Finding connected components in binary images

1. Introduction

Connected components analysis in binary images is covered extensively in Davies (2017), Chapter 8, Section 8.3. The basic concept is fairly simple: take a binary image containing shape fragments, find which are connected to each other, and label them appropriately—typically numbering them in the order in which they are found in a forward raster scan through the image. However, as we shall see, the devil is often in the detail and more care is needed to be sure that connectedness algorithms run correctly than might a priori be expected for such simple images.

The algorithm presented below is a fully worked out version of the rather cursory one given in Table 8.2 of Davies (2017). We start with the input section: note that this takes an input binary image containing black objects on a white background and represents the objects as 1s on a background of 0s, as in Figure 1.

```matlab
str00='tangle.bmp'; % input image
str0='CCA'; % output text
str1=strcat(str0,'1.txt'); % ditto
str2=strcat(str0,'2.txt'); % ditto
Outtxt1=fopen(str1,'w');
Outtxt2=fopen(str2,'w');
frame=imread(str00);
grey=rgb2gray(frame);
[ly,ix]=size(grey);
t0=uint8(grey);
t1=uint8(zeros(iy,ix));
for y=1:iy
  for x=1:ix % represent black objects by 1s
    if t0(y,x)==0, t1(y,x)=1; else, t1(y,x)=0; end
  end
end
P0=0; P1=0; P2=0; P3=0; P4=0; P5=0; P6=0; P7=0; P8=0;
A0=0;
for y=1+1:iy-1 % print starting image as 1s on a blank background
  for x=1+1:ix-1
    A0=t1(y,x);
    if A0==0, fprintf(Outtxt1,'  '); % print image in ASCII format
      else, fprintf(Outtxt1,'%2d',A0);
    end
    fprintf(Outtxt1,'
');
  end
  fprintf(Outtxt1,'
');
end

The next section of code is the fully worked out version of the algorithm presented in Davies (2017), Table 8.2. It ends up with a 'cotable' showing which pairs of pixel labels are found to be adjacent and therefore connected. The important thing to note is that this algorithm works in a forward raster scan, so that when it hits a particular pixel P0, it utilises only the pixel values P2, P3, P4, P5 that have already been calculated to determine the value of P0: the values of pixels P6, P7, P8, P1 are
```
totally ignored (the arrangement of the 9 pixels in the 3 × 3 neighbourhood is shown near the top of the algorithm). If all four of the pixel values $P_2$–$P_5$ are zero, and $A_0=1$, the current pixel must be the starting point of a new object, and, after incrementing the object number $N$ by 1, the pixel is labelled as $N$. If in an object, but not the start of a new object, up to 8 entries are made in the cotable, and the new pixel value $P_0$ is given by the maximum of the four values $P_2$ to $P_5$. Next, before leaving the pixel, its value is recorded in the output image space $t_1$.

At this stage, each line of the image is printed out in ASCII notation, though for clarity each pixel value is presented as a single digit, using a hexadecimal type notation (1, 2, 3, ..., 9, A, B, D, ...), which is suited to representing values up to ~30. The result is shown in Figure 2.

Finally, note that the cotable is initialised to contain only NaN elements, and these are updated to numerical values as they become available: this is a useful facility of Matlab that didn't appear in many earlier languages, and will be used below to streamline the algorithm.

```matlab
N=uint8(0); % initialise to no objects yet found
maxfour=uint8(0);
cotable=zeros(20);
cotable(:,:,)=NaN; % use NaNs to represent empty elements
for y=1+1:iy-1
    for x=1+1:ix-1
        P4=t1(y-1,x-1); P3=t1(y-1,x); P2=t1(y-1,x+1);
        P5=t1(y,x-1); P0=t1(y,x); P1=t1(y,x+1);
        P6=t1(y+1,x-1); P7=t1(y+1,x); P8=t1(y+1,x+1);
        A0=t1(y,x);
        if A0==1
            if P2==0 && P3==0 && P4==0 && P5==0
                N=N+1;
                P0=N;
            else
                maxfour=0;
                if P2>maxfour, maxfour=P2; end
                if P3>maxfour, maxfour=P3; end
                if P4>maxfour, maxfour=P4; end
                if P5>maxfour, maxfour=P5; end
                P0=maxfour;
                if P2>0, cotable(P0,P2)=1; end
                if P3>0, cotable(P0,P3)=1; end
                if P4>0, cotable(P0,P4)=1; end
                if P5>0, cotable(P0,P5)=1; end
                t1(y,x)=P0;
            end
            delta=int8(P0-10);
            if P0==0, fprintf(Outtxt1,’ ‘); % print image in ASCII format
            elseif P0<10, fprintf(Outtxt1,'%d',P0);
            else, fprintf(Outtxt1,' %c',char(delta+'A'));
            end
            fprintf(Outtxt1,'
');
        end
    end
end

cotableN=cotable(1:N,1:N); % eliminates unused rows and columns
...
The cotable produced by the above code is readily represented by the following routine: its output is shown in Figure 3.

```matlab
for i=1:N % print cotable in ASCII format
    for j=1:N
        element=cotableN(i,j);
        if isnan(element)
            fprintf(Outtxt2,' -');
        else
            delta=int8(element-10);
            if element<10, fprintf(Outtxt2,'%2d',element);
            else, fprintf(Outtxt2,' %c',char(delta+'A'));
            end
        end
    end
    fprintf(Outtxt2,'
');
end
fprintf(Outtxt2,'
');
...
```
To proceed further, we need to re-present the ctable in an enhanced form, changing each 1 to the minimum of the indices of the row and column on which it is situated. At the same time, it is ensured that the ctable is fully populated, as in Figure 4. This is achieved with the next piece of code:

```
for i=1:N % change ctable entries to reflect row and column numbers
    for j=1:N
        if ~isnan(ctableN(i,j))
            if j>i, ctableN(i,j)=i; end
        end
    end
end
for i=1:N
    for j=1:N
        if ~isnan(ctableN(i,j))
            if i>j, ctableN(i,j)=j; end
        end
    end
end
for i=1:N % add diagonal elements
    ctableN(i,i)=i;
end
...
```

```
1 - - 1 - - - - - - 1 - - - - - -
- 1 - - 1 - - - - - - - - - - - -
- - 1 - - 1 - - - 1 - - - - - -
1 - - 1 1 - - - - - - - - - - - -
- - 1 - 1 1 - - - - - - - - - -
- - - - 1 - 1 - - - - - - - - -
- - - 1 - 1 1 - - - - - - - - -
- - - - 1 1 - - - - - - - - - -
- - 1 - - - 1 - - 1 1 - - - - -
1 - - 1 - - - - - 1 1 - - - - -
- - - - - 1 - - - - 1 - - - - -
- - - 1 - - - - - - - - 1 1 - -
- - - - - - - 1 1 - - - - - - -
- - 1 - - - 1 - - 1 1 - - - - -
1 - - 1 - - - - - 1 1 - - - - -
- - - - - - 1 - - - - 1 - - - -
- - - 1 - - - - - - - - 1 1 - -
- - - - - - - - - - - - 1 1 - -
- - - - - - - - - - - - - - 1 -
```

Figure 3

```
sigma=419
```

Figure 4

With this representation, it is straightforward to minimise values iteratively along rows, then along columns, until a steady state is reached in which consistent
connected component values are obtained. (To clarify the situation, the idea is to progressively reduce the numbers in Figure 4 until they represent the lowest of all the component values that need to be connected together.) The resulting connected component values lead to an image of the form shown in Figure 5, with values in the correct forward raster scan order—though not necessarily having adjacent values. We shall rectify the latter problem fully later on, with the final result shown in Figure 6.

![Figure 5](image1)

![Figure 6](image2)

Meanwhile, there is the problem of how to detect whether the steady state has been reached. In fact, it is simple to construct a numeric criterion for this, as we merely need to sum the values of all the numeric elements in the cotable—as is done by the following code, which also prints the cotable values in ASCII format:
sigma=0;
for i=1:N % print cotable in ASCII format
    for j=1:N
        element=cotableN(i,j);
        if isnan(element)
            fprintf(Outtxt2,' -');
        else
            sigma=sigma+element; % find sum of numerical elements
            delta=int8(element-10);
            if element<10, fprintf(Outtxt2,'%2d',element);
            else, fprintf(Outtxt2,' %c',char(delta+'A'));
            end
        end
    end
    fprintf(Outtxt2,'
');
end
fprintf(Outtxt2,'sigma=%3d

',sigma);
...

We are now in a position to present the *while* loop that iterates until a steady state is reached:

oldsigma=999;
fprintf(Outtxt2,'oldsigma=%3d  sigma=%3d\n\n',oldsigma,sigma);
iteration=0;
while oldsigma>siga
    oldsigma=sigma;
    iteration=iteration+1;
    for i=1:N % minimise along rows
        mini=99;
        for j=1:N
            if ~isnan(cotableN(i,j))
                if cotableN(i,j)<mini, mini=cotableN(i,j); end
            end
        end
        for j=1:N
            if ~isnan(cotableN(i,j))
                cotableN(i,j)=mini;
            end
        end
    end
    sigma=0;
    for i=1:N % print cotable in ASCII format
        for j=1:N
            element=cotableN(i,j);
            if isnan(element)
                fprintf(Outtxt2,' -');
            else
                sigma=sigma+element; % find sum of numerical elements
                delta=int8(element-10);
                if element<10, fprintf(Outtxt2,'%2d',element);
                else, fprintf(Outtxt2,' %c',char(delta+'A'));
            end
        end
    end
    fprintf(Outtxt2,'
');
end
fprintf(Outtxt2,'sigma=%3d

',sigma);
...
for j=1:N % minimise along columns
    mini=99;
    for i=1:N
        if ~isnan(cotableN(i,j))
            if cotableN(i,j)<mini, mini=cotableN(i,j); end
        end
    end
    for i=1:N
        if ~isnan(cotableN(i,j))
            cotableN(i,j)=mini;
        end
    end
    sigma=0;
    for i=1:N % print cotable in ASCII format
        for j=1:N
            element=cotableN(i,j);
            if isnan(element)
                fprintf(Outtxt2,' -');
            else
                sigma=sigma+element; % find sum of numerical elements
                delta=int8(element-10);
                if element<10, fprintf(Outtxt2,'%2d',element);
                else, fprintf(Outtxt2,' %c',char(delta+'A'));
            end
        end
        fprintf(Outtxt2,'
');
    end
    fprintf(Outtxt2,'sigma=%3d

',sigma);
    fprintf(Outtxt2,'oldsigma=%3d  sigma=%3d',oldsigma,sigma);
    fprintf(Outtxt2,'  end of iteration%2d

',iteration);
end % while loop
...

Figure 7 shows the complete set of cotables that are iteratively generated by the while loop. Note that such loops necessarily involve more repetition than might have been expected, as it is only by arriving back at the same criterion value that we can be sure that the process has been completed.

The next piece of code labels the output image with the correct number of connected values, as already seen in Figure 5.

for y=1+1:iy-1 % label image with correct number of connected values
    for x=1+1:ix-1
        P0=t1(y,x);
        if P0>0, val=cotableN(P0,P0); else, val=0; end
        delta=int8(val-10);
        if val==0, fprintf(Outtxt1,'  ');
        elseif val<10, fprintf(Outtxt1,'%2d',val);
        else, fprintf(Outtxt1,' %c',char(delta+'A'));
        end
    end
    fprintf(Outtxt1,'
');
end
...
The final piece of code shows how to renumber the initial connected values to give consecutive ordered connected values. It then continues to relabel the output image and to list the three vectors obtained from the ctable as part of this process.

```matlab
elements1=diag(cotableN)'; % contains initial connected values
elements2=zeros(1,N); % 1s will show repeated values
elements3=elements1; % will contain final connected values

P=0;
for M=1:N
    if elements2(M)==0 % not a repeated value
        value=elements1(M);
P=P+1;
        elements3(M)=P; % set the final connected value
        for i=M+1:N
            if elements1(i)==value
                elements2(i)=1; % is a repeated value
                elements3(i)=P; % set the final connected value
            end
        end
    end
end

for y=1+1:iy-1 % label image with final connected values
    for x=1+1:ix-1
        P0=t1(y,x); %
        if P0>0, val=elements3(P0); else, val=0; end
        delta=int8(val-10);
        if val==0, fprintf(Outtxt1,'  ');
else, fprintf(Outtxt1,'%2d',val);
        else, fprintf(Outtxt1,' %c',char(delta+'A'));
    end
    fprintf(Outtxt1,'
');
end

fprintf(Outtxt1,'elements1:');
for i=1:N
    fprintf(Outtxt1,'%3d',elements1(i));
end
fprintf(Outtxt1,'
');

fprintf(Outtxt1,'elements2:');
for i=1:N
    fprintf(Outtxt1,'%3d',elements2(i));
end
fprintf(Outtxt1,'
');

fprintf(Outtxt1,'elements3:');
for i=1:N
    fprintf(Outtxt1,'%3d',elements3(i));
end
fprintf(Outtxt1,'
');
fclose(Outtxt1);
fclose(Outtxt2);
```

The final elemental vectors obtained by the above procedures (needed to progress from Figure 5 to Figure 6) are:

- `elements1`: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 15 1 15 15
- `elements2`: 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1
- `elements3`: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2
end of iteration 2
oldsigma=227  sigma=156
sigma=156

end of iteration 3
oldsigma=156  sigma=152
sigma=152

end of iteration 4
oldsigma=152  sigma=152
sigma=152

Figure 7
Access to the image used for testing the above algorithm

The image 'tangle.bmp' used for generating Figures 1–7 appears in the *Image data* section of the Davies (2017) website.