グラフィクスとCAD 37−2
(1989. 2. 22)

ストリングマッチングを用いた高速化テンプレートマッチング方式

AN EFFICIENT ALGORITHM FOR TEMPLATE MATCHING BASED ON STRING MATCHING

朱 瑞豊　　安居院猛　　中嶋正之
Ruifeng ZHU　　Takeshi AGUI　　Masayuki NAKAJIMA

東京工業大学　工学部　像情報工学研究施設
Imaging Science and Engineering Laboratory, Tokyo Institute of Technology.

あらまし　パターン認識の研究において，大きな画像の中からテンプレートと呼ぶ特定のパターンと類似した図形を抽出する技術は必要不可欠である．既に，SSDA 法やテンプレートの部分情報を利用する方法などの多くの方法が提案されている．その中で高速にマッチングする方法として，ピラミッド構造を利用する方法が着目されているが，しかし，テンプレートが細かい特徴を持つ場合においてはピラミッド構造を利用してもテンプレートマッチングの演算時間を短縮することが困難であった．そこで，本報告では，二次元のテンプレート画像と入力画像を一次元のストリングに変換し，これらのストリングに対して有限オートマトンを用いて，入力画像の上で高速にテンプレートマッチングを行う手法を提案する．本手法は入力画像のサイズに対して線形的な演算時間を持ち，また，テンプレート画像が細かい特徴を持つ場合に有効である．

ABSTRACT. By reducing template matching to string matching in a natural way, we discribe a new approach for template matching. This approach having a high degree of insensitivity to noise is computationally linear to the size of original images. Computer experiments show that for the matching of images tested in this paper, the amount of computation is reduced on average by a factor of about 1200, compared with the "brute-force" method. However, the true significance of this approach is that template matching can be accomplished in cases which are difficult to use hierarchical matching techniques, and can be accomplished with a large computational saving.

1. INTRODUCTION

Template matching refers to the process of matching a template with a corresponding region of an image. Let T and P be a template and an image, respectively. The theme is to locate in image P the subimage which is most similar to the template T. To locate the most possible position in image P that matches with the template T, a large amount of computation often needs to be performed on the pixel pairs of the template T and the allowed search area in image P. We determine a match by selecting the position of maximum cross correlation between the template T and each possible position of image P. Assuming the sizes of the template T and image P are M1*M2 and N1*N2, respectively. For a particular position in image P, there are M1*M2 pixel pairs that should be compared. The number of such positions is (N1-M1+1)*(N2-M2+1), so the "brute-force" method requires O(M1*M2*(N1-M1+1)*(N2-M2+1)) computation time. Barnea and Silverman [1] presented a class of algorithms called SSDAs. SSDAs reduce the cost of determining a mismatch at a particular position by performing a sequential search which may be terminated before all M1*M2 pixel

pairs are tested. According to [1], a saving of computation time of two orders of magnitude is achieved by SSDAs. The other ways to reduce the cost of locating the best match position are planning [2],[3] and hierarchical matching techniques using pyramid data structures [4]. Additional discussions concerning pyramids data structures may be found in [5]. The hierarchical matching techniques may be logarithmic efficient and attract a great deal of attention. But for some kinds of images, it is impossible to get meaningful small images of reduced-resolution. The reason is that if the reduced-resolution images are too small, we will loose some important details of interest, especially, when the original image is severely corrupted by noises, which results in that it is very difficult to obtain an adequate threshold. If we choose a strict threshold, the object of interest might be lost. Otherwise, we may take a lot of locations that seems to be the location of the true object and search for them to the higher resolution levels. We have to backtrack at a certain level when we make sure that they are false. For such backtracking, the hierarchical matching techniques can not be implemented as efficient as expected.

In the present paper, a new approach for template matching is described. This approach, which has a high degree of insensitivity to noise, is computationally linear to the size of the original image. This is clearly optimal since both the image and the template have to be read and this takes **O**((N1*N2)+(M1*M2)) time. The algorithm needs **O**((N1*N2)+(M1*M2)) space, and also works on-line, assuming the image is processed row by row. Finding a linear approach to the template matching problem is a kind of challenge in the fields of image processing and pattern recognition. Computer experiments show that for the matching of images tested in this paper, the amount of computation in terms of the number of pixel pairs is reduced on the average by a factor of about 1200. However, the true significance of this approach is that template matching can be accomplished in cases which are difficult to use hierarchical matching techniques, and can be accomplished with a large computational saving.

In section 2, we present the definition of mismatch measure. Section 3 will describe the basic concepts of the proposed approach. In section 4, a complete description of the proposed algorithm and its implementation is presented. Section 5 discusses the experiments conducted on images taken from LANDSAT.

2. THE DEFINITION OF MISMATCH MEASURE

P is taken as an N1*N2 array of image with at most gray level K; i.e.,

$$0 \le P[i,j] \le K, \quad 1 \le i \le N1,\ 1 \le j \le N2 \tag{1}$$

T is an M1*M2 array of template having the same gray level range as P; i.e.,

$$0 \le T[i,j] \le K, \quad 1 \le i \le M1,\ 1 \le j \le M2 \tag{2}$$

Throughout this paper, we use a standard measure of mismatch between the template T and image P. The mismatch measure is defined as follows:

$$E_{i0,j0} = \sum_{i=1}^{M1} \sum_{j=1}^{M2} |T[i,j]-P[i0+i-1,j0+j-1]|$$

$$1 \le i0 \le N1-M1+1,\ 1 \le j0 \le N2-M2+1 \tag{3}$$

That is $E_{i0,j0}$ is the sum of the absolute errors between the template pixels and the image pixels with the upper left corner at the coordinate (i0,j0) in image P. The degree of

similarity at the given location is high when (3) is small. A threshold SRH is designed to obtain: (1) The first match where SRH is firstly exceeded. (2) The true match where the total mismatch is minimal (3) All matches where SRH is exceeded. Assume the location of the true match is at (i1,j1), the total mismatch at (i1,j1) is minimal, that is:

$$E_{i1,j1} \leq E_{i0,j0},$$
$$0 \leq i0 \leq N1-M1+1,\ 0 \leq j0 \leq N2-M2+1 \qquad (4)$$

Note that for this mismatch measure, the cost of applying the template T is directly proportional to the size of the template T.

3. THE BASIC CONCEPTS

The basic concept is to reduce the template matching problem to the string matching problem in a natural way. For this we define a kind of atomaton called string matching machine, which is abbreviated as SMM. Let PT and S be a keyword and an arbitrary string, respectively. The string matching problem is to find all occurrences of the keyword PT included in S. We first makes a SMM from the keyword PT, then apply S as input to the SMM. We discribe the behavior of the SMM by two functions: a goto function g and a failure function f. The SMM consists of a set of states, and each state is represented by a number. The SMM processes the string S by reading the characters in S, making state transitions and occasionally emitting output. The goto function g maps a pair consisting of a state and an input character into a state or the message fail. The failure function f maps a state into a state. The failure function is referred whenever the goto function reports fail. We also assign some states as output states which means that a keyword has been found. For details concerning such atomaton refer to [6],[7]. The behavior of the SMM is given in algorithm 1.

Algorithm 1.
Input. A string S[1..N2] and a SMM with goto function g and failure function f.
Output. locations at which PT occurs in S.
Method.

```
1.begin
2. read(s); state:=0;
3. for i:=1 until N2 do
4. begin
5.  while g(state,S[i])=fail do
      state:=f(state);
6.  state:=g(state,S[i]);
7.  if state=M2 then write('found at',i)
8. end
9.end.
```

The algorithm to construct the SMM from PT is summarized below:
Algorithm 2. Construction of the SMM.
Input. keyword PT[1..M2] and a parameter EE.
Output. A SMM with goto function g and failure function f.
Method. We assume g(s,a)=fail if 'a' is undefined or if g(s,a) has not yet been defined.

```
 1.begin
 2. read(PT); state:=0;
 3. for i:=1 to M2 do
 4. begin
 5.  for j:=(PT[i]-EE) to (PT[i]+EE) do
       g(state,j):=state;
 6.  state:=state+1
 7. end;
 8. for all a such that g(0,a)=fail do
      g(0,a):=0;
 9. for i:=1 to M2 do f[i]=i-1;
10.end.
```

The purpose of parameter EE is to accept those inputs which are different from PT[i] at most EE. In this way, the SMM can work on approximate matching instead of exact matching. So the images corrupted by noise can be matched by such SMM.

4. THE ALGORITHM

The general scheme of the algorithm is composed of two distinct steps, a translation step and a matching step. The purpose of the translation step is to translate the two-dimensional template T and image P to one-dimensional numbers of T1 and P1, respectively, while the matching step searches P1 for the occurrences of T1 by using algorithm 1 in which T1 is considered to be a keyword.

4.1. THE TRANSLATION STEP

The translation step sums each possible M1-pixel of the image P vertically, where M1 is the column length of the template , then computes the average of it and stores the average in P1. In the same way T1 is available from the template T. That is we translate the M1-pixel to a number by packing them together in a computer word, which we then treat as a number k in the following. This corresponds to writing the M1-pixel as a number in a radix-10 number system. The number k corresponding to the M1-pixel section of P[i0,j0], P[i0+1,j0] ... P[i0+M1-1,j0] in position(i0,j0) is given in (5).

$$k=((int)P[i0,j0]+(int)P[i0+1,J0]+\ldots +(int)P[i0+M1-1,j0])/M1, \qquad (5)$$

where **int** is used to change each pixel into an integer. But shifting one position down in the image P simply corresponds to replacing k by k-(P[i0,j0]-P[i0+M1,j0])/M1. This leads to a considerable saving in computation.

4.2. THE MATCHING STEP

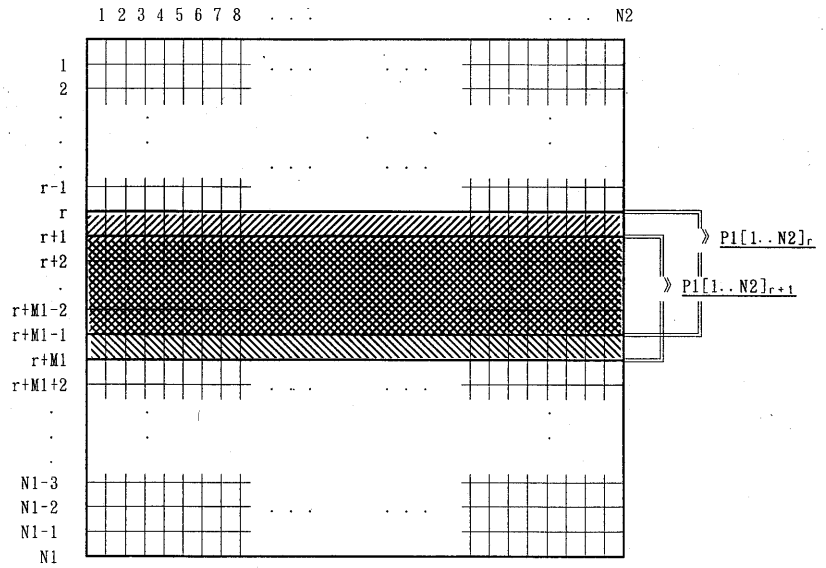
The matching step searches P1 for the occurrences of T1 using the SMM technique row by row. The algorithm is as follows:

ALGORITHM 3.
INPUT. The image and template are stored in the P[1..N1,1..N2] and T[1..M1,1..M2], respectively.
OUTPUT. Locations in image P at which the template matches with that region mostly.
METHOD.

```
 1. begin
 2.   read(P); Image-Trans;
 3.   read(T); Template-Trans;
 4.   r:=1; L=0;
 5.   repeat MATCHING(T1, P1,found,c);
 6.   if found=true then
 7.     begin
 8.       L:=L+1;
 9.       writeln('The upper-left corner
          of the template is found at',
          r,c-M2+1);
10.     end;
11.    change(r);
12.    r:=r+1;
13.    until (r>N1-M1+1);
14.    if L=0 then writeln('not found')
15.  end.

Procedure Template-Trans;
 1. begin
 2.   for i:=1 to M2 do
 3.   begin
 4.     TT:=0;
 5.     for j:=1 to M1 do
          TT:=TT+(int)T[j,i];
 6.     T1[i]:=TT/M1;
 7.   end
 8. end;
Procedure Image-Trans;
 1. begin
 2.   for i:=1 to N2 do
 3.   begin
 4.     TT:=0;
 5.     for j:=1 to M1 do
          TT:=TT+(int)P[j,i];
 6.     P1[i]:=TT/M1;
 7.   end
 8. end;
```

In the above algorithm we first compute two vectors of T1 and P1 from template T and image P, respectively.



$P1[1..N2]_{r+1}=P1[1..N2]_r-(P[r,1..N2]-P[r+M1,1..N2])/M1$

Fig. 1 The computation of P1 from r times to r+1 times.

Secondly, the MATCHING procedure is used to match T1 with P1. If matching process fails at row r, that is found=false, another procedure change(r) is used to change the value of P1 from r times to r+1 times. The change(r) procedure is obviously optimal since shifting one row down in image P simply corresponds to replacing P1[j] by: P1[j]-(P[r,j]-P[r+M1,j])/M1. Such operations are implemented repeatly, until r>N1-M1+1. Fig.1 illustrates the computation of P1 from r times to r+1 times. Procedure change(r) is as follows:

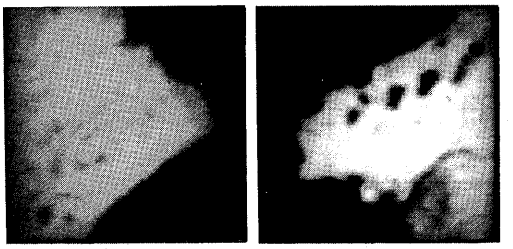
```
Procedure change(r);
1. begin
2.  for j:=1 to N2 do
    P1[j]:=P1[j]-((int)P[r,j]
          -(int)P[r+M1,j])/M1;
3. end;
```

The MATCHING procedure is the same as algorithm 1 except that a procedure Mismatch-measure(found,r,c) is inserted at the end of algorithm 1.

```
Procedure Mismatch-measure(found,r,c);
1.begin
2. found:=true; i:=1; j:=1; Error:=0;
3. repeat
4. repeat
5. Error:=Error+abs((int)T[i,j]
          -(int)P[r,c-M2+j]);
6.  if Error>SRH then found:=false;
7.  j:=j+1;
8. until (j>M2) or (found=false);
9. i:=i+1;
10. until (i>M1) or (found=false)
11. end;
```

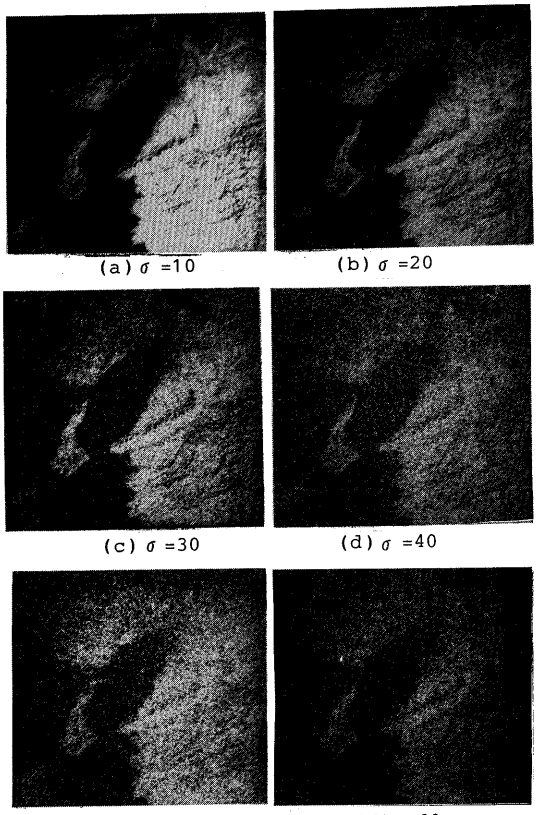
Algorithm 3 really only finds a position in image P from which the vertical averages are close to those of the template T, or they may be different from each other at most EE. EE is defined in algorithm 2. So to be sure, we should accumulate the errors between the image P and template T pixel by pixel, and compare it with a threshold SRH. When the accumulated error exceeds SRH, this location is judged as a failure match location, we call such

Fig.2 The original image taken over Osaka



(a) (b)

Fig.3 The templates extracted from the original image of Fig.1.
(a) Part of Awajishima island
(b) A region on the Osaka-Wakayama border.



(a) σ =10 (b) σ =20
(c) σ =30 (d) σ =40
(e) σ =50 (f) σ =60

Fig.4 Noise-corrupted images.

failure as "false alarm". The number of such false alarm is L in algorithm 3, so the computational cost for algorithm 3 is (N1-M1+1)*N2+L*M1*M2 in worst case. However, if we expect to match many different templates with the same image, we can extend P1 to a (N1-M1+1)*N2 array and compute P to obtain P1 beforehand. The cost of computing P can be shared among these costs, since P need be computed only once.

5. EXPERIMENTAL RESULTS

In order to verify the performance of the new approach described in this paper, a sequence of experiments were conducted on data taken from LANDSAT. Fig.2 shows the 480*512 original image P_0 taken over Osaka, Japan. The templates, which are part of Awajishima island and a region on the Osaka-Wakayama border, were extracted from the 64*64 subimage of the original image P_0 and used to search the noise corrupted images for the most similar 64*64 subimage. The templates are shown in Fig.3. Six more images were created by adding to the gray level of each pixel of P_0 a Gaussian noise with zero mean and standard deviation σ = 10, 20, 30, 40, 50, 60, respectively. Fig.4 shows these images when σ = 10, 20, 30, 40, 50, 60, respectively. The threshold SRH defined in section 4.2 was taken as M1*M2*0.7* σ. For notational convenience,

standard deviation σ, match location, accumulated errors and the number of false alarm are given as 4-tuples of the form (σ ,(r,c),E,L), Such a 4-tuples represents at row r and column c the template matched with a correlation value E, when the standard deviation of noise is σ . It should be noted that a considerable amount of locations judged as false alarm are within a circle range centered at the true match location. The experiments are made on VAX 11/750 system. The computational cost is evaluated in the following way. We assume that the pixel compare time, integer multiple time and integer add time are equal. To form a basis of comparison, the number of pixel pairs needed to be compared in the brute-force method is C1 as calculated in (6).

$$C1=2*(N1-M1+1)*(N2-M2+1)*M1*M2 = 1.5338*10^9 \text{ equivalent integer add} \quad (6)$$

where N1=480, N2=512, M1=64 and M2=64. As for the proposed approach, the costs are:

(1) Computing T1 is M1*M2+M2 equivalent integer add.
(2) Computing P1 in initialization is M1*N2+N2 equivalent integer add.
(3) In procedure change of Algorithm 3, the total cost for computing P1 is 3*(N1-M1+1)*N2 equivalent integer add.
(4) The number of comparison using SMM is (N1-M1+1)*N2 equivalent integer add.
(5) The cost for processing false alarm at most is L*M1*M2 equivalent integer add.

Thus, the total cost C is:

$$C=M1*M2+M2+M1*N2+N2+3*(N1-M1+1)*N2+(N1-M1+1)*N2+L*M1*M2 = M2+M1*M2+5*N2-3*M1*N2+4*N1*N2+L*M1*M2 = 8.9146*10^5+L*4096 \text{ equivalent integer add} \quad (7)$$

The computational efficiency η is:

$$\eta = C1/C \quad (8)$$

The entries of Table 1 show the 4-tuple, the total computational cost C and the computational efficiency η. For the matching of images tested in this paper, the amount of computation in terms of the number of pixel pairs is reduced on the average by a factor of about 1200. However, the true significance of this approach is that template matching can be accomplished in cases which are difficult to use hierarchical matching techniques, and can be accomplished with a large computational saving.

REFERENCES:

[1] D. I. Barnea and H. E. Silverman, A class of algorithms for fast digital image registration, IEEE Trans. Computer C-21, No.2, 1972, pp. 179-186.

[2] G. J. VanderBrug and A. Rosenfeld, Two-stage template matching. IEEE Trans. Computers C-26, No. 4, 1977, pp. 384-393.

[3] A. Rosenfeld and G. J. VanderBrug, Coarse-fine template matching. IEEE Trans. System Man Cybernet. SMC-7, No. 2, 1977, pp. 104-107.

[4] S. L. Tanimoto, Template matching in pyramids, Computer Graphics and Image Processing 16, 1981, pp. 356-369.

[5] S. L. Tanimoto and T. Pavlidis, A hierarchical data structure for picture processing, Computer Graphics and Image Processing 4, 1975, pp. 104-119.

[6] A. V. Aho, J. E. Hopcroft and J. D. Ullman, The design and Analysis of Computer Algorithm, Addison-wesley, Reading, MA, 1974.

[7] A. V. Aho and M. J. Corasick, Efficient string matching: An aid to bibliographic search, Comm. ACM 18(6), 1975, pp.333-340.

TABLE 1.

The computational cost and computational efficienay

Threshold SRH=0.7*64*64*σ

EE*	(σ , (row, column), E, L)	C=8.9146×10^5+L*4096	η =C1/C
6	(10, (110, 330), 20560, 89)	1.2560 × 10^6	1221
6	(20, (110, 330), 42290, 75)	1.1986 × 10^6	1280
8	(30, (110, 330), 63931, 121)	1.3871 × 10^6	1106
10	(40, (110, 330), 88164, 151)	1.5099 × 10^6	1015
10	(50, (110, 330), 108916, 40)	1.0553 × 10^6	1453
12	(60, (110, 330), 137749, 19)	0.9693 × 10^6	1582
			Ave:1276
4	(10, (130,250), 20152, 245)	1.8949 × 10^6	809
6	(20, (130,250), 43104, 112)	1.3502 × 10^6	1136
8	(30, (130,250), 63778, 414)	2.5872 × 10^6	613
10	(40, (130.250), 87824, 31)	1.0184 × 10^6	1505
10	(50, (127,250), 130505, 60)	1.1372 × 10^6	1357
12	(60, (132,248), 140578, 20)	0.9733 × 10^6	1580
			Ave:1167

* EE is defined in Algorithm 2. of section 3.1.