Exercise 01e: Rainfall Watershed Algorithm

Rodrigo Pueblas

12/04/2020

In this exercise, we are going to try to recreate pseudo-code that computes a faster watershed implementation, using the approach from the paper "An Improved Fast Watershed Algorithm based on finding the Shortest Paths with Breadth First Search" (Suphalakshmi and Anandhakumar, 2012).

The proposed approach is divided in two steps:

1. Create an arrow representation of the image, in which every pixel points to their minimum neighbour.
2. Label the regional minimum and propagate the labels along the image via arrowing.

Below can be found a pseudocode implementation. The Python implemented approach gave the following result:

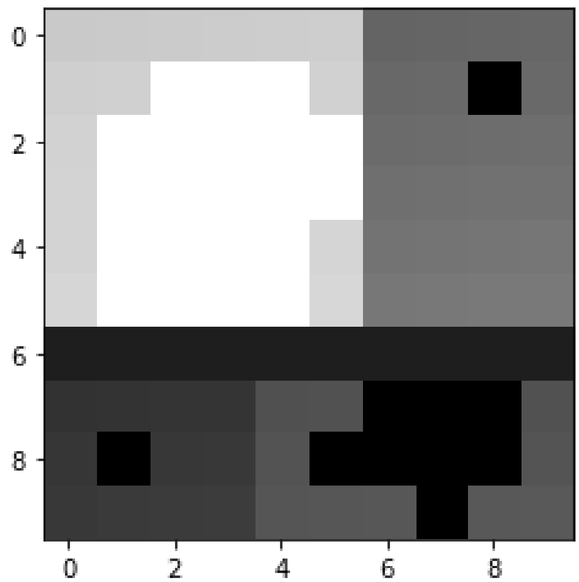

Figure 1: Input image for the algorithm

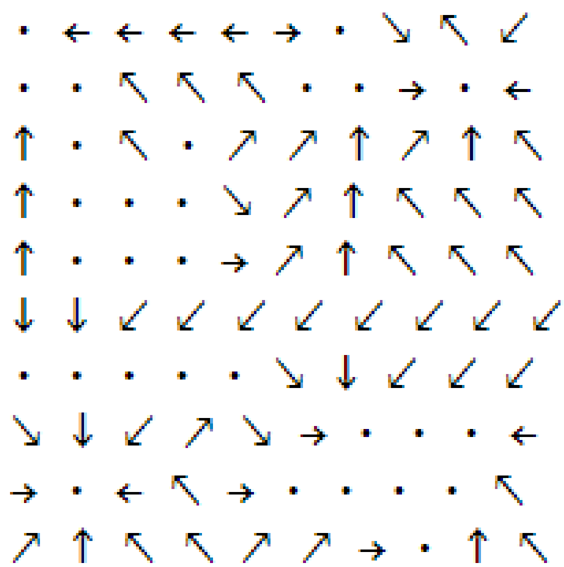

Figure 2: Arrow representation for the result of the algorithm

Algorithm 1 Main

```
procedure ARROWING
    image ← input image
    pointers ← zeros(size(image))
    labels ← array<str>[size(images)]
    for each pixel ∈ image do
        if pixel == max(neighbours(image, pixel)) and pixel == min(neighbours(image, pixel)) then
            // pixel is plateau
            pointers[pixel] ← bfs(image, pixel)
            labels[pixel] ← ”plateau”
        else
            if pixel ≤ min(neighbours(image, pixel)) then
                // pixel is local minimum
                pointers[pixel] ← -1
                labels[pixel] ← ”minimum”
            else
                // pixel is normal
                pointers[pixel] ← getMinDirection(image, pixel)
                labels[pixel] ← ”normal”
            end if
        end if
    end for
end procedure
```

Algorithm 2 GetMinDirection

```
procedure getMinDirection(image, pixel)
    if image[pixel[0],pixel[0]+1]==min(neighbours(image, pixel)) then
        return 0
    end if
    if image[pixel[0]-1,pixel[0]+1]==min(neighbours(image, pixel)) then
        return 1
    end if
    if image[pixel[0]-1,pixel[0]]==min(neighbours(image, pixel)) then
        return 2
    end if
    if image[pixel[0]-1,pixel[0]-1]==min(neighbours(image, pixel)) then
        return 3
    end if
    if image[pixel[0],pixel[0]-1]==min(neighbours(image, pixel)) then
        return 4
    end if
    if image[pixel[0]+1,pixel[0]-1]==min(neighbours(image, pixel)) then
        return 5
    end if
    if image[pixel[0]+1,pixel[0]]==min(neighbours(image, pixel)) then
        return 6
    end if
    if image[pixel[0]+1,pixel[0]+1]==min(neighbours(image, pixel)) then
        return 7
    end if
end procedure
```

Algorithm 3 BFS

```
procedure BFS(IMAGE, PIXEL)
    visited ← zeros(size(image))
    queue ← new queue()
    queue.append(pixel)
    parent ← new dict()
    queue.enqueue(neighbours(image, pixel))
    while queue.size() ¿ 0 do
        pair t ← queue.dequeue()
        if !visited[t] then
            visited[t] ← 1
            if image[t[0],t[1]] ¡ image[pixel] then
                return backtrace(parent, pixel, t)
            end if
            for int i = -1; i ¡ 2; i++ do
                for int j = -1; j ¡ 2; j++ do
                    if !(i == 0 and j == 0) and !visited[t[0]+i,t[1]+j] and image[t[0],t[1]] == image[pixel] then
                        parent[adjacent] ← t
                        queue.enqueue(image[t[0]+i,t[1]+j])
                    end if
                end for
            end for
        end if
    end while
    return -1
end procedure
```

Algorithm 4 backtrace

```
procedure BACKTRACE(PARENT, START, END))
    path ← list(end)
    while path[-1] ≠ start do
        path.append(parent[path[-1]])
    end while
    path ← path.reverse()
    if path[1][0] == start[0] and path[1][1] == start[0]+1 then
        return 0
    end if
    if path[1][0] == start[0]-1 and path[1][1] == start[0]+1 then
        return 1
    end if
    if path[1][0] == start[0]-1 and path[1][1] == start[0] then
        return 2
    end if
    if path[1][0] == start[0]-1 and path[1][1] == start[0]-1 then
        return 3
    end if
    if path[1][0] == start[0] and path[1][1] == start[0]-1 then
        return 4
    end if
    if path[1][0] == start[0]+1 and path[1][1] == start[0]-1 then
        return 5
    end if
    if path[1][0] == start[0]+1 and path[1][1] == start[0] then
        return 6
    end if
    if path[1][0] == start[0]+1 and path[1][1] == start[0]+1 then
        return 7
    end if
end procedure
```