An Image Matching Approach based on String Matching using Remainder-Prime Method

Ravendra Singh
RKDFIST, Bhopal

Jasvinder Pal Singh
RKDFIST, Bhopal

H.N.Verma
GLA University, Mathura

ABSTRACT

In this paper, we have proposed a method for document image search based on the code-vectors. The code vector representation of sub images enable us to use image matching techniques that make processing faster. The document images are converted to digital form using image scanner and are exhibited for public in image format, like jpg, jpeg, png etc. The adopted method uses a sliding window approach, in this approach each sub image clipped by narrow window is converted to code vector and these code vectors are used in image matching. Our method is based on modified Rabin Karp method that can reduce the computation cost.

Keywords: - Image matching; Image segmentation; Vector to code generation; Rabin Karp method.

1. INTRODUCTION

This paper presents a method, which is fast and accurate for document image matching. In this proposed method code-vectors are used in image searching. The code vector played a vital role in document image matching. It is an image matching technique based on codes. In the recent development of the digital technology, images can be converted into digital form using an image scanner and can be store in form of digital library for public use[1].

The approach, which is adopted for document images, is the sliding window approach. In this approach, each sub image clipped by the narrow window are converted into a two dimensional descriptor, which intern converted to a code vector using a suitable prime number and modulo division. These multi dimensional descriptors are also called feature vectors[2].

In this paper, we have proposed a novel method for converting multi dimensional descriptor into encoded representation. A set of integers has retrieved that can be used as the code for the purpose of sub- image detection and hence it is called code vector of the sub image.

These code vector representations of sub image enable us to use image matching techniques that make processing faster than original vector based method[1]. Our code vectors consist of four values, which are based on a selected prime number. The primality testing is the problem of finding large primes[10].

The density of prime numbers is feasible to test random integers of the appropriate size until we find a prime. The prime distribution function $\pi(n)$ specifies the number of prime that are less than or equal to n. The prime number theorem gives useful approximation to π (n)[10].

2. RELATED WORK

Mostly documents image search[3] includes use of OCR(Optical Character Recognition) [4]. He and Downton[5] used OCR system for digital libraries of a museum. Drira et al.[6] worked on the improvement of the accuracy of OCR. Kluzner et al.[4] gives another OCR method for old books in old fonts using different techniques.

On the other hand there are some other techniques which do not use OCR for character recognition. One such method is LSPC (Locality Sensitive Pseudo Code)[2], which is a kind of vector that contains discrete values. The LSPC discretize vectors into a list of integers with less loss of information compared with usual vector quantization[7]. LSPC is a technology where vectors can be converted into pseudo-code expressions without loss of their information. LSPC is a list of integers with its length smaller than the original vector[1].

Yet another method is the Locality Sensitive Hashing (LSH)[8] method, which do not uses OCR for character recognition and is used for indices. It is a famous probabilistic approximate nearest neighbor method. The nearest neighbor problem is defined as a collection of n points, build a data structure which, given any query point, reports the data points that is closest to the query. These data points are lived in a d-dimensional Euclidean space[9].

We also avoid character recognition. Instead the encoded feature vectors are used as a code without character recognition. The proposed approach uses code vectors that consist of four values obtained by applying modulo division using a prime number over each row of the entire small region[1].

3. PRIMALITY TESTING

A. Outline of Primality Testing

One of the important bases of the proposed method is primality testing[10-12]. In this section, we consider the problem of finding large primes[13], and we begin with a discussion of the density of primes[10]. It is feasible to test random integers of the appropriate size until a prime number is not found. The prime distribution function $\pi(n)$ specifies the number of primes that are less than or equal to n[10]. For example, $\pi(10) = 4$, since there are four prime numbers less than or equal to 10, namely, 2, 3, 5 and 7.

B. Composition of Primality Testing

This sub section describes the definition and factorization of the prime numbers.

Definition: (Prime Number Theorem)

It states that –

$\pi(n)/(n/\ln n)=1$.

Where limit ($n\to\infty$).

The approximation n/ln n gives reasonably accurate estimate of $\pi(n)$ even for small n. for example, it is off by less than 60% at $n=10^9$ where $\pi(n)=50847534$[10], and (n/ln n)= 4825492[10].

In the remainder of this section, the problem of determining whether or not a large odd integer n is prime has considered. It is assumed that n has the prime factorization

$N = p_1^{e1}\ p_2^{\ e2} \ldots . p_r^{\ er}$.

Where $r \geq 1$, $p1_,\ p_2 \ldots p_r$ are the prime factors of n, and $e_1, e_2, e_3, \ldots . e_n$, are positive integers. The integer n is prime if and only if r=1 and e1=1.

One simple approach to the problem of testing for primality is trial division. Trial division works well only if n is very small.

4. PROPOSED METHOD

The proposed work has been partitioned into several distinct steps:

A) Image Segmentation

B) Vector Extraction

C) Code Generation

D) String Searching of the Code.

E) Image Matching

Among these steps, (A), (B) and (C) are the preprocessing steps and (D) and (E) are the searching and matching phases respectively.

In phase (D) vector string is searched into the vector code of the given image and in phase (E) search image vector is compared to the corresponding portion of the given image vector, if phase (D) has found successfully.

(A) Image Segmentation:

First the documented image is converted into a matrix and then segmentation is done. The segmented image was divided into small regions, which were used to form vectors, e. g. first segment contain a_{ij} for i=1, 2, 3, 4…m and j=1, 2, 3, 4….n where a_{ij} is the element of original image matrix.

(B) Vector Extraction:

Each segmented image was fed to vector extraction. The method to obtain vectors is as follows:

A small matrix (segmented image) is used to generate the vector. The matrix is recomputed by dividing all of its elements by a prime number(in our case, the prime number is 101) and then taking the remainder, e.g. a_{ij}=remainder of a_{ij}/p for i=1, 2, 3, 4…m, and j=1, 2, 3, 4…n. .And p is the prime number. Now each row of the small matrix will form one vector.

(C) Code Generation:

In this process, each vector is converted into its relevant code. To be more exactly, suppose the segmented image was divided into small regions, the code vector is generated corresponding to each row of the small region. This code vector consists of the four values corresponding to the four rows of the small region. These vector codes are obtained by taking the remainder of sum of the elements of each row vector of the small region. i.e. if $V=[v_1, v_2, v_3, v_{4 \ldots\ldots. Vn}]$ is a vector then $v_i = (\sum a_{ij})/p$ for i=1, 2, 3, 4…m and j=1, 2, 3, 4…n. Stores these code values into an array so that these code values can be further used in searching.

(D) String Matching of the Code:

The string matching[14] can be defined as: the text is an array T[1…n] of length n and the pattern is an array P[1…m] of length m(≤n). We further assume that the elements of P and T are the characters drawn from a finite alphabet Σ. For example, we may have ∑ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} or ∑ = {a, b, c, d, e, f} etc. The character arrays P and T are often called strings of characters. All string matching algorithm scan the text with the help of a window which is equal to the length of the pattern. The first process is to align the left end of the pattern with the text window and then compares the corresponding characters of the window and the pattern. This process is known as an attempt. After a whole match or a mismatch of the pattern, the text window is shifted in the forward direction until the window is positioned at the (n-m+1) position of the text. This approach is the naive brute-force algorithm. In brute force algorithm, window is shifted to the right by one character after an attempt. This is the most primitive method of sequential scanning, which check all position in the text T whether an occurrences of the pattern P starts there or not. This can be implemented in complexity O((n-m+1)m).

For a large character of text the brute-force algorithm is not efficient to perform this task. To solve this problem, there are several well known algorithms in the literature [14-16] so far. These algorithms have their own advantages and limitation based on the method they use to calculate the shift value (the number of characters the window should move forward). The algorithms vary in the order in which character comparisons are made and the distance by which the window is shifted on the text after each attempt. We have used modified Rabin-Karp[15-16] method for string matching. We implement it work on our code vectors. Each time the code vector of searching image is compared against the code vector of a piece of given image, such that the size of the piece of the given image is exactly the same as the size of the searching image. In case of a mismatch we select next piece of given image whose size is same as the size of the searching image. In case of a match the control is passes over to next, image matching, step.

(E) Image Matching:

When the code-vector of the searching image is matched with the code-vector of a piece of the given image, then the searching image matrix is compared to the corresponding portion of the given image matrix, and each pixel of the searching image is compared with each pixel of the image. If each pixel is matched, then the portion of the sub-image found, will contain the searching image.

This step compares the images only if previous step successfully passed.

5. SOPHISTICATED ALGORITHM

The algorithm makes use of elementary number-theoretic notations such as the equivalence of two numbers modulo a third number. In our algorithm each character is a decimal number, and compute values by modulo p(=101), where p is a prime number. The values of the image pixels lie between 0 - 255. There are 54 prime numbers between 0 and 255. And the mid prime number is 103(if lower median is selected). We selected 101 as the value of p, which is near to mid prime number. Since n modulo 101 gives a number between 0 to100, where n is any integer ϵ [0, 255].

6. RESULT ANALYSIS

The size of the Source image is m-by-n pixels and let the size of the searching image be x-by-y pixels. The Pre-Processing time and the Matching time of the proposed algorithm are calculated for the variable size of the search image and the size of the source image is fixed (size=15) in case of the source image. The spurious hits are also calculated for the proposed algorithm, to obtain the accuracy of the proposed algorithm. The spurious hits are fake code values of the search image that matches with the code values of the source image, but in actual the search image is not matched within the source image.

Our pre-processing phase includes finding the code vectors for source image and searching image. These processes have complexities Θ(mn) and Θ(xy) respectively. Since usually the size of source image is greater than the size of searching image i.e., mn>xy, collectively we can say the pre-processing phase has complexity Θ(mn).

The complexity of matching phase is O(mnxy), more exactly it is Θ(mn*O(4+xy)/4).

The experimental results of the proposed algorithm is shown with the help of the tables, the first table have the experimental entries for the pre-processing and matching time of the fixed size of source image and the variable size of the search image.

The two analysis graphs for the pre-processing and matching time of the proposed algorithm is also shown in which x-axis related to size of the search image and y-axis related to the preprocessing and matching time. The first graph shows the preprocessing time of the proposed algorithm and the second graph shows the matching time of the proposed algorithm.

Table1: Experimental result of the variable size search image of the proposed algorithm.

Size of the Source image	Size of the Search image	Pre-processing Time	Matching Time	No. of Spurious Hit
15	1	0.052099	1.546639	0
15	2	0.003034	0.086410	3
15	3	0.000256	0.000064	0
15	4	0.004205	0.439811	0
15	5	0.001782	0.052241	0
15	6	0.001093	0.043761	0
15	7	0.001229	0.000195	0
15	8	0.000926	0.000153	0
15	9	0.000773	0.000130	0
15	10	0.000551	0.000084	0
15	11	0.000447	0.000058	0
15	12	0.000220	0.000043	0
15	13	0.000214	0.000044	0
15	14	0.000159	0.000044	0
15	15	0.000093	0.448092	0

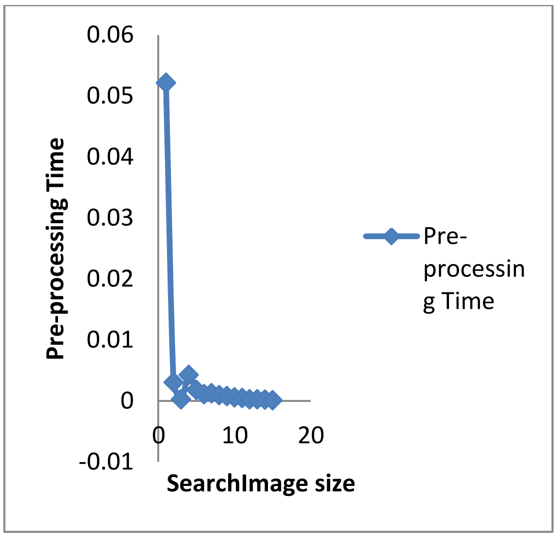

Figure 1: Analysis graph for the Pre-Processing time of the proposed algorithm.

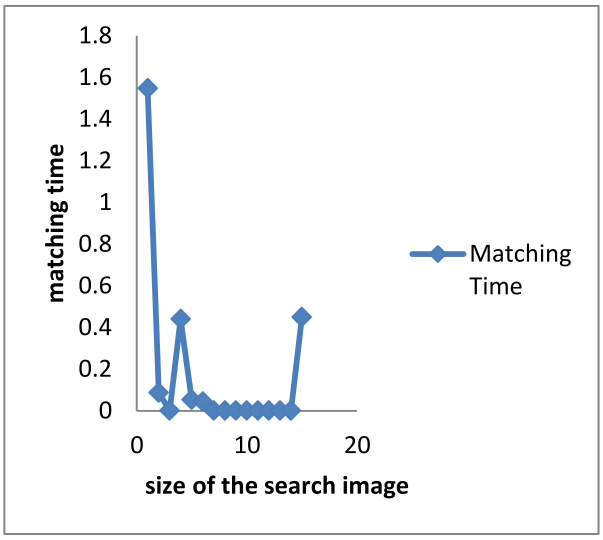

Figure 2: Analysis graph for the matching time of the proposed algorithm.

Experimental Results of the variable size of the source and search images of the Proposed Algorithm:

Here the more results are shown in the table, which contains the experimental results of the variable size of both the images i.e. source image and search image. The table contains two entries in each column except the column for the size of the source image, the first entry in the column of the size of the search image shows the minimum size of the search image and the second entry shows the maximum size of the search image;

The first entry in the column of the preprocessing time shows the minimum preprocessing time and the second entry shows the maximum preprocessing time;

The first entry in the column of the matching time shows the minimum matching time and the second entry shows the maximum matching time.

The first entry in the column of the spurious hits shows the number of fake hits in case of the minimum size of the search image and the second entry shows the number of fake hits in case of the maximum size of the search image.

Table2: Experimental Results of the variable size of the source and search images of the Proposed Algorithm.

Size of the Source image	Size of the Search image	Pre-processing Time	Matching Time	No. of Spurious Hit
15	1×1	0.039289	1.064649	0
	15×15	0.001762	0.066549	0
14	1×1	0.000794	0.000417	0
	14×14	0.002361	0.435763	0
13	1×1	0.001483	0.00517	0
	13×13	0.002279	0.439762	0
12	1×1	0.000725	0.022267	1
	12×12	0.003093	0.428895	0
11	1×1	0.000588	0.21817	0
	11×11	0.002647	0.423166	0
10	1×1	0.000498	0.021753	0
	10×10	0.002661	0.436696	0
9	1×1	0.000447	0.023392	0
	9×9	0.002804	0.434369	0
8	1×1	0.000335	0.023016	0
	8×8	0.002282	0.433000	0
7	1×1	0.000265	0.3336284	0
	7×7	0.002248	0.442265	0
6	1×1	0.000232	0.043597	0
	6×6	0.002678	0.429108	0
5	1×1	0.000163	0.041591	0
	5×5	0.002672	0.431304	0
4	1×1	0.000121	0.050273	0
	4×4	0.002249	0.432182	0
3	1×1	0.002477	0.431166	0
	3×3	0.002441	0.436649	0
2	1×1	0.002486	0.431546	0
1	1×1	0.002346	0.431144	0

Note: Size of the image is taken in a square image form.

The values in the column of the spurious hit are mostly zeros; it means fake code match is approximately near to zero. So the proposed algorithm is fast and accurate on the basis of the experimental shown in the above tables.

7. CONCLUSION

A fast image searching method for document images has proposed. The source and searching image are successfully read and stored. Segmentation is done successfully and each segment is also converted into code vectors. Modified Rabin-Karp string search method is applied successfully and finally image is compared. The proposed method increases the search speed with exact matching. The work also includes developing an efficient algorithm to realize exact matching for the document image in varying font sizes. With such an advanced algorithm, it would be possible to develop a fast algorithm, which is applicable to more difficult problems such as string matching of hand written documents.

8. REFERENCES

[1] Kengo Terasawa, Takahiro Shima and Toshio Kawashima, “A Fast Appearance-Based Full Text Search Method for Historical Newspaper Images,” ICDAR, 1520-5363, 2011.

[2] K. Terasawa and Y. Tanaka, “Locality Sensitive Pseudo-Code for Document Image,” Proc. ICRDAR2007, vol. 1, pp. 73-77, 2007.

[3] C. L. Tan, W. Huang, Z. Yu, Y. Xu, "Imaged Document Text Retrieval Without OCR", IEEE Trans. On PAMI, vol. 24, no. 6, pp. 838-844, 2002.

[4] V. Kluzner, A. Tzadok, Y. Shimony, E. Walach, A. Antonacopoulos, "Word Based Adaptive OCR for Historical Books", Proc. ICDAR2009, pp. 501-505, 2009.

[5] J. He and A. Downton, "Evaluation of a User-Assisted Archive Construction System for Online Natural History Archives", Proc. ICDAR2005, pp 442-446, 2005.

[6] F. Drira, F. LeBourgeois, H. Emptoz, "Document Images Restoration by a New Tensor Based Diffusion Process: Application to the Recognition of Old Printed Documents", Proc. ICDAR2009, pp 321-325, 2009.

[7] Messing, D. S, Van Beek. P, Errico. J. H., "The MPEG-7 Colour structure descriptor: Image description using colour at Local Spatial Information; International Conference on Image Processing", Thessaloniki, Greece, 2001, ISBN: 0-7803-6725-1.

[8] A. Gionis, P. Indyk, R. Motwani, "Similarity Search In High Dimension via Hashing", Proc. VLDB 1999, pp. 518-529, 1999.

[9] Alexandr Andoni, Piotr Indyk, "Near –Optimal Hashing Algorithm for Approximate Nearest Neighbor in High Dimensions", Proc. Symposium on Foundations of Computer Science, FOCS'06 pp.459-468, 2006.

[10] H. Cohen and H. W. Lenstra, Jr. "Primality Testing and Jacobi Sums", Mathematics of Computation, 42(165), pp 297-330, 1984.

[11] Gary L. Miller, "Riemann's Hypothesis and Tests for Primality", Journal of Computer and System Sciences, 13(3), pp 300-317, 1976.

[12] Michael O. Rabin, "Probabilistic Algorithm for Testing Primality", Journal of Number Theory, 12(1), pp 128-138, 1980.

[13] Leonard M. Adleman, Carl Pomerance and Robert S. Rumely, "On distinguishing prime numbers from composite numbers", Annals of Mathematics, 117, pp 173-206, 1983.

[14] H. N. Verma, Ravendra Singh, "A Fast String Matching Algorithm", International Journal of Computer Technology and Applications, Vol. 2(6), pp. 1877-1883, 2011.

[15] Richard M. Karp and Michael O. Rabin, "Efficient randomized pattern-matching algorithms", IBM Journal of Research and Development, 31(2), pp 249-260, 1987.

[16] Karp-Rabin, "An analysis of the Karp-Rabin String Matching Algorithm", 0020-0190/90/©1990-Elsevier Science Publisher B.V.(North-Holland).