Available online at www.sciencedirect.com
SCIENCE DIRECT®
Pattern Recognition Letters 26 (2005) 1266–1274
Pattern Recognition Letters
www.elsevier.com/locate/patrec

A fast watershed algorithm based on chain code and its application in image segmentation

Han Sun *, Jingyu Yang, Mingwu Ren

Department of Computer Science, Nanjing University of Science and Technology, Xiaolinwei 200, Nanjing, Jiangsu 210094, China

Received 21 September 2004
Available online 10 December 2004

Abstract

In this paper, a novel algorithm is proposed for the watershed transformation. This new algorithm is based on chain code. The traditional concept of chain code is first expanded into point-out chain code and point-in chain code. Then two theorems, which depict how to make watershed transformation based on chain code, are proposed and proved. In the end, the description of this new algorithm is presented, and its complexity is discussed in detail. Experiments show that the presented algorithm runs fast. Further more, the new algorithm can provide more information to the following image processing.

Keywords: Watershed; Chain code; Connected component; Steepest path; Image segmentation

1. Introduction

The traditional watershed transformation algorithm is usually implemented by simulating a flooding process (Hagyard et al., 1996; Vincent and Soille, 1991). During the processing, the image is taken as a surface of mountainous terrain, and the gray value of each pixel denotes the altitude of that point. In this terrain, there exist deep valleys (*minima*), high ridges (*watershed lines*), and steep or gentle hillsides (*catchment basins*). Firstly, holes are pierced in each minimum, and then this surface is slowly immersed into a lake. Starting from the minima of lowest altitude, water will progressively fill up the different catchment basins. Now, at the points where the water coming from different minima would merge, a 'dam' is built to prevent intermingle. At the end of the flooding procedure, each minimum is completely surrounded by dams, which delimit its associated catchment basins. All the dams correspond to the watersheds that are needed. Fig. 1 shows such one-dimension flooding process.

* Corresponding author. Fax: +86 25 84315510.
E-mail address: henrysun2000@vip.sina.com (H. Sun).

0167-8655/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2004.11.007

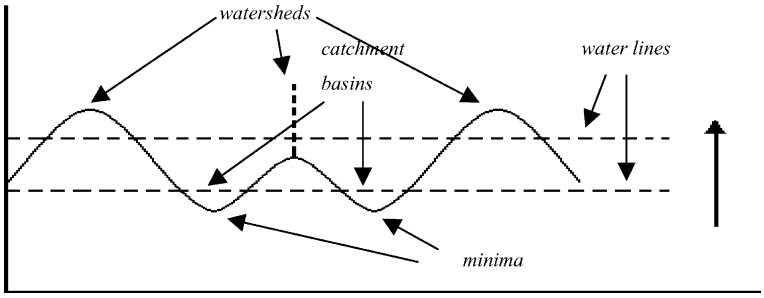

Fig. 1. One-dimension flooding process.

Vincent and Soille (1991) proposed the classical watershed algorithm (called *Algorithm VS*). This algorithm has two main steps: sorting and flooding. First, all pixels are sorted in the ascending order of their gray values. So the pixels of the same gray value can be directly accessed simultaneously. Once the pixels have been sorted, the progressive flooding can be processed from the minimal gray level. Suppose the flooding has been done up to a given level h, each catchment basin that is already discovered—i.e., those catchment basins whose corresponding minimum has the altitude lower or equal to h—is supposed to have a unique label. Then the pixels of altitude $h + 1$ will be processed. Those pixels among them, which have at least one labeled neighbor, are put into a first-in-first-out queue. Starting from these pixels, the queue structure enables to extend the labeled catchment basins including them by computing geodesic influence zones. Then those remained unlabeled pixels of level $h + 1$ are regarded as new minima, and given new labels. In the result, all the pixels belong to the same catchment basin have the same label, and those pixels have the same geodesic distance to different catchment basins are labeled as watershed pixels. Such watershed transformation is very close to the original concept of watershed, and can be easily comprehended. But its complexity is a little high because of the additional step of sorting. In addition, it cannot do image segmentation completely because the watershed pixels in broad watershed zones do not belong to any segmented regions.

Bieniek and Moga (2000) proposed a distinct watershed algorithm based on connected components (called *Algorithm BM*). This algorithm is implemented by simulating a raining process, not flooding as usual. When a raindrop drops on the image surface, it must flow along hillside down to the according valley. The route which raindrop passes is just a connected component, and is also a steepest path between fall-point and the valley. All the connected components, which lead to the same valley, form a catchment basin associated to it. This algorithm can label all catchment basins by only scanning the whole image four times. Compared with these methods implemented by simulating flooding, this algorithm is faster, and can segment image completely.

This paper presents a new watershed algorithm based on chain code. The novelty of this approach is to adopt chain code to depict connected components. The new algorithm first simulates raining to generate connected components using chain code instead of pixel address used in *Algorithm BM*, and then simulates flooding to label catchment basins by tracing chain codes. It also only scans image four times, and can segment image completely. Compared with *Algorithm BM*, the proposed algorithm can describe connected components more clearly, and provides more information to the following processing.

The paper is organized as follows: In Section 2, the traditional definition of chain code is expanded into point-out chain code and point-in chain code. Then two theorems, which depict how to make watershed transformation based on chain code, are proposed and proved. In Section 3, the description of the new algorithm is presented, and then its complexity is discussed in detail. Experiments and conclusions are shown in Section 4.

2. Definitions and theorems for chain code based watershed transformation

Chain code is a very common tool in image processing. This paper first expands the traditional concept of chain code into point-out chain code and point-in chain code.

Definition 1. *Point-out chain code* is the directional code that current pixel points to one of the eight neighbor pixels, shown in Fig. 2(a).

The value of point-out chain code can be any one in the set {0, 1, 2, 3, 4, 5, 6, 7, and 8}, where the code '8' means current pixel does not point to any neighbor pixel. This paper restricts each pixel can only point to at most one neighbor pixel.

Definition 2. *Point-in chain code* is the directional code that neighbor pixel points to current pixel, shown in Fig. 2(b).

Because more than one neighbor pixel can point to current pixel at the same time, let one bit record one point-in directional code, thus one byte can record all the eight directional codes. So the value of one pixel's point-in chain code should be the result obtained from OR operation on the elements of the set {0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, and 0x80}, where the code '0x00' means no neighbor pixel points to current pixel.

According to the definition of point-out and point-in chain code, the connectivity of two pixels p_i and p_j can be expressed by a string of chain codes. Starting from p_i, we can go along point-out chain codes to p_j; on the contrary, starting from p_j, we can also go along point-in chain codes back to p_i. As is known, the pixels in one connected component are connected. So chain codes can be used to denote and label such connected components.

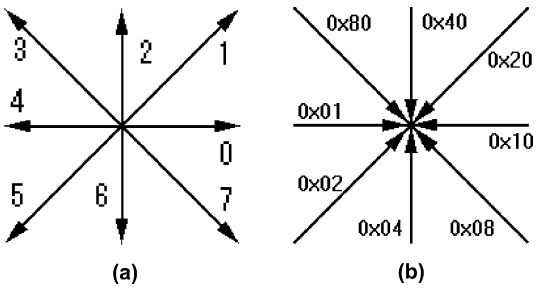

Fig. 2. Point-out and point-in chain code: (a) point-out chain code and (b) point-in chain code.

In the following, two theorems, which depict how to make watershed transformation based on chain code, will be proposed and proved.

Theorem 1. *In an image, the steepest path from any pixel to according regional minimum pixel can be expressed by a string of point-out chain codes.*

Proof. The steepest path L from current pixel p_l to according regional minimum pixel p_0 is composed of such a string of pixels: $p_l p_{l-1}, \ldots, p_1 p_0$, where their gray values $f(p_i)$ satisfy the following condition, $f(p_l) \geqslant \cdots \geqslant f(p_i) \geqslant f(p_{i-1}) \geqslant \cdots \geqslant f(p_0)$ $(i = l, \ldots, 2, 1)$, and pixel p_{i-1} is the lowest neighbor of pixel p_i. Obviously, such steepest path can also be described by means of chain code. Starting from pixel p_l, let point-out chain code of every pixel p_i points to the lowest neighbor, whose value should be less than or equal to current pixel's value. Then all the pixels linked by the string of point-out chain codes constitute the steepest path from p_l to p_0. Therefore, if all the pixels in an image have such point-out chain codes, then each pixel can go along steepest path to its according regional minimum. Now how to label every pixel's point-out chain code is discussed in the following. For the sake of discussing more clearly, the same sample image data as in (Bieniek and Moga, 2000) is used, shown in Fig. 3(a).

As mentioned above, the value of lowest neighbor pixel to be pointed should be less than or equal to current pixel's. So the way of labeling point-out chain code could be divided into two steps according to the two relationships of gray value between current pixel and its neighbors.

The pixels that have neighbors of lower gray value are firstly to be processed. Let $f(p)$ be the gray value of current pixel p, $f(p')$ be the gray value of neighbor pixel p', and $N(p)$ be the set of neighbor pixels. Then the steepest neighbor pixel q is defined as $q = \{p' \in N(p) | f(p') = \min_{p'' \in N(p)} f(p''), f(p') < f(p)\}$. Therefore, according to the definition of steepest path, the point-out chain code of current pixel p should point to the

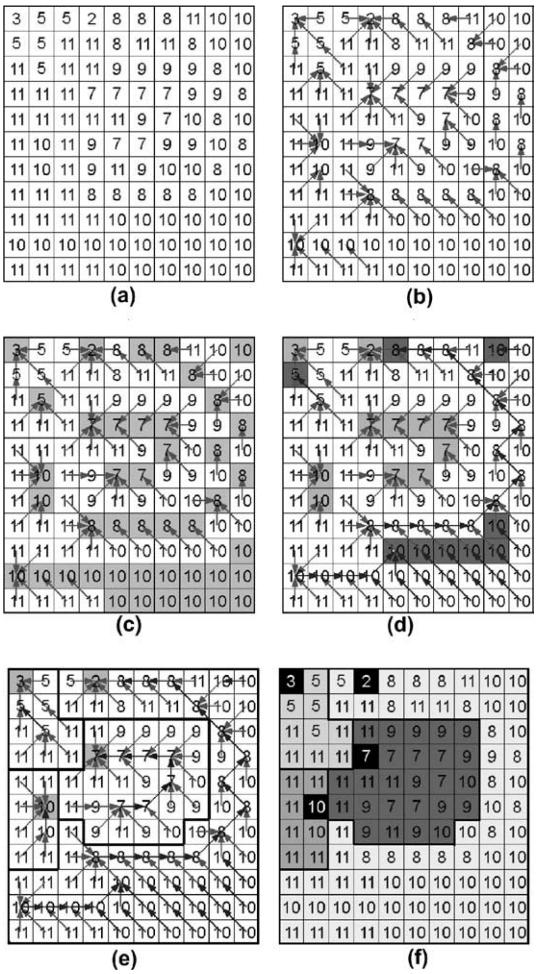

Fig. 3. Watershed transformation based on chain code.

steepest neighbor q. For instance, Fig. 3(b) is the result of such processing on Fig. 3(a), in which 67 pixels that satisfy such condition are labeled point-out chain codes, and the arrows denote the directions of point-out chain codes.

Then it is turn to label remained pixels, which are marked with gray color in Fig. 3(c). These pixels do not have lower neighbors, so their point-out chain codes cannot be designated like above. As observed in Fig. 3(c), these remained pixels are located in two places: minima and non-minima plateaus. On the edge of non-minima plateau, these pixels, marked with darker color in Fig. 3(d), have the same gray value as those in the plateau and have been labeled point-out chain codes in the above step. So the pixels in non-minima plateau can be labeled by propagating from these edge pixels through a fist-in-first-out queue, and the result is shown in Fig. 3(d). As for those in minima plateaus, there does not exist such *seed* pixels. So an additional step is needed to designate fictitious seed pixels. Choose the first pixel in each minima plateau, marked with gray color in Fig. 3(e), and let its neighbors of the same gray value point to it. Then these neighbors have point-out chain codes and can be regarded as seeds for propagating. The final result can be seen in Fig. 3(e).

Now only one pixel in each minima plateau has not been labeled point-out chain code. But these pixels are just the regional minimum pixels, or the end points of steepest paths. In this paper, they are designated a special point-out code, '8'.

Up to now, the method to label all the pixels' point-out chain codes is presented. Therefore, Theorem 1 proves to be right. For example, in Fig. 3(a), the steepest path from pixel $(0,9)$ to its according minimum pixel $(0,3)$ can be described by such pixel string linked by point-out chain codes: $(0,9) \rightarrow (0,8) \rightarrow (1,7) \rightarrow (0,6) \rightarrow (0,5) \rightarrow (0,4) \rightarrow (0,3)$.

In fact, the process that every pixel goes along point-out chain codes to according minimum is just a rain-simulating process. □

Theorem 2. *Catchment basin can be labeled by tracing point-in chain codes from according minimum pixel.*

Proof. As is known from Theorem 1, all the pixels in a catchment basin can reach the same minimum pixel along their respective steepest path linked by the string of point-out chain codes. When labeling point-out chain code, the corresponding point-in chain code could be labeled at the same time. Then starting from the minimum pixel, all the pixels, accessed by tracing point-in chain codes, should belong to the same catchment basin. Such tracing process will be terminated when all the pixels in the associated catchment basin are visited. As for Fig. 3(e), the result of such catchment basin labeling is shown in

Fig. 3(f), in which every catchment basin is marked with different color.

In fact, the tracing process is just a process of simulating flooding. □

3. Algorithm description and complexity analysis

3.1. Algorithm description

According to above discussion, chain code based watershed algorithm is described as below:

Input: source image f
Output: label image l
Step 0: Initialize
$PointOut(\cdot)$: Point-out chain code, initialized to 8;
$PointIn(\cdot)$: Point-in chain code, initialized to 0.
Step 1: Label point-out and point-in chain codes of the pixels having lower neighbors
p: Current pixel
$p' \in N(p)$: Neighbor pixel of p
$f(p)$, $f(p')$: The gray value of p, p'
$q = \{p' \in N(p) \mid f(p') = \min_{p'' \in N(p)} f(p''), f(p') < f(p)\}$: The steepest neighbor
$PO\_Code(p \to q)$: Point-out chain code of p pointing to q
$PI\_Code(p \leftarrow q)$: Point-in chain code of p being pointed by q

```
• Raster scan (p) {
    If (Exist(q)) {
      PointOut(p) = PO_Code(p → q);
      PointIn(q) = PointIn(q)|
                   PI_Code(q ← p)
    }
  } // End scan
```

Step 2: Label point-out and point-in chain codes of the pixels in non-minima plateaus

```
(1) Raster scan (p) {
      If (PointOut(p)! = 8 and
          PointOut(p') == 8 and
          f(p) == f(p'))
        fifo_put(p)
      }// End scan
(2) While (fifo_empty() == FALSE) {
      p = fifo_get()
      For each (p' ∈ N(p) and f(p) == f(p')
      and  PointOut(p') == 8) {
        PointOut(p') = PO_Code(p' → p);
        PointIn(p) = PointIn(p)|
                     PI_Code(p ← p')
        fifo_put(p')
      }// End for
    }// End while
```

Step 3: Label point-out and point-in chain codes of pixels located in minima plateaus

```
• Raster scan (p) {
  If (PointOut(p) == 8 and PointOut(p') ==
  == 8 and f(p) == f(p')) {
    PointOut(p') = PO_Code(p' → p);
    PointIn(p) = Point In(p)|
                 PI_Code(p ← p')
    stack_push(p')
  }// End if
  While (stack_empty() == FALSE){
    p' = stack_pop()
    For each (p'' ∈ N(p') and f(p'') == f(p')
    and PointOut(p'') == 8
    and PointOut(p')! → p){
      PointOut(p'') = PO_Code(p'' →  p');
      PointIn(p') = PointIn(p')|
                    PI_Code(p'  ← p'')
      stack_push(p'')
    }// End for
  }// End while
} End scan
```

Step 4: Label catchment basins by tracing point-in chain codes

```
(1) current_label = 1
(2) Raster scan (p){
      If (PointOut(p) == 8){
        stack_push(p)
        While (stack_empty() == FALSE){
          p' = stack_pop();
          l(p') = current_label
          For each (PI_Code(p' ← p'')
          In PointIn(p')){
            l(p'') = current_label;
            stack_push(p'')
          }// End for
        }// End while
        current_label = current_label + 1;
      }// End if
    }// End scan
```

In the algorithm, Steps 1–3 respectively label point-out and point-in chain codes according to three different types of pixels. At Step 2, some pixels that belong to watershed pixels or watershed zones in (Vincent and Soille, 1991) are designated to according catchment basins using an approximation of the geodesic distance. So this algorithm can segment image completely. At Step 3, the statement *If* is used to designate *seeds* for propagating in minima plateaus. When all the point-out and point-in chain codes are labeled, Step 4 is used to mark the labels of catchment basins. Up to now, the whole procedure of chain code based watershed transformation is presented in detail.

3.2. Complexity analysis

Given an image with n pixels, including n_1 pixels that have lower neighbors, n_2 pixels in the non-minima plateaus, and n_3 pixels in the minima plateaus, where $n = n_1 + n_2 + n_3$. Now the complexity of the proposed algorithm is analyzed step by step.

At Step 1 a linear scan is performed, so the cost of this step is $O(n)$. At Step 2, a linear scan is first performed to find the edge pixels on non-minima plateaus, and then n_2 pixels are inserted into the FIFO queue. So the overall cost of Step 2 is $O(n + n_2)$. At Step 3, only a linear scan is needed to label all the pixels on minima plateaus, therefore the cost is $O(n)$. At Step 4, every pixel is traced only once, so this step's cost is also $O(n)$. As a result, the overall time complexity of the algorithm is $O(4n + n_2)$.

As concerns the memory requirements, this algorithm uses (1) an input image buffer f; (2) an output label image buffer l; (3) buffers for store point-out and point-in chain codes; (4) buffers for queue and stack, but they can share a common space. Thus the space occupancy is not huge.

4. Experiments and conclusion

As is known, the image segmentation based on watershed follows such basic steps, shown in Fig. 4. In the procedure, the preprocessing of gradient calculation is essential. Only in the gradient image, the boundaries of objects could be located on the ridges and taken as watershed pixels, and then the watershed algorithm could work in the right way. At the same time, the step of noise reduction and the method of calculating gradient also affect the initial region numbers of watershed transformation. Because the watershed transformation often causes over-segmentation, the step of region merging should be introduced. These aided steps are important and attract researchers' interest greatly, which could be found in many literatures.

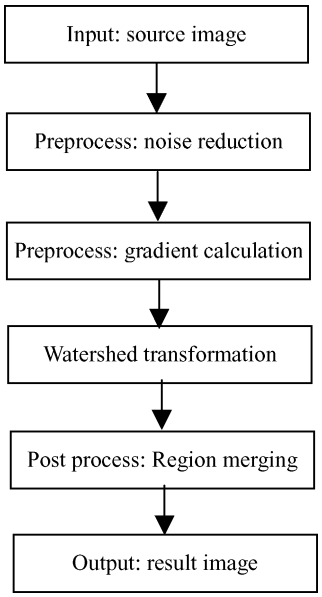

Fig. 4. Watershed-based image segmentation procedure.

In the following experiments, the gradient operator is defined as

$$G(p) = \max_{m\times m}(N(p)) - \min_{m\times m}(N(p)), \tag{1}$$

where $\max_{m\times m}(N(p))$ and $\min_{m\times m}(N(p))$ are the maximum and minimum values in $m \times m$ neighbor pixels. Such gradient operator could reveal the watershed pixels more clearly.

In this paper, the method of region merging is derived from Haris et al. (1998). The main idea of this method is that the most similar pair of adjacent regions should be merged first. In the method,

the nearest neighbor graph (NNG) is used to store the similarity of adjacent regions. And the similarity of regions is described as

$$\delta(R_i,R_j) = \begin{cases} \dfrac{S_{R_i} \times S_{R_j}}{S_{R_i} + S_{R_j}} \times |\mu_{R_i} - \mu_{R_j}| \times E_{R_i,R_j} \\ \quad R_i \text{ and } R_j \text{ are adjacent,} \\ \infty \quad \text{otherwise,} \end{cases} \tag{2}$$

where S_{R_i}, S_{R_j} are the region areas of region R_i, R_j; μ_{R_i}, μ_{R_j} are the mean values of region R_i, R_j; E_{R_i,R_j} is the edge intensity between R_i and R_j. It is a little different from that in (Haris et al., 1998). Here the edge intensity is introduced, which is an important factor in region merging. In the end, the terminate condition of region merging could be judged by the threshold value of the similarity degree or the region numbers which is known in advance.

The test images, Lena, brain and blood, are shown in Fig. 5(a), Fig. 6(a) and Fig. 7(a). Fig. 5(b), Fig. 6(b) and Fig. 7(b) are the gradient images, which are computed by formula (1), where $m = 3$. The watershed transformation results are shown in Fig. 5(c), Fig. 6(c) and Fig. 7(c). Fig. 5(d), Fig. 6(d) and Fig. 7(d) are the results of region merging, and the threshold values of the similarity degree are 700,000, 800,000 and 1,800,000, respectively. It is obviously that the result is determined by such threshold value. If the threshold value is not appropriate, the result could not be satisfying.

Table 1 gives the detailed data of the presented watershed algorithm compared with *algorithm VS* and *algorithm BM*, which are performed on a PC with a P4 2.4 GHz CPU. Seen from Table 1, three conclusions could be drawn: (1) The three algorithms produce the same number of regions, that is to say, they are identical in function. (2) Compared with *Algorithm VS* and the new algorithm, *Algorithm VS* does not do segmentation completely, and a number of watershed pixels do not

Fig. 5. Watershed transformation and region merging (Lena): (a) Lena (512 × 512); (b) gradient image, $m = 3$; (c) watershed results (10,656 regions); (d) merging results (47 regions).

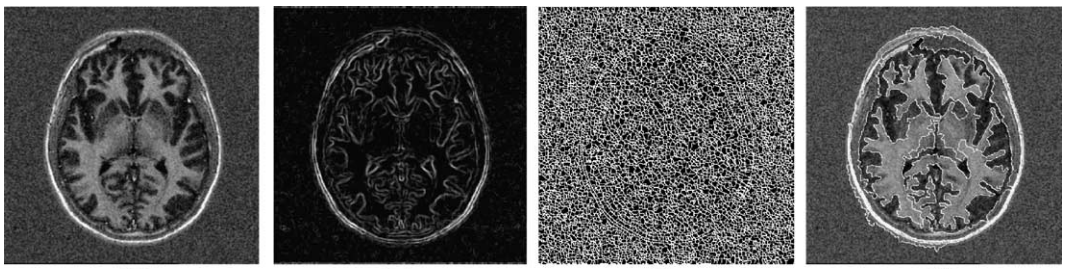

Fig. 6. Watershed transformation and region merging (brain): (a) brain (388 × 395); (b) gradient image, $m = 3$; (c) watershed results (6227 regions); (d) merging results (11 regions).

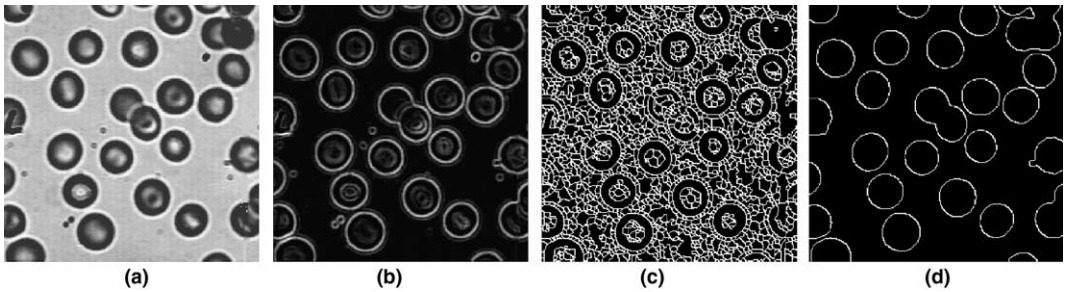

Fig. 7. Watershed transformation and region merging (blood): (a) blood (272×265); (b) gradient image, $m = 3$; (c) watershed results (1644 regions); (d) merging results (27 regions).

Table 1
Result of three different watershed algorithms

Test images	Parameters	Algorithm VS	Algorithm BM	New algorithm
Lena (512×512)	Regions	10,656	10,656	10,656
	Watershed points	44,452	0	0
	Running time (ms)	234	67	64
Brain (388×395)	Regions	6227	6227	6227
	Watershed points	29,605	0	0
	Running time (ms)	141	35	33
Blood (272×265)	Regions	1644	1644	1644
	Watershed points	7392	0	0
	Running time (ms)	47	20	20

belong to any regions. (3) *Algorithm VS* consumes more time than *Algorithm BM* and the new one because of the sorting operation. Further more, the proposed algorithm is a little faster than *Algorithm BM*, because the complexity of *Algorithm BM* is $O(4n + n_2 + n_3 \log n_3)$, which needs additional time on *FIND* operation.

In addition, the proposed watershed algorithm is more helpful to the following processing. First, starting from any pixel, its associated catchment basin could be traced by chain codes without scanning the whole image. But the other watershed algorithms do not have the ability. Second, in Step 4 of the proposed algorithm, the region-related information, such as size and average of gray value, could be calculated when labeling catchment basins, which is useful to the following region merging.

In conclusion, this paper proposed the chain code based watershed algorithm, which solves the classical problem from a new point of view. Experiments show that the new algorithm's time and space complexity is very low. Further more, such transformation is more helpful to the following image processing.

In nature, the watershed transformation cares the ordering relation between current pixel and its neighbors. For example, the gradient is often regarded as such relation in gray images. So the proposed watershed algorithm could be extended to color images only if choosing appropriate relation, such as LUV gradient in (Shafarenko et al., 1997).

References

Bieniek, A., Moga, A., 2000. An efficient watershed algorithm based on connected components. Pattern Recogn. 33 (3), 907–916.

Hagyard, D., Razaz, M., Atkin, P., 1996. Analysis of watershed algorithms for grayscale images. In: Proc. IEEE Internat. Conf. on Image Processing, March, pp. 41–44.

Haris, K., Efstratiadis, S.N., Maglaveras, N., Katsaggelos, A.K., 1998. Hybrid image segmentation using watersheds and fast region merging. IEEE Trans. Image Process. 7 (12), 1684–1699.

Shafarenko, L., Petrou, M., Kittler, J., 1997. Automatic watershed segmentation of randomly textured color images. IEEE Trans. Image Process. 6 (11), 1530–1544.

Vincent, L., Soille, P., 1991. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. IEEE Trans. Pattern Anal. Machine Intell. 13 (6), 583–598.