Disjoint Set Data Structure for Morphological Area Operators

Hai Gao, Ping Xue,
School of EEE,
Nanyang Technological University,
Singapore 639798.
E-mail: {ehgao, epxue}@ntu.edu.sg

Weisi Lin,
Institute for Infocomm Research,
21 Heng Mui Keng Terrace,
Singapore 119613.
E-mail: ws.lin@i2r.a-star.edu.sg

and Chaohuan Hou
Institute of Acoustics,
Chinese Academy of Sciences,
Beijing, 100080, China.
E-mail: houch@public.bta.net.cn

Abstract

Morphological openings and closings are basic operators in mathematical morphology. Morphological area openings and area closings have the same functions as standard morphological openings and closings but avoid distorting object boundaries. The conventional implementation of morphological area operators is based on the heap data structure. A new methodology for implementing morphological area operators, which relies on the disjoint set data structure, is proposed in this paper. The computational complexity of our method is \( O(N) \) where \( N \) is the image size, and not related to the area parameter. Experiments show that the new implementation of grayscale area operators runs faster than the conventional implementations. The computational complexity of binary area operators is also studied in this paper.

1. Introduction

Standard morphological operators such as dilation, erosion, opening, closing, top-hat, etc., are all defined together with a structuring element or a set of structuring elements. For image filtering, standard morphological filters are often designed to smooth noisy gray-level images by a determined composition of opening and closing with a given structuring element. For example, a standard morphological opening (or closing) simplifies the original image by removing the bright (or dark) components that do not fit within the structuring element.

However, the boundaries of objects may be distorted according to the shape of the structuring element used.

Morphological area operators can avoid the disadvantages of standard operators. An area open (or close) operator on an image will remove all bright (or dark) connected components that do not have a minimum area size \( S \). Area openings and area closings depend only on an area parameter \( S \) but not on the shape of the structuring element. In this manner, the area operators avoid the associated problems of imposing the structuring element shape on a processed image.

Morphological area openings and closings have been widely used in image processing in recent years [1]-[6]. The conventional implementation [7] [8] of morphological area operators is based on the heap data structure [9]. A new implementation of morphological area operators relying on the disjoint set data structure, is proposed in this paper. The new implementation for grayscale area operators has significantly lower computational complexity as compared with the conventional implementations. Moreover, the computational complexity of our new implementation is not related to the given area parameter, \( S \). It is observed that our proposed algorithm is more suitable for grayscale area operators than binary ones.

2. Definitions

In this paper, a square grid of 8-connectivity is used to provide the connectivity and the neighborhood relationship among pixels.

2.1 Maximum and minimum

A maximum (or minimum) [10] refers to a connected component in which the pixels are with the same grey-level value and the value of this component is strictly higher (or lower) than the values of its neighboring pixels.

2.2 Binary Area Operators

Binary area opening (or binary area closing): to remove from a binary image all connected components of “1” (or all connected components of “0”) with area less than a given area parameter, \( S \). (see Figure 1 and Figure 2.)

\[
\text{Figure 1. Illustration of binary area opening: (a) original binary image; (b) connected components of “1”; (c) after binary area opening (S=3).}
\]

\[
\text{Figure 2. Illustration of binary area closing: (a) original binary image; (b) connected components of “0”; (c) after binary area closing (S=3).}
\]

2.3 Grayscale Area Operators

Grayscale area opening (or grayscale area closing): to remove from a grayscale image all bright (or dark) connected components which do not have a minimum area of a given parameter, \( S \). (see Figure 3 and Figure 4.)

This work is supported by A-STAR, Singapore under Grant LIT 2002-4.
Figure 3. Illustration of grayscale area opening: (a) original grayscale image; (b) after grayscale area opening with a given area parameter, $S$.

Figure 4. Illustration of grayscale area closing: (a) original grayscale image; (b) after grayscale area closing with a given area parameter, $S$.

Note that a binary image can be regarded as a special case of a grayscale image, in which there is only two gray level values. That is to say, an implementation of grayscale area operators is applicable to binary ones. However, binary images have their own characteristics, and implementing binary area operators could differ from implementing grayscale ones so as to achieve better efficiency. In this paper, emphasis will be put on implementation of grayscale area operators, and implementations of binary area operators will also be discussed.

From the definitions, it is also easy to find out that area openings are similar to area closings. For the sake of simplicity of presentation, we will focus on how area openings are implemented. It is a straightforward task to adapt an implementation of area openings to area closings.

3. Conventional Implementation of Grayscale Area Operators

The data structure used by the conventional implementation of grayscale area opening is the heap [9] data structure. A pixel heap is basically a complete binary tree of pixel pointers, which satisfies the heap condition: the grayscale value of any heap pixel is larger than the value of its children.

There are two basic operations [9] in the heap data structure: insert an element into a heap and extract an element from a heap. Note that the heap condition should be kept in these operations.

Adopting the heap data structure, Luc Vincent [7] [8] had proposed a fast algorithm for grayscale area opening.

Let $I$ be the original grayscale image. The heap based algorithm [7] [8] first extract and label all the maxima [10] of image $I$, and then process each maximum one by one.

For each maximum of $I$, do the following [1] [7] [8]:

1) Initialize the heap and add all neighboring pixels of this maximum to the heap.

2) Do the following until the area is larger than $S$ or another mountain peak is met:

   a) Extract the largest pixel $p$ from the heap.

   b) If pixel $p$ is not larger than the last extracted pixel, pixel $p$ will be labeled and all its neighboring pixels that have not been added into the heap in the processing of this maximum are put into the heap.

   c) If pixel $p$ is larger than the last extracted pixel, this means that another mountain peak is met.

3) Cut the mountain peak associated to this maximum: give all the labeled pixels (including pixels belonging to this maximum) the intensity of the last labeled pixel.

To gain more insight on the actual peak cutting process, we further examine the scenario with multiple maxima within an area whose size is not bigger than $S$. Figure 6(a) illustrates a case with two peaks in a small vicinity. As shown in Figure 6(b–d), the first peak is cut before the second one, and the cutting continues until the resultant area reaches $S$. Apparently, pixels belong to the first peak are visited twice in the heap-based algorithm as shown in Fig. 6.

It can be deduced that if there are $N$ small peaks exist within an area whose size is below $S$, pixels belonging to the first peak will be visited for $N-1$ times, and pixels belonging to the second peak will be visited for $N-2$ times, and so on.

It is worth highlighting that in the conventional algorithm, some pixels are visited for more than one time whereas some pixels might be never visited. The larger the given area parameter $S$, the more pixels are visited and the more times some pixels are visited repeatedly. This is to say, the computational complexity increase with the area parameter $S$. This is the main disadvantage of the heap based algorithm.
4. Disjoint Set Data Structure for Grayscale Area Opening

4.1 Disjoint Set Data Structure

Disjoint sets [11] refer to a collection of mutually exclusive sets. An item must occur in one and only one of those sets.

Hopcroft and Ullman [12] stored disjoint sets in trees. Each set is represented by a tree. In each tree, every node corresponds to an item of a set and every node points to its parent whilst the root of the tree does not have a parent. A set is uniquely identified by the root of the tree.

There are three basic operations [11] in the disjoint set data structure:

- MakeSet(x): create a new set containing a single item, x, which does not exist in any set previously.
- Find(x): return the root of the tree containing x. This operation is performed by starting from node x and following the pointer to the parent until the tree root is reached.
- Union(x, y): Form a new set that is the union of the two sets which contain item x and item y respectively. This operation is performed by first carrying out Find(x) and Find(y) to find out the root of the tree containing x and the root of the tree containing y, and then making one of the roots a child of the other.

A path compression technique [11] [13] can be applied to the Find operation to reduce the average cost of the Find operation.

4.2 Area Opening Based on the Disjoint Set Data Structure

Let us propose a new way to implement grayscale area opening with the disjoint set data structure described in Part A of this section. For an image with N pixels, the trees are implemented in an integer array, named parent, with the same size as the image and parent[p] denoting the parent of a pixel p. When pixel p has no parent (i.e., p being a root of a tree), parent[p] is set to a negative integer.

In our algorithm, pixels are processed in decreasing gray-level order, and the algorithm starts by pixel sorting. We store the sorted pixels in an integer array L of length N. In this paper, the pixels are presented in L using their location indices, image_width*y+x, where x and y are pixel coordinates. To perform pixel sorting, we just need to calculate the gray-level histogram of the image and allocate space for each level in array L accordingly. From left to right and from top to bottom, each pixel of the image is put into array L according to its gray-level value.

As each pixel p is processed, a new singleton set containing pixel p is first created by MakeSet(p), and then check all the neighboring pixels of pixel p. For each already processed neighboring pixel n, if the set containing the current pixel p and the set containing pixel n are not the same one, these two sets are united by Union(p,n) if the following set union criterion is met.

Set union criterion: according to the definition of area opening, the set containing pixel n and the set containing pixel p are united if and only if the area of the former is no larger than the given area parameter, S. There is an exception: if the two roots of the two sets are with the same grey-level value, i.e., the root of the latter set is with the same grey-level value as pixel p, the two sets are united without any constraint.

The root of the united set is determined according to the new root selection criterion: when two sets are going to be united, between the two roots of the two sets, the root pixel that appears later in L will always be chosen as the new root of the new united set. With such new root selection criterion, the current pixel p is always chosen as the new root of the new set.

According to the new root selection criterion, when set union Union(p,n) is carried out, no computation is needed to figure out the root of the set containing the current pixel p because the root must be p since the pixels are stored in decreasing grey-level order; furthermore, comparison of two roots is not necessary because the root of the set containing p is definitely the new root of the united set. When Union(p,n) is carried out, we only need to find out the root of the set containing pixel n, and then make this root be a child of pixel p.

According to set union criterion, the two neighboring sets should be united if and only if the area of the the set not containing pixel p is no larger than a given area parameter, S, except that if the two roots are with the same grey-level value, the two sets should be united without any constraint.

The exception of the set union criterion can be explained in more details: in the heap-based algorithm, the peaks are cut one by one while in the disjoint set based algorithm, all the peaks are cut simultaneously. In the disjoint set based algorithm, if two neighboring set are with the same grey-level value, this means the two peaks had already been cut to become one. And the two sets are in fact one connected component. That’s why we should unite the two sets without any constraint.

This can also be viewed as that the pixels in the disjoint set based algorithm are processed level by level. We first process pixels with level 255, and then we process pixels with level 254, and so on. At the end of the processing of each level, if the area of a set is no larger than S, the peak corresponding to this set should be cut. In other words, this set will be united with some other set in the next steps of processing for lower level pixels.

After all the pixels are processed, a resolving step is carried out to get the area opening result. It can be done easily by giving each set the grey-level value of its root. In this resolving step, the pixels are visited in the reverse processing order, i.e., parents are always visited before their children. A pixel to be visited is either a root with no parent or a pixel whose parent is already resolved. If a root pixel is visited, its grey-level value will not be changed; if a non-root pixel is visited, it will be given the same value as its already visited parent.
5. Complexity Analysis and Experimental Results

5.1 Complexity Analysis

As we have pointed out in Section 3, the computational complexity of the heap-based algorithm is not only related to the area parameter $S$, but also related to the image contents, to be more exact, related to the number of maxima in the image and the distribution of these maxima. So it is not able to find out an accurate formula to describe the computational complexity of the heap-based algorithm. However, we will give a rough analysis on the computational complexity of this algorithm. The heap-based algorithm can be divided into two steps: the maximum detection step, and the peak cutting step. The maximum detection [10] extracts all maxima in an image and requires two scannings of the whole image (i.e., it runs in $O(N)$ time). In the peak cutting step, for a heap of size $S$, adding a pixel into the heap or extracting a pixel from the heap takes $O(\log S)$ time on average [9]. The computational complexity of this step has relations with the number of the heap operations carried out. Suppose each pixel is added into the heap once and extracted from the heap once, the peak cutting step will run in $O(N \cdot \log S)$ time. So the two-step algorithm runs in $O(N(1 + \beta \cdot \log S))$ time where $\beta$ is a positive coefficient related to the number of maxima in the image and the distribution of these maxima. It is expected that $\beta$ differs when this algorithm is applied to different images.

The new algorithm based on the disjoint sets has three steps: pixel sorting, set union, and pixel resolving. In pixel sorting, only two scannings of the image with size $N$ are necessary to construct the array $L$ of pixel location indices. An additional scanning of the grey-level histogram is also required, but its complexity is negligible as compared to that of the scanning of the whole image. In pixel resolving, one scanning of the array $parent$ with size $N$ is needed. So both the pixel sorting and the pixel resolving are with the complexity being linear to $N$. As for the set union step, the trees dynamically grow by means of new node insertions (referred to as incremental tree set union [11]). When $Union(p,n)$ is carried out, we only need to find out the root of the set containing pixel $n$, and then make this root be a child of pixel $p$. Gabow and Tarjan [14], as well as Leobl and Nesetril [15] showed that the algorithms for incremental tree set union in such a case involves with the complexity linear with $N$. With all three steps, the overall complexity of the algorithm is linear to $N$, i.e., the entire algorithm runs in $O(N)$ time. As can be seen obviously, the computational complexity of our new algorithm is not related to the given area parameter $S$ as compared with the conventional algorithm.

5.2 Experimental Results

Figure 7 gives an example of area opening and area closing of the luminance component. In this example, the given area parameter $S$ is 80. We can see that in Figure 7(b) all “bright” components with area less than 80 are removed, and in Figure 7(c) all “dark” components with area less than 80 are removed.

We tested our algorithm on images with different sizes and using different area parameter values. The images for testing are the first frames of the table tennis sequences in QCIF, SIF, and CIF formats. Table I lists the timing results of the conventional implementation based on heap data structure whereas Table II lists the timing results of our new implementation. The C-program is tested in a Pentium IV PC of 2.4GHz with 512M of RAM. From Table I, we can see that the processing time for the conventional algorithm is proportional to image size $N$, and is increased with the area parameter, $S$. As can be seen in Table II, the time for our new algorithm is also proportional to image size $N$, but is not related to the area parameter, $S$.

![Figure 7](image-url)
Table I. Timing results of the conventional implementation based on the heap data structure.

<table>
<thead>
<tr>
<th>Image Size (N)</th>
<th>S=40</th>
<th>S=80</th>
<th>S=120</th>
<th>S=160</th>
<th>S=200</th>
</tr>
</thead>
<tbody>
<tr>
<td>176*144(QCIF)</td>
<td>41.1ms</td>
<td>45.6ms</td>
<td>48.6ms</td>
<td>50.4ms</td>
<td>51.9ms</td>
</tr>
<tr>
<td>352*240(SIF)</td>
<td>166.3ms</td>
<td>183.5ms</td>
<td>195.6ms</td>
<td>202.9ms</td>
<td>208.8ms</td>
</tr>
<tr>
<td>352*288(CIF)</td>
<td>199.6ms</td>
<td>220.2ms</td>
<td>234.7ms</td>
<td>243.5ms</td>
<td>250.6ms</td>
</tr>
</tbody>
</table>

Table II. Timing results of our new implementation based on the disjoint set data structure.

<table>
<thead>
<tr>
<th>Image Size (N)</th>
<th>T_h (seconds)</th>
<th>T_d (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>176*144(QCIF)</td>
<td>0.009</td>
<td>0.028</td>
</tr>
<tr>
<td>352*240(SIF)</td>
<td>0.031</td>
<td>0.093</td>
</tr>
<tr>
<td>352*288(CIF)</td>
<td>0.037</td>
<td>0.112</td>
</tr>
</tbody>
</table>

Table III. Timing results of binary area opening. (T_h is the time for the algorithm based on the heap data structure; T_d is the time for the algorithm based on the disjoint set data structure.)

For the heap based binary area opening, it should be noted that no maximum detection is needed. We just need to label the connected components of “1” and count the area of the connected components. During the labeling, no sorting operation is needed when a pixel is added into a heap or extracted from a heap because all the pixels are with the same grey-level value “1”. And pixels are added into the heap without area constraint. For binary area opening, the heap is in fact a queue. Each adding or extraction operation runs in \( O(1) \) time. After the labeling, we need to change the value of the connected components with area less than the given area parameter, \( S \).

6. Implementation of Binary Area Opening

Both the heap based algorithm and the disjoint set based algorithm for grayscale area opening can be used to deal with binary images. However, the binary images have their own characteristics, and both algorithms should be slightly modified to achieve better efficiency.

For the disjoint set based binary area opening, we just need to process pixels with grey-level value “1”, no pixel sorting is needed. We just need to process pixels with grey-level value “1” in the reverse order, i.e., visit pixels with grey-level value “1” from right to left and from top to bottom in the image. In the pixel resolving step, we just need to resolve pixels with grey-level value “1” in the reverse order, i.e., visit pixels with grey-level value “1” from left to right and from top to bottom in the image. In the image. In the process of the algorithm based on the heap data structure; \( T_h \) is the time for the algorithm based on the disjoint set data structure.)

7. Conclusions

We have proposed a new implementation methodology for morphological grayscale area openings and closings based on the disjoint set data structure. Our new implementation is much faster than the conventional algorithm which is based on the heap data structure, for grayscale area operations. The larger the given area parameter \( S \), the more times of the processing time are saved, since the computational complexity of the proposed method is not related to the area parameter whereas the computational complexity of the conventional method increases with the area parameter. It is also clarified that the conventional algorithm based on the heap (or queue) data structure is more suitable for binary area operations.

REFERENCES


