example shown in Figure 8, a page is divided into 2×2 sub-patches.

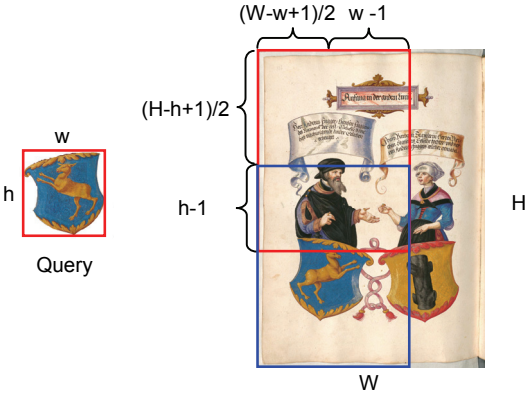


Figure 8. *left*) A query of size *h*×*w*. *right*) A page of size H×W, which is divided into 2×2 sub-patches (only two on the left are plotted). Note there is an overlap between them to include all offsets.

The only two parameters that need to be set are *StopRatio* and *Sec*. We can set *StopRatio* to any value ≤ 1. If *StopRatio* ≤ (*h*×*w*)/(*H*×*W*) (the area ratio of the query to the page), then no division would be performed and the algorithm degrades to the brute force algorithm; If *StopRatio* = 1, then we would have to divide the page until the size of the query, which is another undesirable "extreme point". In this case, when the patch gets very close to the query size, very few sub-patches can be pruned, and it is more efficient to do a brute force search. Although we know the *best StopRatio* has a value between (*h*×*w*)/(*H*×*W*) and 1, it is difficult to estimate the optimal value in advance. This is because the optimal value depends on both the data (the book) and the user's query.

However, finding a good (but not necessarily optimal) value for the *StopRatio* is easy. Notice that the choice of "valid" *StopRatio* is quite limited, because given the height and width of the page and query, and the value for *Sec*, the maximal depth of division is a fixed small number. In addition, as we will show in the next section, once the depth is above a certain number, the running time does not change much (it only increases slightly when approaching the maximal depth), and this holds for queries in various sizes and content on different datasets. Given that the value is not critical, we learn a good value by experimenting on a small set of queries, and simply hard code that value thereafter.

For the second parameter *Sec*, its value can be any integer in the range [2, min(*H*-*h*+1, *W*-*w*+1)]. A larger *Sec* reduces the size of sub-patches, and thus obtains a tighter lower bound distance. However, as noted in Figure 8, there are overlaps between sub-patches, and the area of overlaps increases with the value of *Sec*:

$$\begin{aligned} Area(Overlap) &= sum(Area(SubPatch)) - Area(Patch) \\ &= SubH \times SubW \times Sec^2 - H \times W \\ &= ((H-h+1)+(h-1)\times Sec)((W-w+1)+(w-1)\times Sec) \\ &\quad - H \times W \end{aligned} \tag{6}$$

This tells us that the cost of tighter lower bounds is to be forced to check more area and more sub-patches. If *Sec* = min(*H*-*h*+1, *W*-*w*+1), we can obtain the tightest lower bound, but we also generate the most sub-patches, and the algorithm would be similar to a brute force one (in which the size of the sub-patch equals that of the query, and so the lower bound distance is the exact distance). Another drawback of setting a larger *Sec* is that the algorithm has to maintain many sub-patches before we can prune them.

Our empirical testing suggests something that has been hinted at in other domains [6]. Once a lower bound is reasonably tight, making it tighter has little extra value, especially when searching large datasets. For example, the top left sub-patch in Figure 8 can be pruned even if *Sec* is set to just 2. In section V we will show in almost all cases, setting *Sec* to 2 can achieve excellent performance, and that the exact value of *Sec* is not critical to the performance.

We can now give the full algorithm to search for the nearest neighbor of a given color patch in a book in Table 3:

Table 3. Fast algorithm using the lower bound distance to search the nearest neighbor *NN* of a query *Q* in a book *B*

```
Algorithm [NN] = NN_LB(Q,B,PageHist,BinMatrix)
1   best-so-far = INF;
2   h = height(Q); w = width(Q); Hist_Q = Hist(Q);
3   foreach page B{i} in B
4     LB(i) = Dist(Hist_Q,PageHist(B{i}));
5   endforeach
6   Sort pages by I such that LB(I(i))≤LB(I(i+1));
7   foreach page B{I(i)} in B
8     if LB(I(i)) ≥ best-so-far
9       break;
10    endif
11    UpdateNN_DC(h,w,Hist_Q,BinMatrix{I(i)});
12  endforeach
```

[3] NN_BruteForce in Table 2 (search the query in a patch) is only slightly different from the one in Table 1 (search in a book), and is thus omitted for brevity.

Lines 3-5 first calculate the lower bound distance from the query to each page, then lines 7-12 search pages in the increasing order of their lower bound distances. Our experimental results in Section V will show that by using this simple heuristic strategy, the eventual nearest neighbor is always found in the first few pages and the algorithm can be stopped at any time after a very short startup time. Once we find a page whose lower bound distance is not smaller than *best-so-far* (line 8), the algorithm can be terminated (since we can guarantee that there is no better match remaining).

It is obvious that finding a smaller *best-so-far* early on helps to prune more aggressively. In addition to sorting all pages (line 6), we apply two more optimizations in each page search (omitted in Table 2 and Table 3 for brevity):

- We can quickly find a good "*first* best-*so-far*". We downsample the query and the first page, find the *NN* by the brute force algorithm in the downsampled space, and then project back the location of *NN* to the original page to obtain the "*first best-so-far*" value.
- We can sort sub-patches based on their distances to the query. Then we check most promising sub-patches first, and can terminate the search in a patch when the lower bound distance to the next sub-patch is not less than the *best-so-far*.

B. Motif Discovery

As we shall see in our experimental section, the query-by-content algorithm introduced in the previous section has already proven useful to several domain experts. However, true data mining comes with the discovery of previously *unknown* patterns. In this section we show that we can further extend our ideas to allow for the discovery of patterns which repeat *within* or *between* books. As this idea closely models the idea of *DNA motifs* and *time series motifs* [18], we call such patterns *color patch motifs*.

Definition 7: The *Color Patch Motif* is a pair of color patches {*CP*, *CP'*} of the given size *h*×*w* that is most similar among all possible pairs (excluding those residing in the same page) in a book *B*.

This definition generates an incredibly large search space. If we consider the example in Section III, then the brute force motif discovery algorithm has to compare each color patch in a page to all color patches in other pages. The number of comparisons required is about 7.75×10^{17}. Clearly we need to design an efficient algorithm. Before we design the algorithm for efficient motif discovery, it is helpful to first answer the following questions:

- *Do we really need to compare all these pairs?*

If we learn the experience we gained from the nearest neighbor search, the lower bound distance defined in formula (3) between a query and color patches can be trivially extended to our motif discovery scenario. For two patches *CP1* and *CP2*, which are contained in larger patches *CP1'* and *CP2'* respectively, we have:

$$Dist(CP1', CP2') \le Dist(CP1, CP2) \qquad (7)$$

This means that if the distance between two patches is larger than *best-so-far*, no better motif can be found between these two patches. Therefore, a similar divide-and-conquer algorithm as the one shown in Table 2 can be applied.

- *Is one motif enough?*

The definition above will only return the most similar pair of color patches, a single pattern. If we generalized it to instead find top *N* motifs, many *trivial matches* with a slight shift (say, 1 pixel away) from the best motif would be returned. If *N* is not large enough, then all top motifs may correspond to a single pattern. We can generate a more diverse top *N* motif set by answering the next question.

- *Do we need to return the exact location of motifs?*

One factor contributing to the diversity problem is that we have to return the *exact* location of the motifs with the *exact* user-specified size. If instead we returned two *slightly larger* patches, which contain a motif pair of the size specified by users, the end user would surely not care. As we shall see, this slight relaxation of the problem makes it much easier to produce an efficient algorithm.

Given this relaxed definition, trivial matches can be combined into these slightly larger patches (we call them *leaf patches* below). Astute readers may see that *leaf patch* pairs may also be trivial, but should note that the quantity of trivial matches decreases significantly (e.g.: even for a leaf patch only 9 pixels larger in height and width, it contains 100 patches of the motif size; and two such leaf patches can contain up to 10,000 trivial matches). Furthermore, as we will show in the next section (Figure 13, etc), by simply plotting *all* leaf patches, or just showing *one* leaf patch pair, the patterns are still quite obvious to the human eye.

Having made these observations, we are now in a position to formalize the algorithm to update the current best motif between two patches, given the (user-specified) height and width of the motif, *BinMatrix* of two patches and the distance between them. Table 4 makes this concrete:

Table 4. Algorithm to update the current best motif of size *h*×*w* between two patches (*BinMatrix1* and *BinMatrix2*) with distance *d*.

```
Algorithm UpdateMotif(h,w,BinMatrix1,BinMatrix2,d)
1   H = height(BinMatrix1); W = width(BinMatrix1);
2   if (h×w)/(H×W) ≥ StopRatio
3     Hist1 = BinMatrix1(1:h,1:w);
4     Hist2 = BinMatrix2(1:h,1:w);
5     if Dist(Hist1, Hist2) < best-so-far
6        best-so-far = Dist(Hist1,Hist2);
7     endif
8     Save BinMatrix1, BinMatrix2, d;
9     return;
10  endif
11  Divide both patches into Sec×Sec sub-patches;
12  foreach sub-patch p in 1st patch
13   foreach sub-patch q in 2nd patch
14    if Dist(p,q) ≤ best-so-far
15    UpdateMotif(h,w,BinMatrix_p,BinMatrix_q,Dist(p,q));
16    endif
17   endforeach
18  endforeach
```

When a pair of patches is small enough (line 2)[4], we do not issue a brute force search as we did for the nearest neighbor search in Table 2; instead, we only calculate the distance for one pair of sub-patches (which can be the one in

[4] Here we assume two patches are in a similar size for simplicity. The algorithm can be easily extended to handle the case where their size differs a lot [27].

the top left corner, as shown in lines 3-4) contained in the leaf patch pair, and use this value to update *best-so-far* (lines 5-7). Each leaf patch pair along with their distance are also recorded for further reference (line 8), since we have not checked all but one sub-patch pair of the motif size.

If two patches are not small enough, we divide them into sub-patches as in Table 2, and calculate all pair-wise distances. A further check between sub-patches is required only when the lower bound distance is not larger than *best-so-far* (lines 14-16), based on formula (7).

By doing this, we save time by eliminating brute force searches between leaf patches, while still guaranteeing that there is no false dismissal: the top one motif *must* exist in one recorded leaf patch pair. The proof is straightforward:

Proof:

We denote the top one motif pair as *M1* and *M2*, and two leaf patches containing them as *LP1* and *LP2*.

With formula (7), we have:

$$Dist(LP1, LP2) \le Dist(M1, M2) \qquad (8)$$

With the definition of the motif , we have:

$$Dist(M1, M2) \le best - so - far \qquad (9)$$

With (8) and (9), we have:

$$Dist(LP1, LP2) \le best - so - far$$

Which means, *LP1* and *LP2* can always pass the test in line 14 of Table 4, and thus will be recorded. □

One thing worth mentioning in the above algorithm is that the evaluation of *StopRatio* is quite different from the one in Table 2. Here we do not only care about the *speed* but more about the *quality* of retrieved results. *StopRatio* has to be close to 1, or the lower bound distance of leaf patch pairs would be too loose and we would not find a similar color patch pair of the user specified size. Given that the "valid" *StopRatio* is quite limited and its value is discrete, it is not hard to pick one. However, we should also keep in mind that the judgment for a good *StopRatio* is subjective and thus we make it adjustable. For example, the user can increase *StopRatio* if he/she wants to find more similar patterns.

Finally, the framework of the complete algorithm to find motifs from a book is very similar to the one in Table 3. We first calculate lower bound distances between all pairs of pages, and sort page pairs in the increasing order of their distances, and then for each pair call *UpdateMotif* in Table 4. The search can be terminated if the distance between two pages is larger than *best-so-far*. The last step is to scan all saved leaf patch pairs, removing those with larger distances than the eventual *best-so-far*.

V. EMPERICAL EVALUATION

We have designed all experiments such that they are easily reproducible. To this end, we have built a webpage [27] which contains all datasets and code used in this work, together with spreadsheets which contain the raw numbers displayed in all the figures. In addition, the webpage contains additional experiments which we could not fit into this work.

We divide our empirical evaluation into two sections: The first section contains informal case studies to demonstrate the utility of our system, and give the reader an intuition as to the kinds of scenarios in which our ideas can be used. In the second section we evaluate our ideas using the classic data mining metrics of sensitivity to parameters, speed, etc.

A. Case Studies

1) Query by Content

An historian at UC-Riverside wishes to find out how long humans have known that fish in the genus *Scorpaena* are poisonous. Fortunately, illustrated books on fish have been popular since the 16th century and hundreds of such books are now online. We indexed one such text from 1834, since its title "*A Selection of the Most Remarkable and Interesting of the Fishes found on the Coast of Ceylon*" refers to the region the historian is interested in. A Google image search found an illustration of one member of the genus, *Scorpaena scrofa*, and, as shown in Figure 9, we used this image as a query to our book.

The best match to our query shows a member of the genus, *Scorpaena mile*, and the accompanying text does not mention that the fish is poisonous.

In this case we were fortunate enough to find an illustration of a member of *Scorpaena.* Suppose we were not so fortunate, could we obtain a good match using a *real* image of a fish?

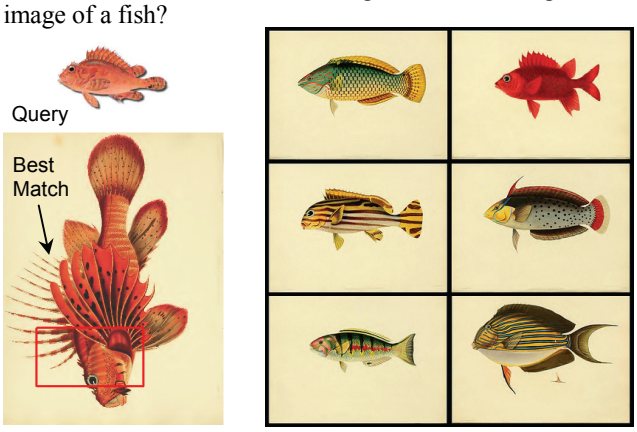


Figure 9. *left*) A query image of an illustration of *Scorpaena scrofa* and its best match in the text [1]. *right*) For context, six random other fish from the same text.

To test this possibility, we queried the database with the first three images retrieved in a Google image search for "*Scorpaena*". Figure 10 shows the results.

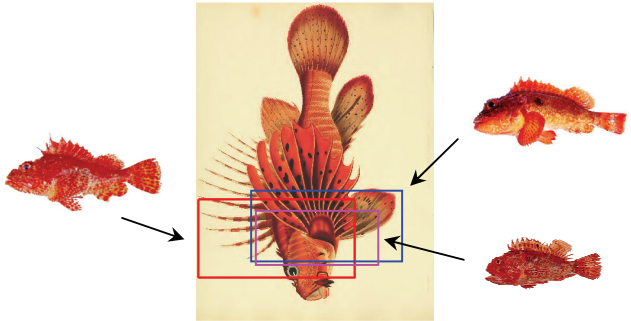


Figure 10. Three images of *real* fish from the genus *Scorpaena*. Each of them was used as a query to the text [1], and each of them found the same.

In the above four scenarios, the average time taken to find the *exact* nearest neighbor was 16.5 seconds (ranging from 5.6 to 29.0 seconds); however, the time to report the page on which the best match occurs took less than 0.1 second, allowing a truly interactive search. Note that in every case our query window size was smaller than the correct matching fish, and in the wrong orientation; however, it is clear in these experiments these are not critical sensitivities.

While we envision our work primarily for finding patterns within and between historical texts (cf. Section I), the results shown in Figure 10 emboldened us to consider one example where there is great utility in querying historical texts with modern images.

Historians and genealogists often need to search manuscripts to trace lineages. In addition to using text, they often use coats of arms as clues. Coats of arms are extremely common in Western manuscripts beginning in the fourteenth century (similar emblems exist in other cultures, for example, the Japanese *Mon*). In some cases a single text may have over 1,000 examples of different coats of arms.

In this case, finding query images is easy, since there is a huge *wiki-like* community of amateur historians that have created clean idealized versions of coats of arms; Figure 11.*left* shows an example. Some individual enthusiasts have collected or created more than 50,000 examples [10].

Figure 11. *left*) A modern idealized version of the coat of arms of Fugger von Babenhausen is used as a query. *center*) The best match to query in the 1545 text, *Das Ehrenbuch der Fugger*. *right*) A zoom-in of the best match.

As we can see in Figure 11.*left* the modern images are idealized, "clean" and created with a small color palette. Could such images be correctly matched to real hand-colored instances from 500-year-old texts? To test this we queried a text *Das Ehrenbuch der Fugger* (*The secret book of honor of the Fugger*, referred as *Fuggers* book for short below) [4]. This manuscript impressively illustrates the genealogical self-esteem of the Fugger family, a famous dynasty of wealthy merchants. The text contains portraits of 138 members of the family, including the matriarch, the *Mother Fugger*, together with their coats of arms. Figure 11.*center* shows the best match to the query, and Figure 11.*right* shows an enlarged detail.

While the nearest neighbor *is* similar, with some editing we can make it *more* similar. Figure 12 illustrates this.

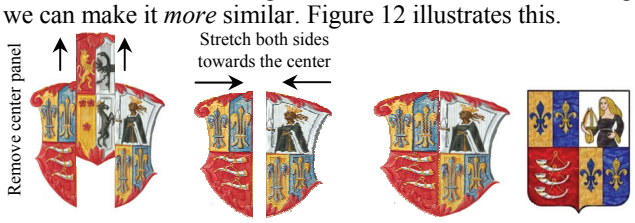


Figure 12. *left*) We can edit the coat of arms discovered in our search by removing the center panel and stretching the two remaining sections to meet in the middle. *right*) The resulting image looks more like the query.

We did not do the additional step of reversing the direction of the three horns in the bottom left quarter, since the handedness of objects is (typically) irrelevant in heraldry. This result suggests that the coat of arms we found was the original Fugger von Babenhausen coat of arms, augmented (the technical term is *Quartered*) by the insertion of a new panel. This query suggests that our distance measure is quite robust to "distortions" in the image, especially to the inevitable differences in the color palettes used by modern historians and the color palettes available to medieval artists.

Before moving on, it would be remiss of us not to comment on an invariance that our measure achieved. Both the original and query arms show a woman holding a mitre ("*Pope's hat*"). In the query image the woman in clearly Caucasian and blond. However, in the discovered coat of arms the woman appears to African. This is not the result of discoloration of the original color, or an error by the artist. From external sources we discovered that the black figure is the women mentioned in the Biblical book Song of Songs 1:5 "*I am black but comely, O daughters of Jerusalem, ...*". The explanation as to why the query image features a white woman is the obvious and disquieting one. According to Ralf Hartemink, a noted expert on heraldry, "*In the 19th and early 20th century many black people were made white, as racism became more common...*" [11].

2) *Motif Discovery*

We first tested our motif discovery algorithm on the 526 page *Fuggers* book which we investigated in the last section. Due to space limitations, we show two typical examples in Figure 13, and archive the complete results in [27].

Figure 13. *left*) A pair of pages which contains 27 leaf patch pairs, with 6 leaf patches on left page and 7 on the right page displayed. *right*) another pair contains 170 leaf patch pairs, with 16 and 18 leaf patches on each page.

In the above results, we plotted out *all* leaf patches (the highly overlapped rectangles), without specifying which two are in a pair. However, as we can see, this does not affect the utility of the patterns. So instead of allowing the algorithm to test all 170 combinations (in Figure 13.*right*), we can have the algorithm save time by reporting just *one* pair, which may not be optimal, but will surely satisfy a user.

Our work on datasets made us realize a limitation of motif discovery. The domain expert in this field wanted to ask questions such as "Are there two or more examples of a shield that is *not* a *Fuggers* shield, indicating that two or more members of the same family had married into the *Fuggers* dynasty?". This translates to "Are there motifs that do not feature pale blue and gold?". To answer such queries we need to do *supervised motif discovery*. These essentially reduce to the constraint of *must-include* or *must-exclude* these colors. Such constraints are trivial to include in our

algorithm. For brevity, we defer details to [27] and content ourselves with a single example shown in Figure 14.

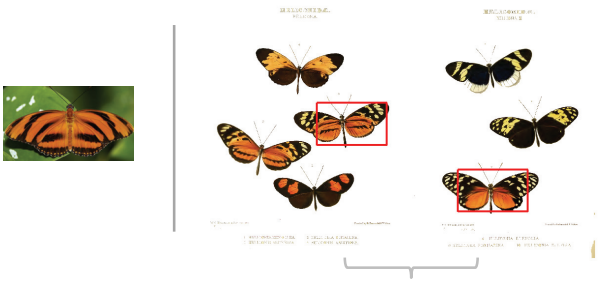


Figure 14. *left*) A photo of an orange/black butterfly used as a *must-include* color constraint on motif discovery. *right*) A pair of tiger butterflies found by the *supervised motif discovery* algorithm in [13].

3) Link Analyses

We can trivially extend our work on finding motifs *within* a book, to finding motifs *between* two books. This allows us to add hyperlinks between the two texts that researchers can follow as they investigate a topic [2]. This is a more challenging problem, since even two logically identical items from different texts may vary much more in the size, color palette used, style, etc. than those from the same manuscript.

We performed link analyses on two old books about butterflies. One of them contains illustrations of about 1250 species [9]. The other is a volume of "new" butterflies (referred to as "Exotic Butterflies" below) [13]. As the latter text is more heavily annotated, users who have questions about the butterfly in the former book can find helpful references in the latter one if there is a link created. It took about one hour to "join" these two books, and in Figure 15 we show two representative links found.

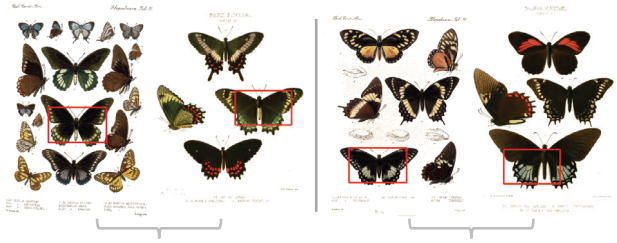


Figure 15. Two links found between two butterfly books. Note that the technique seems robust to the text variability (color palette, size, etc.). According to an entomologist, the left match is correct at the *genus* level and the right match is correct at the *species* level.

These results are very assuring and alleviate our worries about the variability that exists between different books. Figure 16 shows some additional examples.

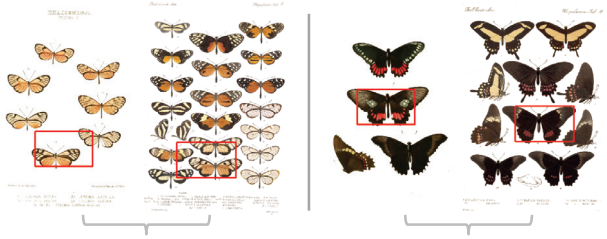


Figure 16. Two examples of links found by a motif join. Both matches are correct at the *species* level.

B. Data Mining Metrics

1) Sensitivity to Parameters

Previously we discussed parameter settings *theoretically*. Here we intend to show, *empirically*, that all parameters used by our algorithms are relatively insensitive and that it is easy to learn settings which generate effective and efficient results.

We first investigated the relation between *Sec* and the execution time of our *NN* algorithm. Since for a given query and a fixed *Sec*, the execution time varies when using different *StopRatio*, we then tested all possibilities (recall that the number of "valid" *StopRatio* is limited) to find the best running time for different *Sec*. Figure 17.*left* shows the results of 5 randomly chosen queries (distorted and only taking 64% of their original area) in the *Fuggers* book.

We can clearly see that: (1) The best execution time of large *Sec* (20,30 and 40) is much worse than small *Sec*; (2) The best execution time of small *Sec* does not vary that much, but still slightly increases with *Sec*. 3 out of 5 queries ran the fastest when *Sec* = 2, while only 0.38 and 0.05 second slower than the best (where *Sec* = 3) in the other two cases.

Next we checked the effect of *StopRatio* (i.e. the depth to stop the division) on the execution time when *Sec* = 2. The result for the same 5 queries is shown in Figure 17.*right*.

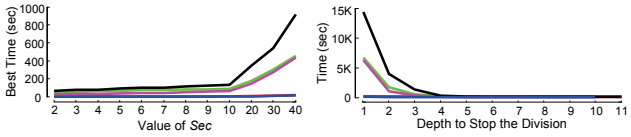


Figure 17. *left*) "Best execution time to find *NN*" vs. "*Sec*" for 5 randomly chosen queries from the *Fuggers* book. Note only 20,30,40 are tested for Sec >10. *right*) "Execution time to find NN" vs. "*StopDepth*" for the same 5 queries when *Sec* = 2. Note *StopDepth* = 1 corresponds to the brute-force search (but still with the heuristic page order).

This plot suggests that as long as we do not set the *StopDepth* too small, the execution time can be decreased to a similarly low level by our algorithm. Hence, we set the *Sec* to 2 and *StopDepth* to 8 for all experiments[5].

The only really important parameter for motif discovery is the motif size. It can be based on the domain knowledge and users' interests. However, for complicated datasets (e.g.: butterflies), it is difficult to find such a "perfect" value. In spite of this, however, it is clear that our algorithm can find some motifs of different sizes (e.g.: Figure 15.*right*), and is efficient enough to search of a range of sizes.

2) Speed and Anytime Property

We randomly picked 10 new queries from the *Fuggers* book, and reran the *NN* search with learned parameters. The average running time was 14.1 seconds (ranging from 1.0 to 39.0 sec). However, for 4 out of 10 queries, the best match appeared on the first page returned, and it took less than 0.1 second. Even in the worst case, the best match was on the 6th we retrieved. In comparison, a search using the brute force algorithm took 14 hours. We archive the *NN* search result for "Exotic Butterflies" in [27]. Its performance is even better:

[5] We did the same experiments (as in Figure 17) on the "Exotic Butterflies" book, and found the similar results, which we report at [27].

the best match was located on the first page for 8 out of 10 queries.

While motif discovery may need to run overnight, the brute force algorithm would take years. Moreover, with the heuristic search order, we can always stop at any time after a very short startup time while still obtaining good results (see Figure 18), essentially making it an *anytime algorithm*.

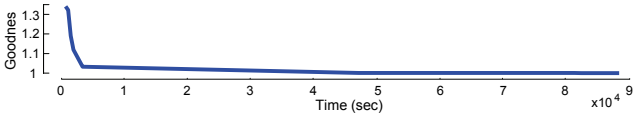


Figure 18. "Goodness of BSF" vs. "Time" for motif discovery in the *Fuggers* book. The "goodness" is measured as "eventual BSF/current BSF".

VI. CONCLUSIONS

We have introduced a novel lower bound for color matching, and two data mining algorithms that exploit it. We have shown our ideas allow true *data mining*, rather than just *query-by-content* for the domain of historical manuscripts. We have made all code and data freely available in order to bootstrap additional research in this area.

REFERENCES

[1] Bennett, J. 1834. A selection from the most remarkable and interesting of the fishes found on the coast of Ceylon. London, E. Bull.

[2] Bourgeois, Le. F., and Kaileh, H. 2004. Automatic metadata retrieval from ancient manuscripts. Document Analysis Systems: 75-89.

[3] D'Orbigny, C. 1849. Dictionnaire Universel d'Histoire Naturelle. Renard & Martinet, Paris.

[4] *Das Ehrenbuch der Fugger* (The secret book of honour of the Fugger) -BSB Cgm 9460, Augsburg, ca. 1545 - 1548 mit Nachträgen aus späterer Zeit.

[5] Das, M., Riseman, E.M. and Draper, B.A. 1997. FOCUS: Searching for Multi-Colored Objects in a Diverse Image Database. CVPR97 (756-761).

[6] Ding, H., et al. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. PVLDB 1(2): 1542-52.

[7] Faloutsos, C. et al. 1994. Efficient and Effective Querying by Image Content. Journal of Intelligent Information Systems, 3:231-262.

[8] Garain, U., Paquet, T. and Heutte, L. 2006. On foreground-background separation in low quality document images. International Journal of Document Analysis 8(1): 47–63.

[9] Godman, Frederick D. et al. 1879-1901. Insecta. Lepidoptera-Rhopalocera. Vol. III (Plates).

[10] Hartemink,R. 2010. Heraldry of the World. *http://www.ngw.nl/int/dld/o/oberkirb.htm*

[11] Hartemink, R. 2010. (Personal communication) April 30th 2010.

[12] Herwig, M. (2007). Google's Total Library: Putting the World's Books on the Web.

[13] Hewitson, William C. 1856. Illustrations of new species of exotic butterflies: selected chiefly from the collections of W. Wilson Saunders and William C. Hewitson.Vol I.

[14] Holley, R. 2009. Many Hands Make Light Work: Public Collaborative OCR Text Correction in Australian Historic Newspapers National Library of Australia. ISBN 978-0-642-27694-0

[15] Ioka,M. 1989. A method of defining the similarity of images on the basis of color information, technical report RT-0030, IBM Research.

[16] Kelly, K. 2006. Scan This Book! N.Y. TIMES, May 14, § 6 (Magazine), at 42.

[17] Like.com. Http://www.like.com/

[18] Lin, J., Keogh, E., Lonardi, S. and Patel, P. 2002. Finding motifs in time series. Proc. of 2nd Workshop on Temporal Data Mining.

[19] Liu, Y., Zhang, D., Lu, G. and Ma, W.-Y. 2007. A survey of content-based image retrieval with high-level semantics. Pattern Recognition, vol. 40, no. 1, pp. 262–282.

[20] Matas, J., Koubaroulis, D. and Kittler, J. 2000. Colour image retrieval and object recognition using the multimodal neighbourhood signature. In: Proc. of the ECCV. (2000) 48–64.

[21] Pass, G. and Zabih, R. 1996. Histogram refinement for contentbased image retrieval. IEEE Workshop on Applications of Computer Vision, pages 96–102.

[22] Renard, L., L. Poissons Ecrevisses et Crabes, de diverses couleurs et figures extraordinaires, que l'on trouve autour des Isles Moluques et sur les côtes des Terres Australes. Amsterdam.

[23] Rui, Y., Huang, T., and Chang, S. 1999. Image retrieval: current techniques, promising directions and open issues. Journal of Visual Communication and Image Representation 10, 39–62.

[24] Smeulders, A.W.M., Worring, M., Santini, S., Gupta, A. and Jain, R. 2000. Content-based image retrieval at the end of the early years. IEEE Trans. PAMI, vol. 22, no. 12, pp. 1349-1380.

[25] Smith, J. R. and Chang, S-F. VisualSEEk: a fully Automated Content-Based Retrieval System. ACM Multimedia. 1996, pp. 87-98.

[26] Swain, M. J. and Ballard, D. H. 1991. Color indexing. International Journal of Computer Vision, 7(1):11-32.

[27] Supporting webpage. http://www.cs.ucr.edu/~qzhu/

[28] The naturalist's library. Conducted by Sir William Jardine. Entomology.

[29] Zhu, Q. and Keogh, E. 2010. Using CAPTCHAs to Index Cultural Artifacts. IDA 245-257.

This research was funded by NSF grants 0803410 and 0808770.