Available online at www.sciencedirect.com
ScienceDirect
Pattern Recognition 40 (2007) 1078–1090
PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY
www.elsevier.com/locate/pr

An improved watershed algorithm based on efficient computation of shortest paths

Víctor Osma-Ruiz*, Juan I. Godino-Llorente, Nicolás Sáenz-Lechón, Pedro Gómez-Vilda

Dpt. of Ingeniería de Circuitos y Sistemas, Universidad Politécnica de Madrid, Ctra. Valencia, Km. 7, 28031, Madrid, Spain

Received 5 July 2005; received in revised form 10 March 2006; accepted 27 June 2006

Abstract

The present paper describes a new algorithm to calculate the watershed transform through rain simulation of greyscale digital images by means of pixel arrowing. The efficiency of this method is based on limiting the necessary neighbouring operations to compute the transform to the outmost, and in the total number of scannings performed over the whole image. The experiments demonstrate that the proposed algorithm is able to significantly reduce the running time of the fastest known algorithm without involving any loss of efficiency.

Keywords: Watershed; Image segmentation; Arrowing

1. Introduction

The watershed transform is one of the most valued tools in the field of digital image segmentation [1]. One of the main advantages of this technique lays in the fact that the result is a set of well delimited areas, so if we consider that these areas represent the searched objects, we will obtain an accurate edge detection defined by a set of connected pixels. This is a great improvement with respect to other segmentation techniques, which address edge detection through unconnected lines that allow the differentiation of the objects at first glance, but are rather complicated when dealing with automatic systems.

The concept of watersheds comes from the field of topography, referring to the division of a landscape in several basins or water catchment areas. A good example is the *continental divide* that separates the USA into two main regions: one associated with the Atlantic Ocean, and another associated with the Pacific Ocean. So, on rainy days, all the drops of rain that fall on one side of the divide flow into one ocean, while rain falling on the other side of the division will flow into the other ocean. It is clear that the water will reach the ocean provided that it is not trapped in a local minimum along the way. Both regions are usually named catchment basins, and each one has an associated minimum (the oceans). The border line that separates both basins is called the watershed line, corresponding to the continental divide in the example. From this point of view, we can consider the image as a topographic surface where each pixel is a point situated at some altitude as a function of its grey level [1–3]. The white colour (grey level 255) is taken to mean the maximum altitude and the black colour (grey level 0) the minimum. The rest of the grey levels match an altitude associated to the image between these extremes.

Given this definition, during the last years, two conceptually distinct techniques have been developed to calculate the watershed transform:

- The first method proposes the transformation by flooding the topographic surface. This technique basically involves gradually immersing the surface in a water container. Previously, a hole has been made in each of the surface minima. The water will begin to flow through the holes, first through those with less altitude but gradually reaching those with a greater altitude. Progressively

* Corresponding author. Tel.: +34 91 336 78 32; fax: +34 91 336 78 29.
E-mail addresses: vosma@ics.upm.es (V. Osma-Ruiz), igodino@ics.upm.es (J.I. Godino-Llorente), nicolas.saenz@upm.es (N. Sáenz-Lechón), pedro@pino.datsi.fi.upm.es (P. Gómez-Vilda).

0031-3203/$30.00
doi:10.1016/j.patcog.2006.06.025

all the catchment basins associated to the minima are flooded. The water coming from the flooding of two or more different basins might converge. At this point, suppose that a dyke is built to prevent the joining. Once the whole surface is immersed, only the dykes will rise above the water level, making up the watershed lines. The watersheds or catchment basins are all the areas surrounded by the lines. There are several algorithms implementing this technique [4,5], which have been subsequently improved [6–10] and even implemented in hardware [11].

- The second method simulates the rain over the surface associated with the image. The drops that fall over a point will flow along the path of steepest descent until reaching a minimum. Such a point is labelled as belonging to the reception basin associated with this minimum. This process is repeated for all the points on the surface, so in the end every point will be assigned to a minimum and the surface will be divided into its catchment basins. In this approach, no point will explicitly belong to a watershed line, because every point is labelled as belonging to a certain basin. The lines will therefore be formed by the edges of the pixels that separate the different basins. This method has been implemented in Ref. [1], and improved in Refs. [12–14]. There also exist other studies focusing on hardware adaptation [15], and parallel processing [16,17].

In this brief description we have not differentiated between methods based on topographic distance [18,19] and methods based on local conditions (defined here as rain simulation) because they are conceptually rather similar [17]. However, it is important to mention that most methods based on topographic distance also produce watershed lines as do those based on immersion.

Most of the processes that follow the flooding criterion face the common problem that large image areas could form a part of the watershed lines, whereas it is desirable that these lines will never have a width greater than the minimum resolution in the digital image (usually one pixel). This fact can be clearly observed in the image in Fig. 1. The catchment basins are represented in light grey, whereas the points with darker grey are those that constitute the watershed lines. The numbers in each cell represent the grey level of the corresponding pixel. Moreover, the points labelled as belonging to a watershed line tend to slow down the merging process that usually follows the calculation of the watershed transform [20–25].

The techniques based on rain-flow simulation do not present the above mentioned problems, because they label all the points as belonging to a basin. In Fig. 2, a different grey level has been used for each of the four catchment basins that divide the image. The thick line represents the border line, i.e., the equivalent to the former watershed lines, this time presenting an ideal width. This paper presents a new algorithm capable of computing the watershed transform according to this criterion, but more efficiently and with lower running time than current approaches.

10	10	12	14	20	20	15	13	11	11
10	10	12	14	20	20	15	13	11	11
12	12	12	14	20	20	15	13	13	13
14	14	14	14	20	20	15	15	15	15
20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20
18	18	18	18	20	20	19	19	19	19
17	17	17	18	20	20	19	18	18	18
15	15	17	18	20	20	19	18	17	17
15	15	17	18	20	20	19	18	17	17

Fig. 1. Watershed lines and catchment basins obtained using the algorithm of Vincent and Soille [6]. In dark grey are represented the watershed lines. In light grey the catchment basins. The numbers represent the grey level of the pixels.

10	10	12	14	20	20	15	13	11	11
10	10	12	14	20	20	15	13	11	11
12	12	12	14	20	20	15	13	13	13
14	14	14	14	20	20	15	15	15	15
20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20
18	18	18	18	20	20	19	19	19	19
17	17	17	18	20	20	19	18	18	18
15	15	17	18	20	20	19	18	17	17
15	15	17	18	20	20	19	18	17	17

Fig. 2. Watershed transform of an image following the criterion of simulation based on raining of Sun et al. [14].

In Refs. [12,14] two algorithms are presented that calculate the watershed by means of rain simulation. The image is explored 4 times (plus another scan to initialize the data structures) allowing a running time lower than the fastest at that moment [6,18].

The algorithm proposed in this paper produces the same result as the previous works using only two scans (plus another to initialize the data structures), thus decreasing the running time obtained by Sun [14] and Bieniek [12]. The effectiveness of the algorithm obtains the same results as the other methods developed to compute the watershed transform. Both assertions are demonstrated throughout this paper and supported by several examples.

The paper is organized as follows: Section 2 describes the basic concepts about watershed transform by means of rain simulation. In Section 3 these basic concepts are developed and applied to build the proposed algorithm. Section 4 demonstrates the efficiency and effectiveness of the algorithm through several experiments. And finally, Section 5 presents the conclusions.

2. Review of watershed concepts

This section presents a review of the basic concepts and terminology to build the watershed transform of a digital image using a rain simulation.

2.1. Definition of regional minimum

A regional minimum is a point or a group of connected points with the same grey level, where none of them have a neighbour with a lower grey level [1,3,26]. In order to determine the connection and neighbourhood of the points, the degree of connectivity has to be defined: it is typically either 8 (each point is considered connected with all its neighbours in vertical, horizontal and diagonal directions) or 4 (each point is considered connected with all its neighbours in vertical and horizontal directions, but not in diagonal) [1]. All the examples in this paper assume 8-connectivity.

Watershed methods are used to establish a topographic analogy between digital images and the way the water flows through a real terrain surface. According to this analogy, for rain simulation-based methods, a regional minimum is the area of the surface where the rain water would get trapped without flowing to lower levels.

In Fig. 3, a circle surrounds the grey level of those pixels belonging to the four regional minima that exist in the image.

29	27	15	13	28	11	15	10	28	29
25	26	9	10	15	14	(8)	11	26	27
20	22	5	8	9	(8)	(8)	10	22	24
15	11	4	(1)	16	(8)	9	14	15	20
12	10	9	10	17	15	23	13	10	14
20	19	18	15	16	18	10	(5)	9	10
23	13	11	14	20	19	12	(5)	7	6
26	12	(9)	10	22	23	15	(5)	(5)	8
27	13	11	15	11	32	11	(5)	(5)	11
29	21	22	18	23	30	27	25	26	15

Fig. 3. An example of regional minima in a digital image.

2.2. Definition of steepest descending path

A descending path from a point (x, y) is a series of connected points with the origin in (x, y), where each point presents a grey level strictly lower than the previous one. Logically, every descending path will end in a regional minimum, because a minimum does not have any neighbour with a lower grey level.

There may exist several descending paths from any given point, because there could be more than one neighbour with a grey level below that at the point that is being analysed. A descending path is said to be a steepest descending path if each point in the path is connected to the neighbour with the lowest grey level. Note that the steepest descending path, like descending paths, need not be unique [1].

Following with the topographic analogy of rain simulation, the steepest descending path is the path that a drop of water would follow while flowing down to a regional minimum.

Fig. 4 shows the steepest descending path originating from point $(0, 0)$ (the upper-left corner) to the regional minimum where it ends. It also shows, in darker grey, two steepest descending paths beginning at point $(8, 5)$ and ending in a different minimum. The choice between one or the other depends only on the implementation.

2.3. Definition of catchment basins

The watershed transform implemented using rain simulation consists of dividing the image into a set of catchment basins, so each point of the image belongs to one and only one of these catchment basins [12] (called watersheds in Ref. [1]).

A catchment basin is formed by a regional minimum and all the points whose steepest descending path fall in that minimum [12].

29	27	15	13	28	11	15	10	28	29
25	26	9	10	15	14	(8)	11	26	27
20	22	5	8	9	(8)	(8)	10	22	24
15	11	4	(1)	16	(8)	9	14	15	20
12	10	9	10	17	15	23	13	10	14
20	19	18	15	16	18	10	(5)	9	10
23	13	11	14	20	19	12	(5)	7	6
26	12	(9)	10	22	23	15	(5)	(5)	8
27	13	11	15	11	32	11	(5)	(5)	11
29	21	22	18	23	30	27	25	26	15

Fig. 4. Example of steepest descending paths. In light grey *is* represented the steepest descending path that originates from point $(0, 0)$ until a minimum situated in $(3, 3)$. In dark grey, two steepest descending paths that start from point $(8, 5)$ to two different regional minima.

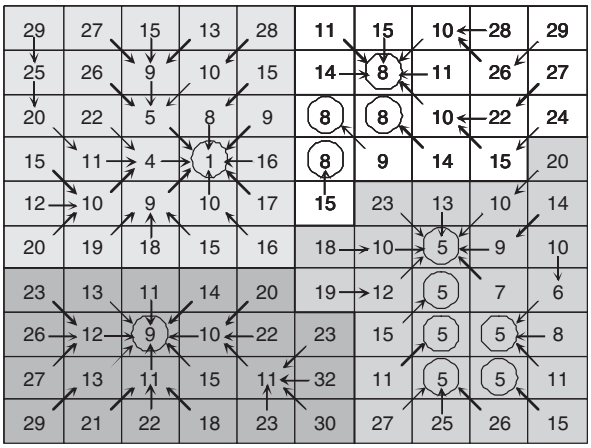

Fig. 5. Catchment basins of the example image.

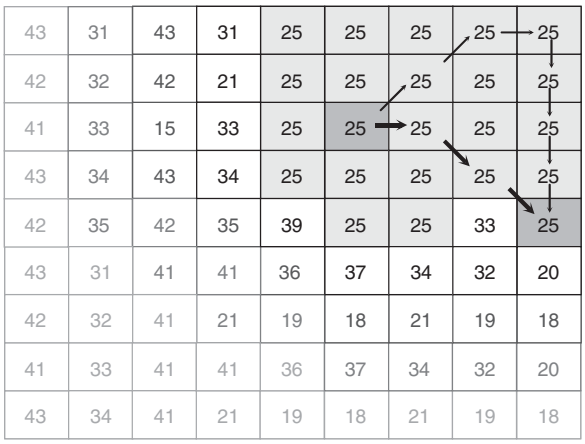

Fig. 6. Representation of the geodesic distance between two points. The domain X consists of pixels with grey level equal to 25. The degree of connectivity is 8.

Thus, according to the topographic analogy, a catchment basin is an area of the image with an associated regional minimum, such that if a drop of water should fall in any given point of that area, it would descend to its minimum following the steepest descending path of that point.

As mentioned above, a point might have several steepest descending paths ending in different minima. Because a point may also belong to different catchment basins, some criterion must be chosen to determine which basin to use. Examples include: grey level of the minimum associated to each catchment basin, position of the catchment basins in the image, randomly, etc.

In Fig. 5, the image has been segmented into four catchment basins, marked with different grey levels, following the steepest descending paths of their pixels. The regional minima of each catchment basin are indicated with a circle, and the steepest descending paths have been represented with arrows indicating the direction of descent. In order to discriminate among the steepest descending paths that start from the same point, highest priority has been given to those going up to the left. The priority decreases as we move to the right and down, so the right-down direction has the lowest priority. This criterion is the one used in the algorithm presented in Section 3.

2.4. Definition of geodesic distance

The geodesic distance between two points of an image (x_1, y_1) and (x_2, y_2) belonging to a certain domain X (a subset of points of the image) is defined as the minimum length of any path from (x_1, y_1) to (x_2, y_2) without leaving the domain X [1,5,6].

In digital imaging it is usual to consider that the distance between a point and any of its neighbours is unity, independent of the direction (vertical, horizontal or diagonal).

Fig. 6 shows the geodesic distance between two points of the image, considering as the domain X a plateau of grey level 25. The starting and ending point are represented with dark grey, and the domain in lighter grey. The thick arrows indicate the geodesic path, and the geodesic distance is 3. The thin arrows indicate another possible path between both points, but with a longer distance.

2.5. Definition of lower-complete image

Many images have regions where the pixels have the same grey level. When these regions do not form a regional minimum they are called non-minima plateaus [12,14] and they pose a problem in calculating the watershed transform. This is because the computation of this transform requires that all the points that do not belong to a regional minimum have at least one neighbour with a lower grey level. This approach ensures that all the points that do not correspond with a minimum will have a steepest descending path and will belong to a catchment basin [1].

To achieve this goal it is enough to sort those points belonging to a non-minima plateau in such a way that all of them have a lower neighbour. To accomplish this sorting, the points of these plateaus are divided into two different groups:

- Descending edge points of the plateau: this group consists of every point of the plateau that has a neighbour with a grey level less than its grey level.
- Inner points of the plateau: this group consists of every point of the plateau whose neighbours have grey levels of equal or higher value than its grey level.

Subsequently, the geodesic distance of an inner point of the plateau to all the points of the edge (with the plateau as domain) is calculated. The lowest value will represent the geodesic distance from this point to the descending edge of

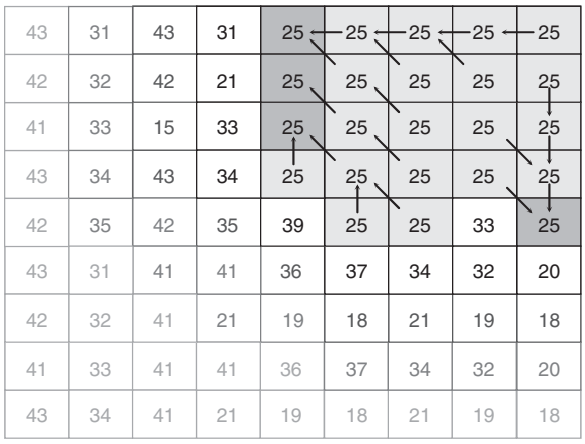

Fig. 7. Arrowing of a plateau with a grey level of 25, considering a geodesic distance from the inner points to the edge of the plateau.

the plateau. The process is repeated for all the inner points of the plateau. The points belonging to the edge of the plateau will have a geodesic distance equal to 0. Once the process is finished, all the points of the image (except those in a minimum) will have a lower neighbour, because if two points have the same grey level, the one with a lower geodesic distance to the descending edge is considered to have a lower grey level. The new image, that keeps an account not only of the grey levels but also of the geodesic distances, to determine the level of each point, is called lower-complete image.

Fig. 7 presents a set of arrows that illustrates the steepest descending path of each point that belongs to the non-minima plateau, by considering the geodesic distance from the inner points to the edge of the plateau. To calculate the geodesic distance, an iterative process is applied, starting from the edge points (represented in dark grey in Fig. 7). The neighbours of every pixel in the plateau are scanned and are made to point to the starting pixel, providing they do not point to any other pixel of the plateau yet.

If we would like to know the exact geodesic distance from any point to the edge of the plateau, we would only have to count the arrows along the path between that point and the edge where the arrows are leading. Nevertheless, this is not necessary because the arrows indicate the steepest descending path for each point belonging to the non-minima plateau. This is not the only way to calculate the geodesic distance, but it is very useful for our implementation.

2.6. Definition of watershed transform

The watershed transform is a matrix of points of the same size as the original image, where each point has been labelled using its steepest descending path as belonging to a unique catchment basin.

From a topographic point of view, the watershed transform represents the division of the surface into its water catchment basins.

The goal is that each catchment basin matches an object in the image. Nevertheless, the result of the watershed transform is usually disappointing, due to the fact that thousands of catchment basins arise where only a few were expected [1]. Following the topographic analogy, this is like pretending to divide a country into catchment basins according to rivers and oceans but detecting even the potholes in the roads. This problem is called oversegmentation and is mainly due to noise in the image. The best solution is to merge the catchment basins after the watershed transform, following various criteria as described in existing literature [1,20–25,27].

3. Algorithm description

3.1. Description of the elements and basic operations

There now follows a brief description of the nomenclature employed to represent data and operations that have been used throughout the paper:

- Input matrix (image): $\boldsymbol{F}$.
- Output matrix (watershed transform): $\boldsymbol{W}$.
- Grey level of pixel p in image $\boldsymbol{F}$: $\boldsymbol{F}(p)$.
- Label of pixel p in the output matrix: $\boldsymbol{W}(p)$.
- Neighbour of pixel p in image $\boldsymbol{F}$: $\boldsymbol{N}(p)$.
- Minimum grey level of the neighbours of a point p in the image $\boldsymbol{F}$, below the grey level of point p: $\min\{\boldsymbol{F}(\boldsymbol{N}(p))\}$.
- Traversing the points of an image: *Scan*(p).
- Insert a pixel p into a queue: *queue_name-put*(p).
- Extract a pixel p from a queue: $p =$ *queue_name-get*().
- Arrowing operation: $\boldsymbol{W}(p) \rightarrow p'$ (p points to p').

All the queues used to build the algorithm are FIFO (first in, first out), i.e. the first element in, is the first element out. In this context, a queue is only a segment of memory to store numbers (such as the position of a pixel), allowing operations over the stored values. The size of the queues (i.e. the allocated memory) is managed using adjoined blocks. If any value has to be stored in the queue, the allocation is carried out for a group of them, so the next value is likely to find free memory remaining, and a new allocation is not needed. The allocated memory is reused after each iteration and is only freed at the end of the whole process. With this approach, the amount of memory used is greater, but the time spent by the algorithm to manage the memory is lower. Bearing in mind that nowadays memory is relatively cheap in comparison with the cost in terms of time, it is considered a good solution.

The regional minima and steepest descending paths must be marked in the output matrix $\boldsymbol{W}$. A positive number is used to identify the regional minima and a number less than or equal to 0 is used to identify the arrowing (the steepest descending paths). The criterion to assign directions in the arrowing is summarized in Fig. 8. The constant value -8

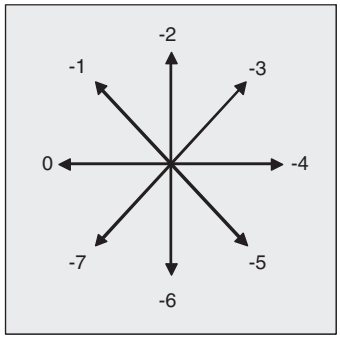

Fig. 8. Arrowing criterion used to implement the algorithm to indicate the direction of the steepest descending paths.

(represented in the algorithm with the label UNVISITED) indicates that the point has not been analysed yet, and −9 (label PENDING) means that it is in the process of being labelled.

For example, let us suppose that the point located at (2, 5) has a neighbour belonging to its steepest descending path in (3, 6). Then the position (2, 5) of the output matrix has to be labelled with a −5.

3.2. Algorithm description

The algorithm presented is able to perform the watershed transform by scanning the image just twice (plus once for initialization). The swiftness of the algorithm to calculate the catchment basins is based on a reduction of the number of neighbourhood operations. Neighbourhood operations introduce a high running time; for example, assuming 8-connectivity, such an operation involves manipulating the 8 surrounding pixels. However, this process is mandatory for every algorithm that calculates the watershed transform. The key to efficiency is making best use of each neighbourhood operation to do all the possible calculations, thus reducing the number of scans.

Following this idea, the algorithm is implemented as follows:

Step 0: Initialization

```
Define UNVISITED = −8 and PENDING = −9
Scan(p){
    W(p) = UNVISITED
}
NumCatchment = 1
Create 4 queues: qPending, qEdge, qInner, qDescending
```

Step 1: Identifying regional minima and steepest descending paths

```
Scan(p){
    //If the point has not been analyzed yet, study it
    If (W(p) = UNVISITED){
        ∀p' belonging to N(p){
            If (F(p') = F(p)){          //This is a plateau
                If (qPending = Ø){ W(p) = PENDING; qPending−put(p)}
                W(p') = PENDING;
                qPending−put(p')
            }
            Else If (F(p') = min{F(N(p))}){
                Min = p'
            }
        }
        //If p belongs to a plateau, make it lower-complete image
        //If not, p is considered a minimum unless there is a
        //lower neighbour
        If (qPending ≠ Ø){
            While (qPending ≠ Ø){
                p' = qPending−get()
                If (p' ≠ p){          //Calculations already done for seed
                    //Put in the queue all the points in the plateau
                    ∀p'' belonging to N(p'){
                        If (F(p'') = F(p)){
                            If (W(p'') = UNVISITED){
                                W(p'') = PENDING; qPending−put(p'')
                            }
                        }
```

```
                    Else If (F(p'') =min{F(N(p'))}){
                        Min =p''
                    }
                }
            }
            //Classify the point as either an edge or inner point
            If (∃ Min){W(p') → Min; qEdge−put(p')}
            Else{qInner−put(p')}
        }
        //If the plateau has no edge points, it is a minimum
        //If the plateau has edge and inner points, make it
        //lower-complete
        If (qEdge ≠ Ø){
            If (qInner ≠ Ø){
                While (qEdge ≠ Ø){
                    p' =qEdge−get()
                        ∀p'' belonging to N(p'){
                            If (F(p'') =F(p') and W(p'') =PENDING){
                            W(p'') → p'
                            qEdge−put(p'')
                        }
                    }
                }
            }
            Else{
                While (qInner ≠ Ø){
                    p' =qInner−get()
                    W(p') =NumCatchment
                }
                NumCatchment =NumCatchment+1
            }
        }
        Else{
            If(Not ∃ Min){
                W(p) =NumCatchment;
                NumCatchment =NumCatchment+1
            }
            Else{W(p) → Min}
        }
    }
}
```

Step 2: Assignment of pixels to catchment basins

```
Scan(p){
    p' =p
    While (W(p') <= 0){          //It is not a minimum
        qDescending−put(p')
        p' =Point pointed to by p' (labelled in W)
    }
    While (qDescending ≠ Ø){
        p'' =qDescending−get()
        W(p'') =W(p') //W(p') =label of c. basin of previous while
    }
}
```

3.3. Explanation of the algorithm

Step 0 initializes the output watershed matrix to UNVISITED, meaning that no analysis has yet been carried out.

Step 1 performs all the operations needed to identify the regional minima used to calculate the catchment basins, and to identify the steepest descending paths by means of arrowing. As part of this process, the geodesic distance of those points that belong to the non-minima plateaus is calculated, in order to identify the steepest descending paths. The process is as follows:

1. Check whether the point has yet been analysed. It is possible that some points could have already been analysed because when a point that belongs to a plateau is detected, all the points of the plateau are also labelled.
2. If the point has not yet been processed, each one of its neighbours will be analysed with two objectives: to detect whether or not it belongs to a plateau (it has a neighbour with the same grey level), and to detect the neighbour with the lowest grey level that is less than the current grey level. If there are several neighbours matching this condition, the point has two or more steepest descending paths. The criterion employed to select just one path is to choose the first according to the scanning order (left to right, up to down).
3. If the point does not belong to a plateau, two possibilities are considered: (a) the point has a neighbour with a lower grey level; if so, this point will be marked in the watershed matrix pointing to this neighbour; (b) all the neighbours have a higher grey level, and if so the point will be marked as a minimum. Determining whether there is a neighbour with a lower grey level has been performed in the previous operation.
4. If the point belongs to a plateau, all the components are analysed to divide them into edge points (points with a neighbour of a lower grey level) and inner points (the rest). The process is as follows: all the points with the same grey level as the current one that have not been analysed (that is, having an UNVISITED in the output matrix $\boldsymbol{W}$) are put into the queue *qPending* and, to avoid repetitions, they are marked in the output matrix with a value PENDING. At the same time, those neighbours with the same grey level are queued into *qPending,* and classified as edge or inner points. Moreover, if these points belong to an edge they are marked in the output matrix $\boldsymbol{W}$ with their lowest neighbour by means of the arrowing technique specified before. All this is done in only one neighbourhood operation.
5. Three situations may occur when the queue *qPending* is empty: (a) the plateau only has inner points and, if so, they are classified as regional minima; (b) the plateau only has edge points, and if so all the points that belong to a plateau will have a lower neighbour already marked; (c) the plateau has edge points and inner points, and if so the geodesic distance has to be calculated to order the points that belong to the plateau. The proposed implementation provides an approximation to the geodesic distance that directly gives an arrowing of all the points. It consists of a new scan of the plateau beginning at the edges, so each neighbour will point to that edge. The following iterations of the algorithm will check the neighbours of those points that have already been labelled, making these neighbours point to their central pixel. The iterations will continue until all the points of the plateau have been labelled.

At the end of step 1, the matrix $\boldsymbol{W}$ will contain all regional minima points labelled with a positive number (different for each minimum), and the rest of points are labelled to indicate the position of their lowest level neighbour. Step 2 goes over the matrix to associate them with the minimum where each steepest descending path ends. The process is as follows: all the points belonging to a steepest descending path are stored in the queue, descending until a point is reached that is labelled as part of a catchment basin. Then the points stored in the queue are labelled with the same number as this catchment basin.

After step 2, all the points have been labelled as belonging to one and only one catchment basin, so the watershed transform of the image is complete.

Fig. 9 presents an example of the execution of our algorithm to divide an image into its catchment basins. Fig. 9a represents the original image $\boldsymbol{F}$. Fig. 9b represents the detection of the minima and steepest descending paths obtained after step 1 of the algorithm (the minima are surrounded with a circle; the light grey cells represent the inner part of the non-minima plateaus; and the edges are represented with dark grey). Fig. 9c represents the output matrix $\boldsymbol{W}$ before the execution of step 2 of the algorithm (for the sake of clarity, the same grey levels as in the previous figure are used). Finally, Fig. 9d presents the resulting watershed transform $\boldsymbol{W}$ after applying step 2 (where each catchment basin is represented with a different grey level).

3.4. Complexity analysis

The presented algorithm requires only two scans (excluding the initialization) to perform the watershed transform.

Step 1 of the algorithm scans all the pixels of the matrix at least once. Those pixels belonging to the non-minima plateaus (with inner and edge pixels) are scanned twice, because they have to be put into two different queues. Let N be the number of pixels of the image, and N_{NMP} the pixels that belong to the non-minima plateaus. The computational cost of step 1 is $O(N + N_{NMP})$.

In step 2, each pixel is added to and removed from a queue to be labelled, regardless of whether it belongs to a plateau or not. This operation is carried out for all pixels except those representing regional minima. Considering N_{RM} the

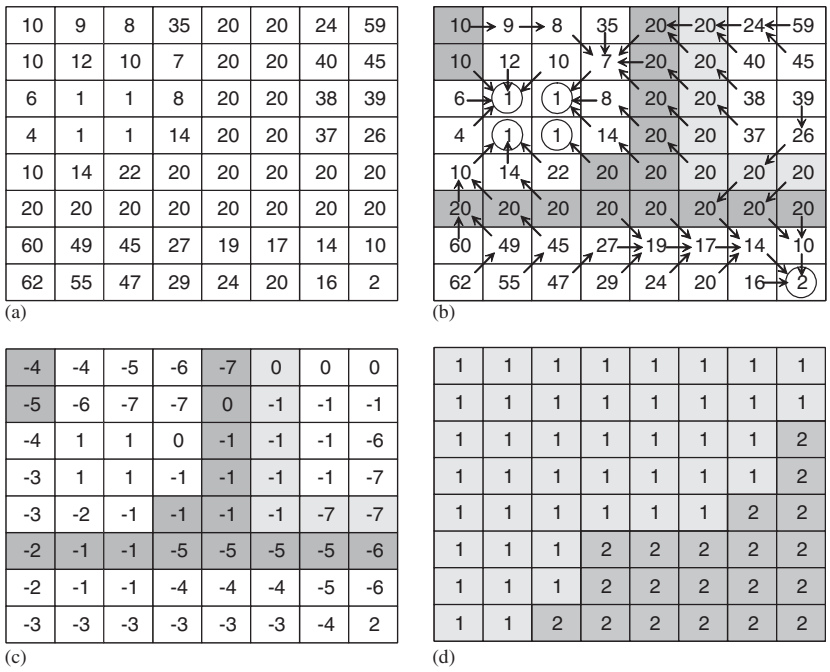

Fig. 9. From left to right, top to down (a, b, c, d): (a) Original image, (b) detection of minima and steepest descending paths carried out in step 1 of the algorithm, (c) output matrix after step 1, (d) output matrix after step 2.

number of regional minima, the computational cost of step 2 is $O(N - N_{RM})$.

The total computational cost of the algorithm is $O(2N + N_{NMP} - N_{RM})$. Sun [14] and Bieniek [12] reported a higher complexity and higher execution time.

Concerning memory requirements, the proposed algorithm only needs an input matrix, the output matrix and four FIFO queues. The size of these queues is smaller than those required in [14]. This fact will be demonstrated in Section 4.

4. Evaluation of the algorithm and comparisons with previous works

4.1. Introduction

In order to demonstrate the efficiency of the new algorithm, several tests have been carried out to measure the computational cost. The running time is compared with that obtained using the algorithm previously developed by Sun, Yang and Ren (called *SYR* from now on) [14], which is demonstrated to have a running time quite similar to that obtained by Bieniek and Moga (*BM*) [12], and lower than that obtained by Vincent and Soille (*VS*) [6]. The tests performed in this evaluation demonstrate that the proposed algorithm is faster than *SYR*.

On the other hand, the present paper is limited to present an algorithm to calculate the watershed transform of any image in an efficient way, and so the measurements are carried out over original images in greyscale, leaving aside the pre-processing (for example, a gradient operation) and post-processing (merging), because these operations are independent of the algorithm for obtaining the watershed transform. However, the algorithm simplifies the post-processing, as *SYR* does

- During step 2 of the algorithm, it is possible to extract several features of each catchment basin, such as its size, the average grey level of its pixels, the standard deviation, etc.
- Starting from any pixel, its associated catchment basin can be identified by arrowing without scanning the whole image.

4.2. Results

Table 1 presents the results of our experiments carried out with images of different sizes (Fig. 10). The table summarizes the time spent by the proposed algorithm and the time spent by *SYR* to calculate the watershed transform. It also specifies the number of catchment basins obtained for each algorithm. All measurements have been performed using an IBM PC Pentium III, equipped with a 1 GHz processor, and 640 MB of RAM.

Looking at the results over images of various sizes, it is easy to see that the proposed algorithm decreases the average running time by more than 31%.

Table 1
Comparison of the new algorithm with *SYR*

Test images (size)	Parameters	Algorithm SYR	New algorithm	% Improvement
Lena (1024 × 1024)	Running time (ms)	1242	951	23.43
	Catchment basins	19 856	19 856	
Baboon (1024 × 1024)	Running time (ms)	1144	779	31.91
	Catchment basins	24 613	24 613	
Peppers (1024 × 1024)	Running time (ms)	1220	913	25.16
	Catchment basins	26 101	26 101	
Lena (512 × 512)	Running time (ms)	309	212	31.39
	Catchment basins	13 953	13 953	
Baboon (512 × 512)	Running time (ms)	287	185	35.54
	Catchment basins	18 197	18 197	
Peppers (512 × 512)	Running time (ms)	298	203	31.88
	Catchment basins	16 143	16 143	
Baboon (256 × 256)	Running time (ms)	72	44	38.89
	Catchment basins	5781	5781	
Peppers (128 × 128)	Running time (ms)	28	18	35.71
	Catchment basins	729	729	

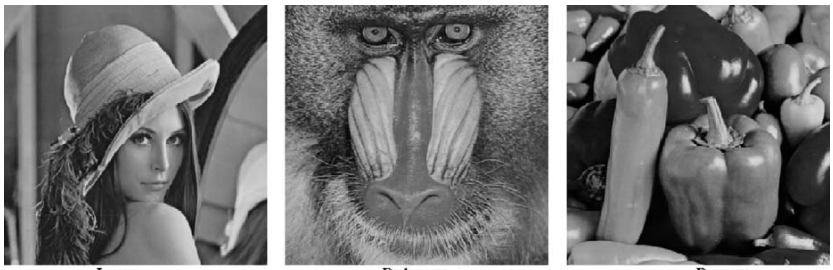

Fig. 10. Test images used for the evaluation of the algorithms.

In terms of effectiveness, our results are the same as any of the other algorithms found in the state of the art (*VS*, *BM* and *SYR*). In [14] it is demonstrated that the algorithm *SYR* obtains the same results as *VS* and *BM*. In Table 1 it is demonstrated that the new algorithm obtains the same results as *SYR*, in terms of the number of catchment basins.

4.3. Some issues concerning implementation

Our implementation of the *SYR* algorithm has been carefully done to obtain the maximum efficiency. In this sense, several improvements have been developed over the algorithm presented in Ref. [14].

The first improvement is to create two fixed memory tables to perform the conversion of the *point-in chain-code* values to the real location of those points that have to be scanned in the last step of the *SYR* algorithm [14]. The two tables have 256 positions (from 0 to 255), each one representing a different value of all possible chain-code values that might appear at a point. The first table establishes the number of *point-in* chain codes starting at a given point, and the second establishes the relative position of each one with respect to the current. Without these tables, the running time of the algorithm would be around 200% higher. For "Baboon 512 × 512", the running time goes from 287 ms to 670 ms. Regarding the *point-out chain codes* a similar improvement has been developed, although the gain is not significant. As they can only point to one location, their conversion can be easily done on the fly. Concerning the proposed algorithm, the arrowing technique works in the same way as the *point-out chain-codes*, so no improvements are needed.

Another improvement developed for the *SYR*, prior to any performance comparison, is related with the way the memory for the queues is allocated. Fig. 11 shows the running time versus the amount of free memory space allocated each time the queues need to increase their size, demonstrating that it is a sensitive choice. The memory management is done as in the proposed algorithm, explained in Section 3.1. Memory allocation by blocks and memory reuse guarantee the minimization of the temporal cost due to these tasks. In Fig. 11, the images "Lena (1024 × 1024)" and "Baboon (512 × 512)" have been used to support this idea. Both show the sensitivity of the algorithm with respect to the basic allocation unit. The sensitivity is clearly influenced by step 2 of the algorithm *SYR* [14], where the points belonging to the

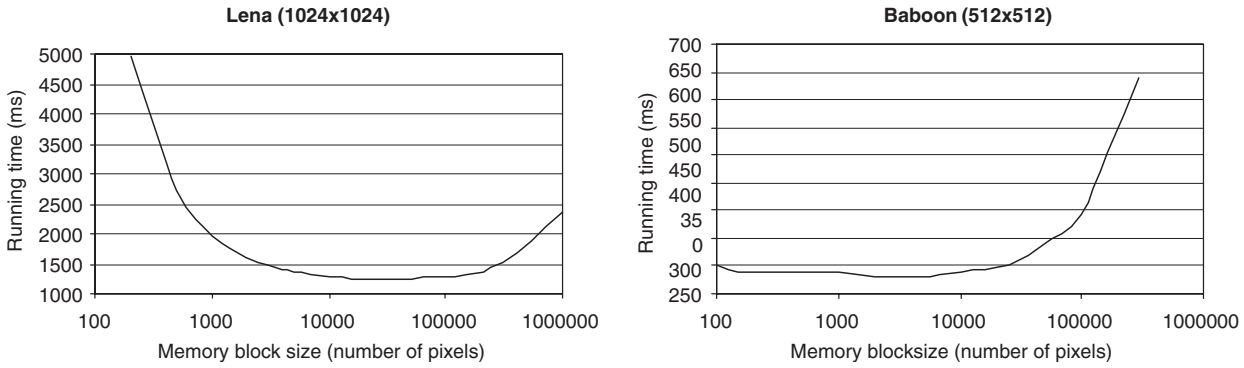

Fig. 11. Running time of the SYR algorithm as a function of the basic memory unit used for the implementation. Left plot: Lena (1024 × 1024). Right plot: Baboon (512 × 512). The x-axis is represented in log scale.

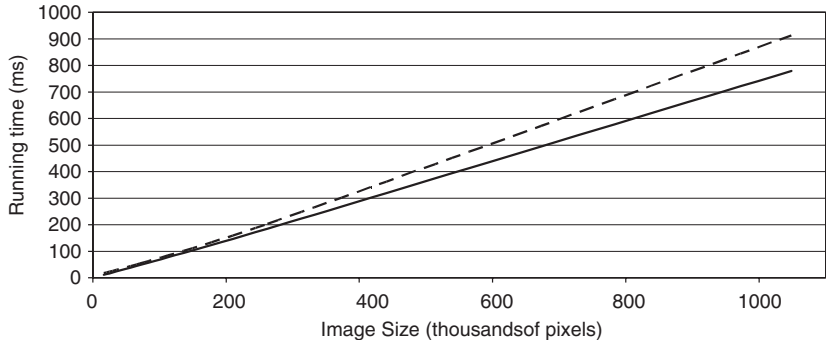

Fig. 12. Running time of the proposed algorithm as a function of the number of pixels in the image. Dashed line: peppers. Solid line: Baboon.

non-minima plateaus are analysed. To carry out this task, the whole image is scanned and the edge points found are stored. These points may belong to different plateaus. While storing these points in a queue, there are certain images that need a lot of memory, like "Lena (1024 × 1024)", which has 47 755 edge points belonging to different plateaus. It takes a time of 4196 ms using memory blocks of 200 elements just to perform step 2, which implies a significant increase with respect to the times presented in Table 1. The other image in Fig. 11, "Baboon (512 × 512)", presents only 2226 edge points, so this problem is not so important. In this example, the time spent in step 2 is around 90 ms using memory blocks of 200 elements.

Looking at the plots it is clear that there exists a minimum around the block size of 50 000 for Lena and from 2000 to 10 000 for the Baboon image. This fact indicates that the best results are obtained by adjusting the initial size to the number of edges in the plateau. The problem is that it is not possible to know a priori the number of edges of the plateau. In any case, it is logical to think that the number of edge pixels in an image is related to the size of the image, and so a compromise must be established between the basic unit of memory as a function of the size of the image. After the analysis of the obtained data, we chose to allocate as much memory as the 2.5% of the number of pixels in the image. Reservations below this threshold produce poorer results due to the fact that the memory allocating operations are more frequent. Moreover, allocations over this threshold produced poorer results because much more memory than was needed was being reserved. For example, "Lena 1024 × 1024", with either small (0.02% of the image) or big memory blocks (100% of the image), yielded far worse results than the optimum (Table 1).

In the proposed algorithm, the queues used to analyse the plateaus are reset for each new plateau that is found while scanning the image. This provides a great reduction of memory that enables memory blocks of around 0.02% of the image size to be used.

Another important advantage of the proposed algorithm is the linearity of its running time with respect to the number of pixels of the image. Fig. 12 plots running time as a function of the image size. Image sizes were varied rescaling the original images. The linearity of running time is apparent for both test images. The plot corresponding to the "Baboon" image has a lower slope than the plot corresponding to "Peppers" because of the presence of many more pixels situated in non-minima plateaus. These pixels require extra time to be processed, thus increasing the running time.

5. Conclusions

The watershed transform obtained by means of rain simulation is a good tool to segment objects in digital images. The speed achieved by the actual algorithms, the accuracy of the results, and the simplifications introduced in the post processing calculation, increase the importance of this tool.

There are some previous works that obtain accurate results for all kinds of images, with low computational costs. To date, the fastest was developed by Han Sun et al. [14]. This work has been improved to manage the memory more efficiently and to carry out complex operations with the minimum possible running time.

Comparing the improved *SYR* algorithm with the proposed one, we can conclude that an improvement of around 31% is obtained, averaging over various image sizes.

Another important aspect is that, as with other arrowing algorithms, the new algorithm achieves linear running time with respect to the size of the input images.

6. Summary

This paper has presented an overview of the construction of the watershed transform of an image and a detailed view of the rain simulation technique. Following this methodology, a new algorithm has been developed yielding the same results as the existing ones, but improving running time by around 31% for various images of various sizes.

To obtain these improvements, our algorithm reduces as much as possible the number of neighbouring operations, which is the most expensive computation in the context of these techniques, as well as the number of scans performed over the original image. The algorithm uses only four queues of reduced size and a single output matrix, which is also used to store the intermediate results, with the same size as the input image.

Thus, we can conclude that the presented algorithm performs better than the best existing ones, in both execution time and allocated memory size, and has at least the same efficiency.

Acknowledgements

This research was carried out under Grants: TIC2003-08956-C02-00 and TIC2002-2273 from the *Ministry of Science and Technology*, and AP2001-1278 from the *Ministry of Education* of Spain. The authors would like to thank the anonymous reviewers for their valuable comments.

References

[1] A. Bleau, L.J. Leon, Watershed-based segmentation and region merging, Comput. Vision Image Understanding 77 (3) (2000) 317–370.

[2] R.C. Gonzalez, R.E. Woods, S.L. Eddins, Segmentation using the watershed transform, in: R.C. Gonzalez, R.E. Woods, S.L. Eddins (Eds.), Digital Image Processing Using MATLAB, Pearson Prentice-Hall, NJ, USA, 2004, pp. 417–425.

[3] A. Bleau, J. De Guise, R. LeBlanc, A new set of fast algorithms for mathematical morphology: I-Idempotent geodesic transforms, CVGIP: Image Understanding 56 (2) (1992) 178–209.

[4] S. Beucher, C. Lantuéjoul, Use of watersheds in contour detection, in: Proceedings of the International Workshop on Image Processing: Real-Time Edge and Motion Detection/Estimation, vol. 132, September 1979, pp. 2.1–2.12.

[5] S. Beucher, The watershed transformation applied to image segmentation, Scanning Microsc. 6 (1992) 299–314.

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

[7] C. Rambabu, I. Chakrabarti, D. Ghosh, An efficient watershed transform computation method, in: Proceeding of IEEE ICICS-PCM 2003, vol. 2, December 2003, pp. 792–796.

[8] S.Y. Chien, Y.W. Huang, S.Y. Ma, L.G. Chen, Predictive watershed for image sequences segmentation, in: Proceedings of IEEE ICASSP'02, vol. 3, Orlando, FL, USA, May 2002, pp. 3196–3199.

[9] S.Y. Chien, Y.W. Huang, L.G. Chen, Predictive watershed: a fast watershed algorithm for video segmentation, IEEE Trans. Circuits Systems Video Technol. 13 (5) (2003) 453–461.

[10] J.P. Thiran, V. Warscotte, B. Macq, A queue-based region growing algorithm for accurate segmentation of multi-dimensional digital images, Signal Process. 60 (1) (1997) 1–10.

[11] C.J. Kuo, S.F. Odeh, M.C. Huang, Image segmentation with improved watershed algorithm and its FPGA implementation, in: Proceedings of IEEE ISCAS 2001, vol. 2, Sydney, Australia, May 2001, pp. 753–756.

[12] A. Bieniek, A.N. Moga, An efficient watershed algorithm based on connected components, Pattern Recognition 33 (6) (2000) 907–916.

[13] V. Grau, A.U.J. Mewes, M. Alcañiz, R. Kikinis, S.K. Warfield, Improved watershed transform for medical image segmentation using prior information, IEEE Trans. Med. Imaging 23 (4) (2004) 447–458.

[14] H. Sun, J. Yang, M. Ren, A fast watershed algorithm based on chain code and application in image segmentation, Pattern Recognition Lett. 26 (9) (2005) 1266–1274.

[15] C. Rambabu, T.S. Rathore, I. Chakrabarti, A new watershed algorithm based on hillclimbing technique for image segmentation, in: Proceedings of TENCON 2003, vol. 4, Bangalore, India, October 2003, pp. 1404–1408.

[16] A.N. Moga, M. Gabbouj, Parallel image component labeling with watershed transformation, IEEE Trans. Pattern Anal. Mach. Intell. 19 (5) (1997) 441–450.

[17] J.B.T.M. Roerdink, A. Meijster, The watershed transform: definitions, algorithms and parallelization strategies, Fundamenta Informaticae 41 (2000) 187–228.

[18] S. Beucher, F. Meyer, The morphological approach to segmentation: the watershed transformation, in: E. Dougherty (Ed.), Mathematical Morphology in Image Processing, Marcel Dekker, New York, USA, 1993, pp. 433–481.

[19] C. Rambabu, I. Chakrabarti, A. Mahanta, Flooding-based watershed algorithm and its prototype hardware architecture, IEE Proc. Vision, Image Signal Process. 151 (3) (2004) 224–234.

[20] G. Bueno, O. Musse, F. Heitz, J.P. Armspach, Three-dimensional segmentation of anatomical structures in MR images on large data bases, Magn. Reson. Imaging 19 (1) (2001) 73–88.

[21] L. Patino, Fuzzy relations applied to minimize over segmentation in watershed algorithms, Pattern Recognition Lett. 26 (6) (2005) 819–828.

[22] S. Eom, S. Chang, B. Ahn, Watershed-based region merging using conflicting regions, in: Proceedings of ICIP 2002, vol. 2, September 2002, pp. 781–784.

[23] H. Zhu, O. Basir, F. Karray, Fuzzy integral based region merging for watershed image segmentation, in: Proceedings of FUZZ-IEEE 2001, vol. 1, December 2001, pp. 27–30.
[24] S.E. Hernandez, K.E. Barner, Joint region merging criteria for watershed-based image segmentation, in: Proceedings of IEEE ICIP 2000, vol. 2, September 2000, pp. 108–111.
[25] K. Haris, S.N. Efstratiadis, N. Maglaveras, A.K. Katsaggelos, Hybrid image segmentation using watersheds and fast region merging, IEEE Trans. Image Process. 7 (12) (1998) 1684–1699.
[26] A. Bleau, J. De Guise, R. LeBlanc, A new set of fast algorithms for mathematical morphology: II-Identification of topographic features on grayscale images, CVGIP: Image Understanding 56 (2) (1992) 210–229.
[27] D.F. Shen, M.T. Huang, A watershed-based image segmentation using JND property, 3 (2003) 377–380.