Automatic Watershed Deliniation

Modelling for Science and Engineering

UAB

Francesc Badia Roca

tutor

Lluís Pesquer

September 1, 2014

Abstract

This work presents a new method to extract the drainage basins from a digital elevation model (DEM). It is based on the O'Callaghan and Mark(1984) approach, which uses a simulation of water flow from the point of view of the graphs and constructing paths with the direction of the water. The present version has two importatnt advantages. On one hand, it works on unprocessed DEMs avoiding the problems caused by creating new pits and flats when cleaning the DEM. On the other hand, it is faster than other versions due to the treatment of the whole DEM at the same time, and avoiding the iterative processes that take lots of time of execution.

Index

Introduction

Goal

The main goal of this work is to create a tool which draws the hydrological basins, given an elevation map of a particular zone.

Motivation

The idea of this work is to provide this tool to MiraMon, which is a GIS (Geographic Information System) developed cooperatively and managed from the CREAF, based in UAB.

GIS and Hydrological Modelling

GIS

A Geographic Information System (GIS) is a computer system designed to capture, store, manipulate, analyze, manage and present all types of geographical data. These systems are an aggregation of tools of visualization, manipulation and analysis of all types of geographical data.

Hydrological Modelling

In order to know how the water behaves in a certain area, we must know many aspects of the geography of the terrain, such as the weather, the underground flows, the porosity of the land at each point... One of the most important aspects of hydrology is the terrain analysis, since water always flows down a slope. So an interesting analysis that GIS is expected to make is to show the hydrological basins of an area from the elevation map.

Standard Approaches

The extraction of the river networks from a DEM (Digital Elevation Model) can be done by different methods. Peucker and Douglas(1975), Douglas (1986), Tribe(1992) determine ridge and valley lines by topographic evaluation. Meisels etal.(1995) propose an original solution based on performing a skeletonization process on the set of elevations of the DEM. But by far the most widely used approach is based on the work of O'Callaghan andMark(1984) that uses a simulation of water flow over terrain to extract the drainage information.

The proposed method in this project is based on this last approach with some changes, done in order to make it more simple and efficient.

As Rueda mentions in the article where he explains an algorithm to extract the river network without preprocessing the DEM, many algorithms fail when assigning the flow direction in the flat areas. He also comments that when some algorithms process the DEM in order to remove pits, new flat areas are generated, which have to be treated again.

The Problem

What is a watershed?

A watershed is a set of slopes inclined towards the same watercourse. Watersheds are always physically delineated by the area upstream from a given outlet point. This generally means that for a stream network, the contributing area upstream reaches a ridge line. Ridge lines separate watersheds from each other.

The strandard processes for delineating watersheds from DEMs are generally a list of steps involving the slope at each point of the map, the flow direction, the flow accumulation and the delineation of streams. At the end the identification of pour points under a certain criterion, and finally the delineation of the watershed.

All this concepts will be explained on the Model section.

Most GIS platforms that have Watershed Delineator use a paramaeter which is the basin minimum size, meaning that the result will be a set of watershed basins and the smaller one will have this number of pixels or points inside. As it will be shown later, the process goes from delineating small basins to bigger basins by a process of joining the first ones. This widely used criterion is not used in the present project and the biggest basins are delineated.

The Model

The nature of the Digital Elevation Data

A DEM may be defined as any numeric or digital representation of the elevations of all or part of a planetary surface, given as a function of geographic location. However, in the present project the attention is focused on the most commonly used data structure for DEMs: the regular square grid. In such a grid, elevations are available as a matrix of points equally spaced in two orthogonal directions.

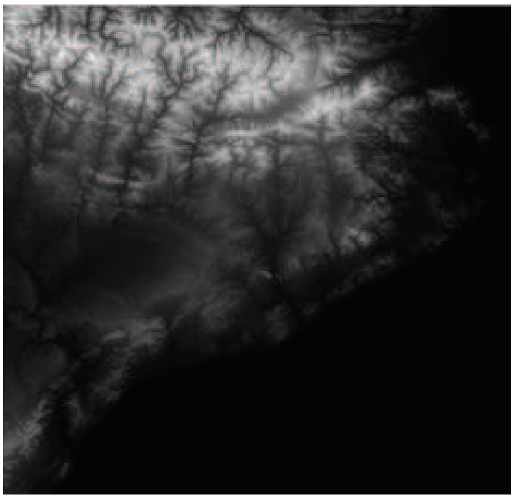


fig 1.

fig 2.

1:3600000

Definitions, Concepts and Assumptions

We begin with the matrix ALTITUDE, containing m x n points, arranged in a grid with n columns and m rows. At each point, we have the elevation of that point in units above some datum. A pit is a cell with non lower-altitude neighbours. A starting point (ridge point), is defined as a cell which has no drainage input. It is assumed that for each cell which is not a pit, the water flows to one and exactly one direction.

Flow Direction

We can define a directed graph whose vertices are the grid points of ALTITUDE and whose edges are the flow direction going down slope. Since at each cell the direction is to one neighbour or none, the graph will be at this point a forest of subtrees.

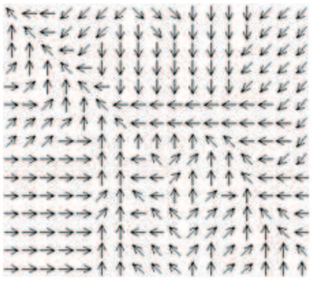


fig 3.

The topography of the flow direction can be represented by a matrix called FLOWDIRECTION. The codification that is used in this approach is generally known in the literature as D8, although a more precise designation is SFD8 (Single Flow Direction chosen from 8 options).

This codification gives at each cell a value for the direction:

↖	↑	↗
←		→
↙	↓	↘

⟷

32	64	128
16		1
8	4	2

Pits Direction

- **Pure Pits** cells with the lowest altitude of the neighbours and itself and, more than that, with the directions of the eight neighbours pointing to the cell. In this case the direction value is set to 0.
- **Single Pits** cells with the lowest altitude of the neighbours and itself, but with at least one neighbour poiting outside, so, to a cell with lower altitude than the pit. In this cases, we assign 3 to the direction and set the flow direction of the pit to the cell with lower altitude than this one.

The reason for distinguishing this two sorts of pits is that in the model the step of cleaning the DEM is avoided. This step is widely used, even in the O'Callaghan andMark(1984) model, but carries some problems to the following steps of the current model.

The pits information is represented in the FLOWDIRECTION matrix, the Pure Pits are assigned to 0, and the Single Pits to 3.

Basin identification

This is an identificator that shows if two cells are in the same basin or subtree in the graph with the direction of the flow of water going down. The Basin identificator is represented in the BASINID matrix.

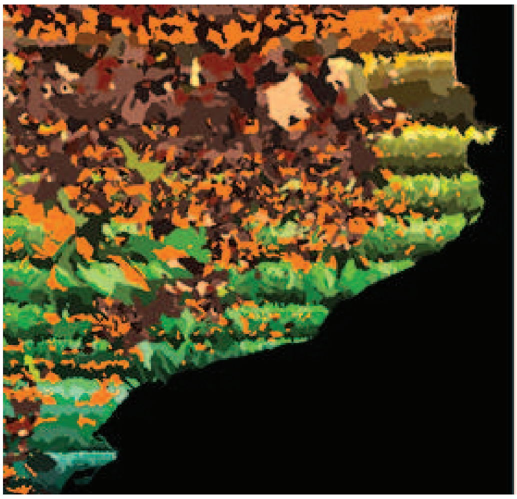


fig 4.

At this point there are a lot of small basins corresponding to the subtrees and ending at the pits. Now, the process of joning this small basins in the right way is started. This is equivalent to connect the pits to the correct cell, so to the pour points of each basin.

Basin boundaries

When a basin identificator is assigned to each cell, the boundaries of this basins are defined to find the pour point of each basin, so the water will flow through the cell of the boundary with least altitude in case of flooding. This information is represented in the BOUNDARY matrix and the values at each cell are 1 if it is boundary or 0, if not.

Pour Points Inside

A pour point inside is a point of a particular basin which is on the boundary of the basin and is the one of the boundary with the least altitude. This information is represented in the PPOINT matrix and at each cell it indicates the pour point inside of the basin that the cell belongs to.

Pour Points Outside

A pour point outside is a point related with the basin, and it is a cell in contact with the boundary of the basin but by the outside part of the boundary, and it is the point with that property which has the minimum altitude. In other words, it is the cell that is out of the basin but is in touch with it and has the lowest altitude. This information is represented in the PPOINTOUT matrix and at each cell it indicates the pour point outside of the basin that the cell belongs to.

Flow Accumulation

The flow accumulation at each cell is the number of cells that a cell has upslope in the opposite direction of the water flow.

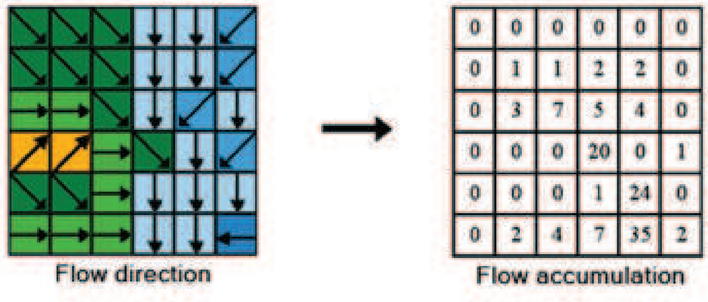


fig 5.

Process to extract the watershed basins

The whole process is a list of nine steps

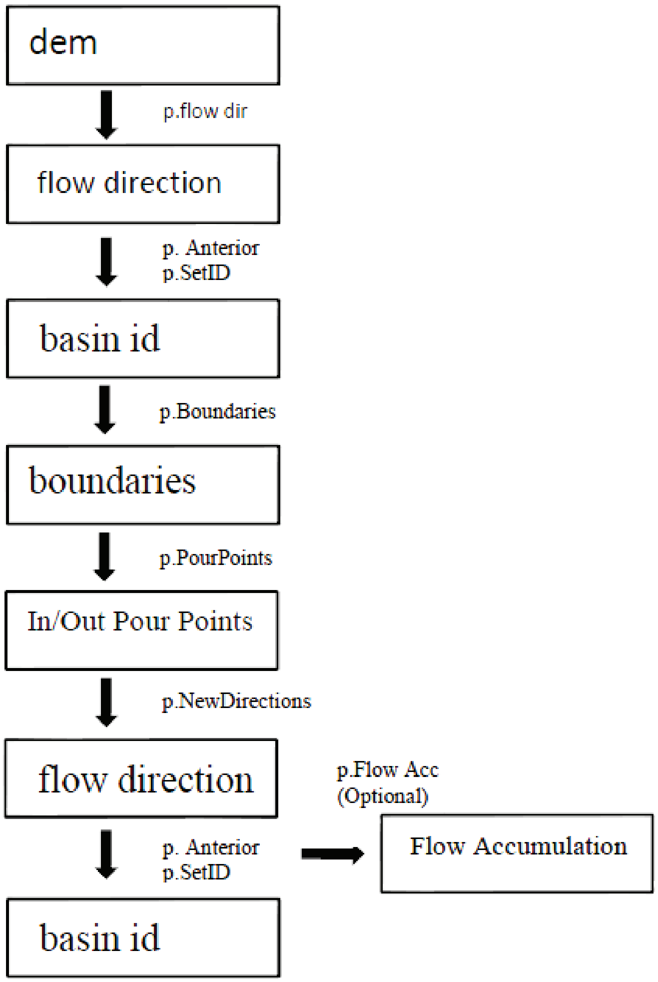


fig 6.

The graph treatment of all the required information is a set of layers represented by matrixes, and with each specific information.

All the matrixes are of the form:

$$\begin{pmatrix} a_0 & \cdots & \cdots & a_{n-1} \\ a_n & \cdots & \cdots & a_{2n-1} \\ \vdots & \ddots & \ddots & \vdots \\ a_{(m-1)n} & \cdots & \cdots & a_{mn-1} \end{pmatrix}$$

The graph has two kinds of elements: cells and edges.

- The cells informations are related to the situation of the cell:
  - altitude,

  - basin id,
  - boundary,
  - pour points,
  - flow accumulation.
- The edge informations are the informations related to the directions:
  - flow direction, the direction of the water going down slope
  - anteriors, the list of cells which the flow direction point to such a cell.

  Every cell has, at the most, one "next" in the direction of the water.

  A cell can have from none previous to a lot of them.

p.flow Dir

Given the DEM, so the altitude matrix, it is scanned and assigned to the FLOWDIRECTION matrix, with these criteria:

- The border of the map are assigned to -1
- The Sea points and the NODATA points are assigned to -1
- The pits are assigned to 0
- The other current points are assigned with the SFD8 code:
  - This codification gives at each cell a value for the direction:

$$\begin{array}{|c|c|c|}\hline \nwarrow & \uparrow & \nearrow \\ \hline \leftarrow & & \rightarrow \\ \hline \swarrow & \downarrow & \searrow \\ \hline \end{array} \longleftrightarrow \begin{array}{|c|c|c|}\hline 32 & 64 & 128 \\ \hline 16 & & 1 \\ \hline 8 & 4 & 2 \\ \hline \end{array}$$

p.Anterior

This process uses an auxiliar layer which is the number of previous cells that has each cell.
After knowing how many previous cells has each cell, if it has some, a list of positions with the previous cells is stored.

1↙	2↘	3↓
4↘	5↘	6↓
7→	8→	9→

then $\left\{ \begin{array}{c} ant(6) = \{2,3\} \\ ant(8) = \{4,7\} \\ ant(9) = \{5,6,8\} \end{array} \right\}$

and $\left\{ ant(1) = ant(2) = ant(3) = ant(4) = ant(5) = ant(7) = \phi \right\}$

p.SetID

This process assigns the same number to all the cells that are in the connected component of the graph, for this process both the flow direction and the anterior direction are used.

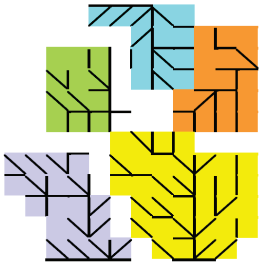


fig 7.

Each diferent colour represents a different basin

p.Boundaries

A matrix is constructed indicating if a cell is in the boundary of a basin or not.

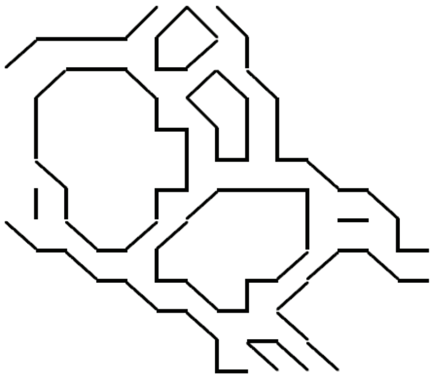


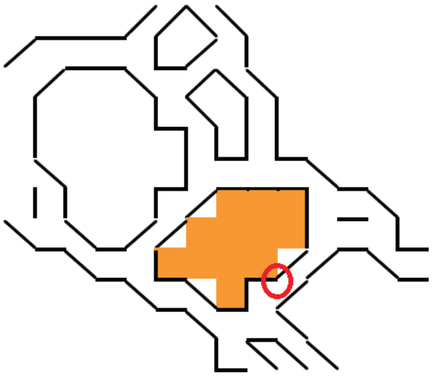
fig 8.

p.PourPoints

Once the boudaries are bounded:

- Inside Pour Point: the lowest altitude point on the boundary is selected.
- Outside Pour Point: the lowest altitude point of the outside neighbours of the boundary is selected.

These two points are the same for all the points of the basin.



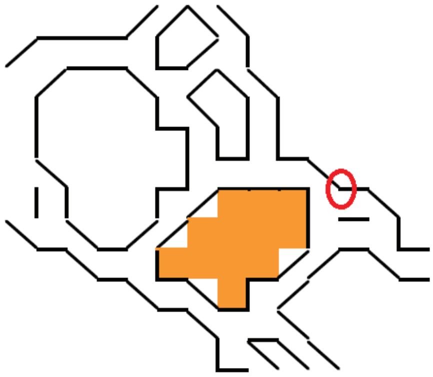


fig 9.
fig 10.

p.NewDirections

This is one of the most important steps. The small basins created until here have one pit or end each one, which is the lowest point of the basin. So, now the flow direction of this pit will be assigned to a point of another basin in the correct way.

The general case, the algorithm selects four points:

1. the pour point inside,
2. the lowest outside point in touch with the pour point inside,
3. the pour point outside,
4. the lowest inside point in touch with the pour point outside.

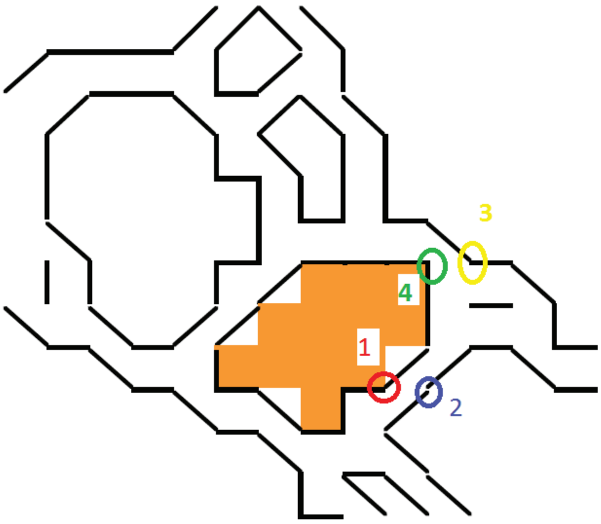


fig11.

If the pit of the basin is a, then:

FlowDir(a) = 2, if altitude(2) $\geq altitude(4)$
FlowDir(a) = 3, if altitude(2) < altitude(4)

Another especial case is when the terrain is flat in a down zone: lakes or wide rivers.
In these cases, the flat zone is scanned until the first lower cell is found.

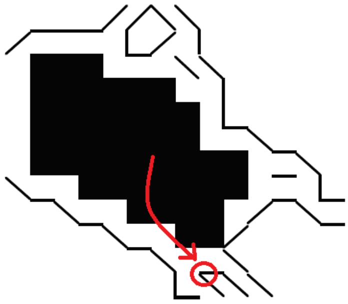


fig 12.

p.Flow Accumulation

This step is optional and creates a layer with the flow accumulation at each cell. This layer is calculated by using the edge information, so both directions: down slope (flow direction) and up slope (previous).

Computing Issues

The result tool is a program from a source code written in C.

The structure of the program is a main function that has the list of calls to the subprocesses that correspond to the list of processes shown before:

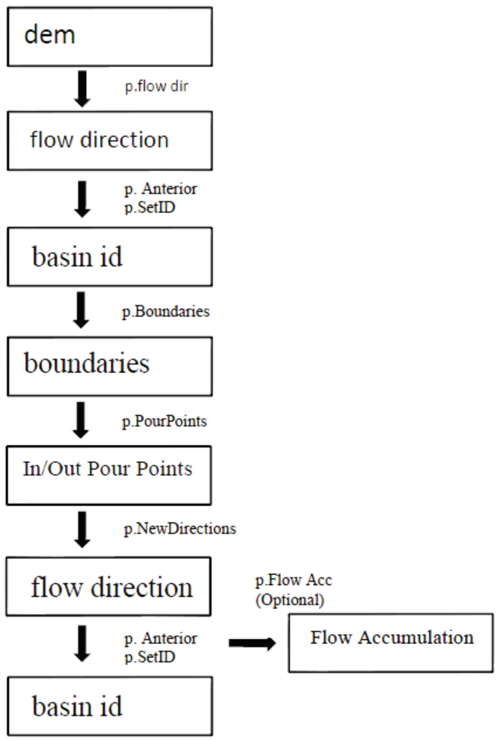


fig 13.

Performance general aspects

The program uses the idea of the graph, so it is moving through the cells using the directions of the flowing water, both up and down. This fact implies that we may need the information of the whole graph (map) at any moment.

There are tools from common approaches that read the maps from the files by scanning groups of three or a certain number of rows, process these rows and its cells and store againg in files. This option forces it to do a lot of iterations to complete certain processes, due to any change at a particular row may have an effect to both the previous and posterior rows.

In consequence, to avoid the effects of this kind of issues, the program has all the information during all the performance. This also carries serious issues: memory and time.

Even choosing this way, there are two options of implementing the program:

- Storing all the information on the dynamic memory of the computer, so the RAM.
  This option has the top of the memory of the computer. Even when the volume increases below the top memory, the performing time also increases, and very quickly.

- Reading and writing continuously from and to files.
  This is the most stable version, since it has no top of size for the map.

This work uses dynamic memory and has a good performance for small maps. One of the next steps could be to adapt the use of dynamic memory to the static one (storing in files) in order to improve the capability of the tool.

Key features

Flow Direction

DEM is treated without removing the pits, so at the first time there are a lot of pits. One way to avoid it is the strategy commented before.

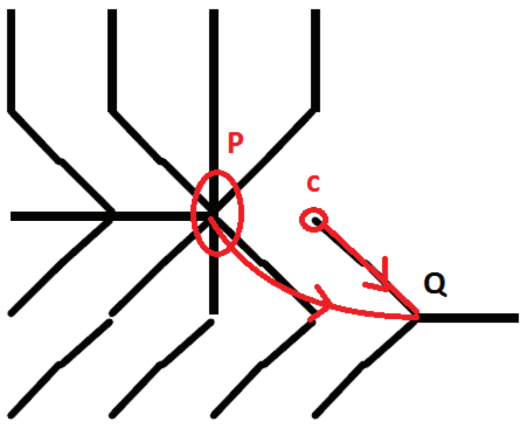


fig 14.

If there is a pit P, but this cell has a neighbour that flows to the cell Q, this will obviously mean that Q has lower altitude than P, so the next of P is set to Q. With this trick, lots of single pits are avoided.

Set Basin Id and Flow Accumulation

These two processes are the strongest reasons why we need all the information at anytime.

To set the basin identificator, a flow simulation starts on a ridge point, goes down slope and, after each step going down, all the id for the cells up slope have to be set

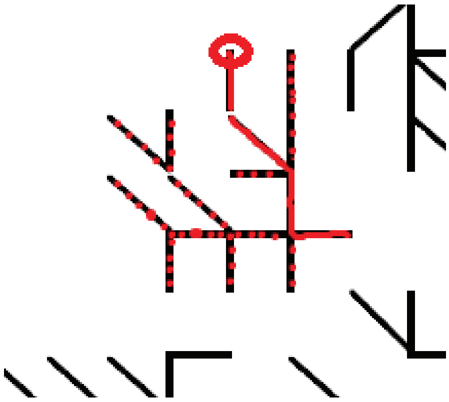


fig 15.

The algorithm uses a recursive function that is called iteratively from the new previous to a ridge point.

A similar algorithm is used to compute the flow accumulation, but instead of seting the same value, it keeps on adding the accumulation value until the ridge points have accumulation 0.

Storing the previous Ones

The map has m rows and n columns, so it has mxn elements. Eventually this magnitude can be a big number for the computer, in terms of memory.

The tool uses several layers, each one of mxn elements. This has a big cost but it can be acceptable.

But the information of the previous ones requires a special treatment:

Each cell may have 8 anteriors at the most, but with the process of removing single pits new anteriors are created, so this number of directions may increase. And even more, at the final process of merging the basins, new connections are created, and this quantity can reach a big number.

The Valence is the number of connections that has one cell in a graph.

If the maximum valence mxn times is stored, the computer probably runs out of memory. And even more, the maximum valance is not known a priori. So a good way to treat this memory will be the dinamicall storing.

Taking into account that there may be a lot of cells that do not have previous, no memory will be required for these cells.

The strategy is as it follows:

- Create a layer that has the number of previous cells at each point.
- Create a layer that indicates -1 if the cell has not previous, and from the elements which hve previous ordered from 0 to K-1 being K the number of cells that has previous.
  This layer has this shape:
  $\text{index} = \{-1, -1, -1, 0, -1, -1, -1, -1, -1, -1, 1, -1, -1, 2, -1, -1, -1, 3, -1, -1, -1, 4, -1, -1, -1.....\}$

With this strategy we can store only K lists of num_previous(i) elements for each one.

New Directions

The process is explained before. But there is a possible improvement than can be applicable to this step.

In the cases of lakes and wide rivers, absolutely flat zones, here is proposed an algorithm which scan all the flat zone until a lower cell is found.

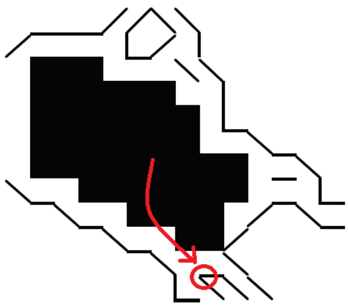


fig 16.

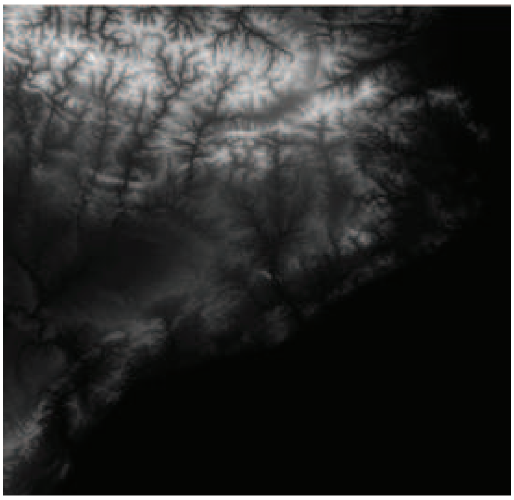
The GRASS GIS system uses a variant of the A* algorithm to find the exit or the pour points for these zones. The A* algorithm is a best-search algorithm and it is widely used in graphs to find optimal paths.

Future Steps - Improvements

- Adapt the whole memory storage to static memory. At least the memory of the previous cells, which is what increases the most, both in time and in memory.
- Apply the A* algorithm to find the paths to the pour points of the flat zones.
- Clean the final basins that point outside the map or to the sea, because in these cases these basins are never more joined to another, so the limits of the map are full of very small basins.
- More in general, find a way of computing the whole process without the need to have the information of the previous cells, which has a high cost.

Results

Resutls are shown for the Catalunya DEM.
The top left picture is a visualization of the DEM.
The top right is a visualization of the direction flow.
The bottom left picture is the first basin identification.
The bottom right picture is the final result, showing the whole Hydrologic Basins.



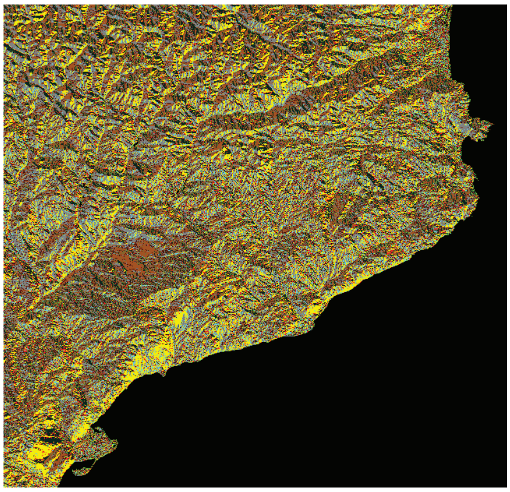
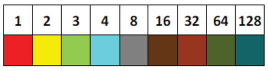
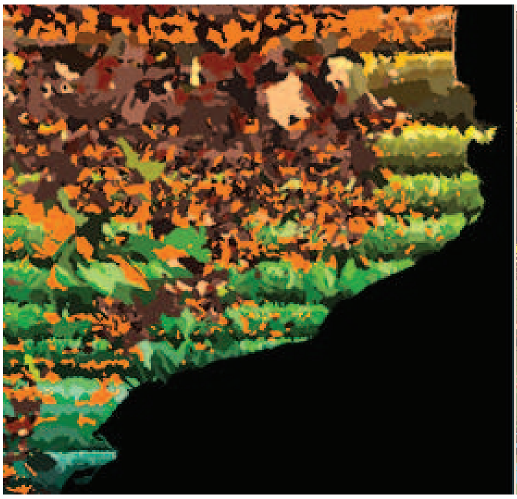


fig 17.
fig 18.

1:3600000 





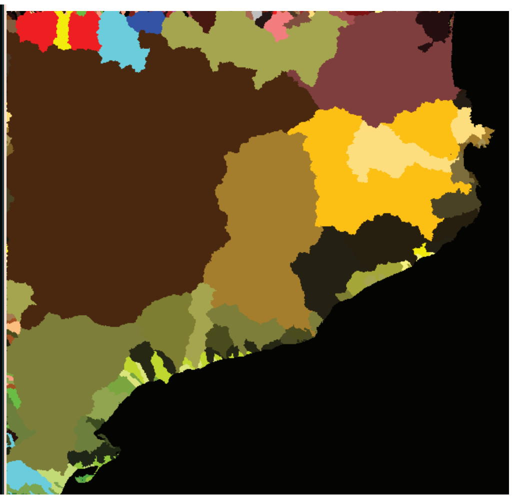


fig 19.
fig 20.
1:3600000 Random colorized.

The dimension of this DEM (picture) is 1386 columns and 1335 rows.
The resolution of the map is 200mx200m

Here there is the time of execution of three different resolutions.

columns	rows	total pixels	time of execution
1080	970	1047600	1.7 seconds
1386	1335	1850310	4,7 seconds
3080	2966	9135280	59,5 seconcs

Idrisi, with the second case: 1386x1335 DEM, takes 32 seconds.

Conclusions

- This project opens a new way on the hydrological analysis of DEMs, which is treating the whole map at the same time and applying graph teory techniques directly.
  Although the tool takes a lot of memory when the size of the DEM is big, new ways and ideas are indicated in order to improve the tool in this sense.

- The project starts from the ideas shown on O'Callaghan, J.F.,Mark,D.M. approach. However it is focused in the creation of a new tool and the development of the source code from the very begining.

- Other models allow to get the resultant basins with a parameter which is a threshold of basin minimum size. In the present project, it could be possible to introduce this kind of parameter, although at the moment, the result shows the biggest basins that the terrain has itself.

- The resultant tool has a good performance, even though there are also some ways of improve it. The changes to make the tool better, as it has been said before, can be done from the point of view of both modelling and computing.

References

- MiraMón. Sistema d'Informació Geogràfica (SIG) i software de Teledetecció
  Centre de Recerca Ecològica i Aplicacions Forestals
  http://www.creaf.uab.es/miramon/index_ca.htm
- O'Callaghan, J.F.,Mark,D.M.,1984.
  The extraction of drainage networks from digital elevationdata.
  Computer Vision, Graphics, and Image Processing
- Antonio Rueda, JoséM.Noguera, CarmenMartínez-Cruz
  A flooding algorithm for extracting drainage networks from unprocessed digital elevation models
- Douglas, D.H.,1986.
  Experiments to locate ridges and channels to create a new type of digital elevation model. Cartographica 23 , 29 – 61.
- Peucker, T.K.,Douglas,D.H.,1975.
  Detection of surface-specific points by local parallel processing of discrete terrain elevation data. Computer Graphics and Image Processing 4, 375 – 387.
- Tribe,A.,1992.Automated recognition of valley lines and drainage networks from grid digital elevation models: a review and a new method. Journal of Hydrology 139 , 263–293 .
- Meisels, A.,Raizman,S.,Karnieli,A.,1995.
  Skeletonizing a DEM into a drainage network . Computers & Geosciences 21 , 187–196.

List of Figures

Annex: Source Code

The Source Code is stored in two files:

- Watershed.final.v6.c
  The main program which stores all the memory, reads the DEM and calls the functions required for the process.

- Watershed.final.v6.h
  The library with the functions required for the process.
  All the functions are gruped in three categories: Modelling functions, Technical Functions and Reading and Writing Functions.

```c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "watershed.final.v6.h"
    int n = 1386; //columns  Catalunya
    int m = 1335; //rows Catalunya

//    int n = 7040; //columns Andorra
//    int m = 5540; //rows    Andorra

int main()
{
    float *altitude;
    short *dir,*isboundary,*nprev;
    int *next, **ant,*ppin,*ppout,*index,*id,*flowacc,i,j,k;

    FILE *fin,*fout;

    //catalunya
    fin  = fopen("MDE200m_ICC_Aster.img","rb");
    fout = fopen("Catalunya_watershed.img","wb");

    //andorra
//  fin  = fopen("MDE_Andorra.img","rb");
//  fout = fopen("Andorra_watershed.img","wb");

    altitude     = ( float* )malloc((m*n)*sizeof( float ));
    dir          = ( short* )malloc((m*n)*sizeof( short ));
    isboundary   = ( short* )malloc((m*n)*sizeof( short ));
    next         = ( int*   )malloc((m*n)*sizeof( int   ));
    ppin         = ( int*   )malloc((m*n)*sizeof( int   ));
    ppout        = ( int*   )malloc((m*n)*sizeof( int   ));
    nprev        = ( short* )malloc((m*n)*sizeof( short ));
    index        = ( int *)malloc((m*n)*sizeof(int));
    id           = ( int *)malloc((m*n)*sizeof(int));
    flowacc      = ( int *)malloc((m*n)*sizeof(int));

    //Reading input data
    InitDataBinnary(altitude,fin,m,n);

    //Cleaning input data
    for(i=0;i<m*n;i++) if(altitude[i]<0) altitude[i] = 0;

    //Flow Direction
    Flowdir(altitude,dir,m,n);/*OK dir en el mar o en nodata és 0*/

    //Set Next
    SetNext(dir,next,altitude,m,n);

    //Set Previous
    ant = SetPrev(ant,nprev,next,index,dir,m,n);

    //Set Basin Id
    j = SetID(id,next,ant,index,nprev,altitude,m,n);

    //Boundaries
    Boundaries(isboundary,id,m,n);

    //Pour points
    Ppoint(ppin,ppout,altitude,isboundary,next,dir,id,m,n,j);

    //Setting new directions (merging small basin)
    Newdirections(next,ant,dir,altitude,id,ppin,ppout,index,nprev,m,n,j);

    //Set previous
    free(ant);
    ant = NULL;
    ant = SetPrev(ant,nprev,next,index,dir,m,n);

    //Final Basin ID
    j = SetID(id,next,ant,index,nprev,altitude,m,n);

    //Flow Accumulation (Optional)
    //Flowaccum(flowacc,nprev,ant,index,m,n);

    //Write the final layer
    WriteDataBinnary(id,fout,m,n);
}

```

```c

#include <stdio.h>
#include <stdlib.h>

#ifndef LIBRARYHEADERFILE_INCLUDED
#define LIBRARYHEADERFILE_INCLUDED

/*****************************************************************************************/
/* HEADERS *******************************************************************************/
/*****************************************************************************************/

/*modelling functions*********************************************************************/
void    Flowdir       ( float *altitude ,short *dir,int m, int n);
void    SetNext       ( short *dir ,int *next,float *alt,int m,int n);
int**   SetPrev       ( int   **ant ,short *nprev,int *next,int *index,short *dir,int m, int n);
void    Setnprev  ( short *nprev ,int *next,int m,int n);
int     SetID         ( int   *id ,int *next,int **ant,int *index,short *nprev,float *altitude,int m,int n);
void    Completeup    ( int   run ,int *id,int j,int **ant,int *index,short *nprev,int m, int n);
void    Boundaries    ( short *isboundary,int *id,int m,int n);
void    Ppoint        ( int   *ppin ,int *ppout,float *altitude,short *isboundary,int *next,short *dir,int *id,int m,int n,int numreg);
void    Newdirections( int   *next ,int **ant,short *dir,float *altitude,int *id,int *ppin,int *ppout,int *index,short *nprev,int m,int n,int numreg);
void    Flowaccum     ( int *flowacc,short *nprev,int **ant,int *index,int m,int n);
int     Accum         ( int position,int **ant,short *nprev,int *index,int m,int n);

/*technical functions*********************************************************************/
void    PointsNear       ( float *altitude, int i, float *b,int m,int n);
void    PosofFloat       ( float*,int*,int,int,int);
void    Positions        ( int *positions,int pos,int m,int n);
int     Dir              ( int);
int     WhichMin     ( float *b);
int     Isintheborder    ( int i, int m, int n);
void    Setindex     ( int *index,short *nprev,int m,int n);


/*reading and writing functions***********************************************************/
void    InitDataBinnary              ( float *a, FILE *f,int m, int n);
void    WriteMatrixDecimalCSVshort    ( short *a,int m,int n,FILE *f);
void    WriteMatrixDecimalCSVint  ( int *a,int m,int n,FILE *f);
void    WriteDataBinnaryShort         ( short *a,FILE *f,int m, int n);
void    WriteDataBinnary          ( int *a,FILE *f,int m, int n);


/*global variables************************************************************************/
float NODATA = -9999;
int pos[9];
int counter=0;

/*****************************************************************************************/
/* FUNCTIONS *****************************************************************************/
/*****************************************************************************************/

/*modelling functions*********************************************************************/

//matrix dir has at the end the SFD8 codification for directions
void    Flowdir(float *altitude,short *dir,int m, int n)
{
    printf("Flow Direction\n");
    int i,k;
    float *b;
    b = (float*)malloc(9*sizeof(float));
    for(i=0;i<m*n;i++)
    {
        PointsNear(altitude,i,b,m,n);
        k = WhichMin(b);
        dir[i] = Dir(k);
    }
    free(b);
    return;
}

//next is the matrix in which positions has the next cell position
void    SetNext(short *dir,int *next,float *alt,int m,int n)
{
    printf("Next positions\n");
    int i,j,posx;
    float minalt = 10000;

    for(i=0;i<m*n;i++)
    {
```

```
        j=0;
        Positions(pos,i,m,n);
        if(dir[i]==0)
        {
            //if pos == -1 we can  not evaluate dir[pos] -> out of map or NOADATA
            if(pos[0]==-1) j++; else if( dir[pos[0]] == 2   ) j++;
            if(pos[1]==-1) j++; else if( dir[pos[1]] == 4   ) j++;
            if(pos[2]==-1) j++; else if( dir[pos[2]] == 8   ) j++;
            if(pos[3]==-1) j++; else if( dir[pos[3]] == 1   ) j++;
            if(pos[5]==-1) j++; else if( dir[pos[5]] == 16  ) j++;
            if(pos[6]==-1) j++; else if( dir[pos[6]] == 128 ) j++;
            if(pos[7]==-1) j++; else if( dir[pos[7]] == 64  ) j++;
            if(pos[8]==-1) j++; else if( dir[pos[8]] == 32  ) j++;

            if(j==8 || j==0) next[i]=-1;
            else {dir[i] = 3; next[i]=-1;}
        }
        else
        {
            if( dir[i]==1  ) next[i] = pos[5];
            if( dir[i]==2  ) next[i] = pos[8];
            if( dir[i]==4  ) next[i] = pos[7];
            if( dir[i]==8  ) next[i] = pos[6];
            if( dir[i]==16 ) next[i] = pos[3];
            if( dir[i]==32 ) next[i] = pos[0];
            if( dir[i]==64 ) next[i] = pos[1];
            if( dir[i]==128) next[i] = pos[2];

        }
        if(Isintheborder(i,m,n)==1) next[i]=-1;
    }
    for(i=0;i<m*n;i++)
    {
        if(dir[i]==3)
        {
            if(Isintheborder(i,m,n)==1) {dir[i] = 0; next[i]=-1; continue;}
            Positions(pos,i,m,n);
            minalt = 10000;
            j=0;
            posx=-1;
            if(pos[0]!=-1) if( dir[pos[0]] != 2   ) if(alt[next[pos[0]]] <minalt ) {
minalt =alt[next[pos[0]]]; posx = next[pos[0]]; }
            if(pos[1]!=-1) if( dir[pos[1]] != 4   ) if(alt[next[pos[1]]] <minalt ) {
minalt =alt[next[pos[1]]]; posx = next[pos[1]]; }
            if(pos[2]!=-1) if( dir[pos[2]] != 8   ) if(alt[next[pos[2]]] <minalt ) {
minalt =alt[next[pos[2]]]; posx = next[pos[2]]; }
            if(pos[3]!=-1) if( dir[pos[3]] != 1   ) if(alt[next[pos[3]]] <minalt ) {
minalt =alt[next[pos[3]]]; posx = next[pos[3]]; }
            if(pos[5]!=-1) if( dir[pos[5]] != 16  ) if(alt[next[pos[5]]] <minalt ) {
minalt =alt[next[pos[5]]]; posx = next[pos[5]]; }
            if(pos[6]!=-1) if( dir[pos[6]] != 128 ) if(alt[next[pos[6]]] <minalt ) {
minalt =alt[next[pos[6]]]; posx = next[pos[6]]; }
            if(pos[7]!=-1) if( dir[pos[7]] != 64  ) if(alt[next[pos[7]]] <minalt ) {
minalt =alt[next[pos[7]]]; posx = next[pos[7]]; }
            if(pos[8]!=-1) if( dir[pos[8]] != 32  ) if(alt[next[pos[8]]] <minalt ) {
minalt =alt[next[pos[8]]]; posx = next[pos[8]]; }
            next[i] = posx;
        }
    }
    return;
}

//each ant[i] is the i+1-st element that has previous,and will have exactly the number of
//anteriors.
//The translation between positions i and the ones that has previous is through the vector
//index index[i] is -1 if i has not previous, and i is the index[i]-th element with previous
//if i has previous
int**   SetPrev(int **ant,short *nprev,int *next,int *index,short *dir,int m,int n)
{
    int i,j,k,l,maxnprev=0,aux;
    printf("Previous positions\n");
    Setnprev(nprev,next,m,n);
    for(i=0;i<m*n;i++)
    {
        if(nprev[i]>maxnprev) {maxnprev = nprev[i];}
    }
    j=0;
    for(i=0;i<m*n;i++)
    {

        if(nprev[i]>0)
        {
            //if(counter==1) printf("i %d, nprev %d\n",i,nprev[i]);
            if(j==0)
```

```
                {
                    ant = (int**)malloc(1*sizeof(int*));
                    j++;
                    ant[0] = (int*)malloc((1+nprev[i])*sizeof(int));
                    ant[0][0] = i;
                    for(k=0;k<nprev[i];k++) ant[0][k+1] = -1;
                }
                else
                {
                    j++;
                    ant = (int**)realloc(ant,j*sizeof(int*));
                    ant[j-1] = (int*)malloc((1+nprev[i])*sizeof(int));
                    ant[j-1][0] = i;
                    for(k=0;k<nprev[i];k++) ant[j-1][k+1] = -1;
                }
                index[i] = j-1;
            }
            else
                index[i] = -1;
        }
        /*j is the number of cells with at least one previous   */
        for(i=0;i<m*n;i++)
        {
            if(next[i]!=-1 && index[next[i]]!=-1)
            {
                k=0;
                while(ant[index[next[i]]][k]!=-1) {k++;};
                ant[index[next[i]]][k] = i;
            }
        }
        return(ant);
}
//nprev[i] is the number of anterior elements that i has
void    Setnprev(short *nprev,int *next,int m,int n)
{
        int i;
        for(i=0;i<m*n;i++)
        {
            if(Isintheborder(next[i],m,n)==1) continue;
            nprev[next[i]]++;
        }
        return;
}
//id is the layer with the labels of the basin
int     SetID(int *id,int *next,int **ant,int *index,short *nprev,float *altitude,int m,int n)
{
        printf("Set ID\n");
        int i,j,run,aux,*numid;
        j=0;
        for(i=0;i<m*n;i++) id[i] = 0;
        for(i=0;i<m*n;i++)
        {
            if(id[i]!=0) continue; /* if id has been set before */
            if(next[i]==-1 && nprev[i]==0 && (Isintheborder(i,m,n)==1 || altitude[i]==0)) {id[i]=-1;continue;} /* sea or map border */
            if(nprev[i]==0) /*starting point*/
            {
                j++;
                id[i]=j;
                run = i;
                while(Isintheborder(run,m,n)==0 && next[run]!=-1 )
                {
                    run = next[run];
                    Completeup(run,id,j,ant,index,nprev,m,n);
                };
            }
        }
        for(i=0;i<m*n;i++) if(Isintheborder(i,m,n)==1) id[i] = -1;
        numid = (int*)malloc((j+1)*sizeof(int));
        for(i=0;i<=j;i++) numid[i]=i;
        for(i=0;i<m*n;i++)
        {
            if(id[i]!=-1) numid[id[i]]=0;
        }
        aux = 0;
        for(i=0;i<=j;i++) if(numid[i]==0) aux++;
        free(numid);
        printf("%d regions\n",aux);
        return(j);
}
//this function assign an id up slope when two steams join
void    Completeup(int run,int *id,int j,int **ant,int *index,short *nprev,int m,int n)
{
```

```c
    int i;
    if(run==-1) return;
    id[run]=j;
    if(nprev[run]==0) return;
    for(i=0;i<nprev[run];i++)
    {
        if(id[ant[index[run]][i+1]] == j ) continue;
        else {Completeup(ant[index[run]][i+1],id,j,ant,index,nprev,m,n);}
    }
    return;
}
//Creates a new layer showing if each point is on the boundary of the basin or not
void    Boundaries(short *isboundary,int *id,int m,int n)
{
    int i,k;
    printf("Boudaries \n");
    for(i=0;i<m*n;i++)
    {
        Positions(pos,i,m,n);
        for(k=0;k<9;k++)
            if(pos[k]!=-1)
                if(id[pos[k]]!=id[i])
                    isboundary[i] = 1;
    }
    return;
}

//Setting pour points inside and out side for each basin
void    Ppoint(int *ppin,int *ppout,float *altitude,short *isboundary,int *next,short *dir, int *id,int m,int n,int numreg)
{
    printf("Pour Points\n");
    int i,j,k,*hashpp,*hashppout,posx;
    float *altitudein,*altitudeout;
    //struct cell *b,*c;
    hashpp = (int*)malloc((numreg+1)*sizeof(int));
    hashppout = (int*)malloc((numreg+1)*sizeof(int));
    altitudein = (float*)malloc((numreg+1)*sizeof(float));
    altitudeout = (float*)malloc((numreg+1)*sizeof(float));
    for(i=0;i<=numreg;i++) {altitudein[i] =10000; altitudeout[i]=10000;}
    for(i=0;i<m*n;i++)
    {
        if(Isintheborder(i,m,n)==1) continue;
        Positions(pos,i,m,n); /*split in two cases: the normal one and the links*/
        for(k=0;k<9;k++)
        {
            if(pos[k]!=-1 /**/&& Isintheborder(pos[k],m,n)==0)
            {
                if(isboundary[i]==1 && id[pos[k]]==id[i] && altitude[pos[k]]<altitudein[id[i]])
                {
                    altitudein[id[i]] = altitude[pos[k]];
                    hashpp[id[i]] = pos[k];
                }
                if(isboundary[i]==1 && id[pos[k]]!=id[i] && altitude[pos[k]]<altitudeout[id[i]] && next[pos[k]]!=-1) /**/
                {
                    altitudeout[id[i]] = altitude[pos[k]];
                    hashppout[id[i]] = pos[k];
                }
            }
        }
    }
    for(i=0;i<m*n;i++)
    {
        ppin[i] = hashpp[id[i]];
        ppout[i] = hashppout[id[i]];
    }
    /*Correction of the ppout in the cases of lakes and wide rivers the id of this regions are -1 but we
    don't want to connect this zones with the borders and sea points*/
    for(i=0;i<m*n;i++)
    {
        if(next[i]==-1 && altitude[i]>0.001 && Isintheborder(i,m,n)==0)
        {
            Positions(pos,i,m,n);
            /*****/
            for(k=0;k<9;k++)
            {
                if(next[pos[k]]==-1 && ppout[pos[k]]!=ppout[0] )
                {
                    ppout[i] = ppout[pos[k]];
                }
            }
```

```
            if(ppout[i]!=ppout[0]) continue;
            for(k=0;k<9;k++)
            {
                if(ppout[pos[k]]!=ppout[0]  )
                    {ppout[i] = ppout[pos[k]]; printf("ASDF\n"); }
            }
            /*******/
            /*******/
        }
    }

    free(hashpp); free(hashppout);
    free(altitudein); free(altitudeout);
    return;
}
//this function assign new directions to the pits of the basins
void    Newdirections(int *next,int **ant,short *dir,float *altitude,int *id,int *ppin, int *ppout,int *index,short *nprev,int m,int n,int numreg)
{
    counter ++;

    printf("New Directions\n");
    int i,j,k,l,a,b,c,d,aux,posx,posy;
    float minalt;

    for(i=0;i<m*n;i++)
    {
        aux = 0;
        if(next[i]==-1 && altitude[i]>0.0001 && Isintheborder(i,m,n)==0)
        {
            Positions(pos,i,m,n); /*isolated point with this characteristics*/
            for(k=0;k<9;k++)
                if(pos[k]!=-1 && next[pos[k]]==-1)
                    aux++;

            if(aux==0) continue;

            if(aux==1)
            {
                a = ppin[i];
                b = -1;
                Positions(pos,a,m,n);
                minalt =10000;
                for(k=0;k<9;k++)
                    if(pos[k]!=-1 && id[pos[k]]!=-1 && id[pos[k]]!=id[i] && altitude[pos[k]]<minalt)
                    {   b = pos[k]; minalt = altitude[pos[k]];}

                c = ppout[i];
                d = -1;
                Positions(pos,a,m,n);
                minalt =10000;
                for(k=0;k<9;k++)
                    if(pos[k]!=-1 && id[pos[k]]!=-1 && id[pos[k]]==id[a] && altitude[pos[k]]<minalt)
                    {   d = pos[k]; minalt = altitude[pos[k]];}

                if(b!=-1 && d!=-1)
                {

                    if(altitude[b]>altitude[d])
                    {
                        if(altitude[i]>=altitude[c])
                        {
                            next[i] = c;
                            nprev[c]++;
                            continue;
                        }
                    }
                    else
                    {
                        if(altitude[i]>=altitude[b])
                        {
                            next[i]= b;
                            nprev[b]++;
                            continue;
                        }
                    }
                }
            }
            /*searching a lower point when the flat zones*/
            j=i;
            k=0;
            while(altitude[j]>=altitude[i])
```

```
            {
                k++;
                for(l=-k;l<=k;l++)
                {   posx = i-k*n+l; if(posx%n==0 ||posx%n==m-1 ||posx<0|| posx>m*n) continue;
                    if(altitude[posx]==NODATA) continue;
                    if(altitude[posx]<altitude[i]) j = posx;}
                for(l=-k+1;l<=k-1;l++)
                {   posx = i-l*n-k; if(posx%n==0 ||posx%n==m-1|| posx<0|| posx>m*n) continue;
                    if(altitude[posx]==NODATA) continue;
                    if(altitude[posx]<altitude[i]) j = posx;}
                for(l=-k+1;l<=k-1;l++)
                {   posx = i-l*n+k; if(posx%n==0 ||posx%n==m-1|| posx<0|| posx>m*n) continue;
                    if(altitude[posx]==NODATA) continue;
                    if(altitude[posx]<altitude[i]) j = posx;}
                for(l=-k;l<=k;l++)
                {   posx = i+k*n+l; if(posx%n==0 ||posx%n==m-1|| posx<0|| posx>m*n) continue;
                    if(altitude[posx]==NODATA) continue;
                    if(altitude[posx]<altitude[i]) j = posx;}
            };
            next[i] = j ;
            nprev[j]++;
        }
    }

    return;
}
//Flow Accumulation
void    Flowaccum    ( int *flowacc,short *nprev,int **ant,int *index,int m,int n)
{
    int i,j,sum;
    for(i=0;i<m*n;i++) flowacc[i]=0;
    for(i=0;i<m*n;i++)
    {
        sum = 0;
        if(nprev[i]==0) {flowacc[i]=0; continue;}
        else
        {
            if(flowacc[i]!=0) continue;
            for(j=1;j<=nprev[i];j++)
            {
                sum = sum+Accum(ant[index[i]][j],ant,nprev,index,m,n);
            }
            flowacc[i] = sum+nprev[i];
        }
    }
    return;
}
//Recursive function that computes the flow Accumulation up slope
int     Accum(int position,int **ant,short *nprev,int *index,int m,int n)
{
    int j,sum=0;
    if(position==-1) return(0);
    if(nprev[position]==0) return(0);
    else
    {
        for(j=1;j<=nprev[position];j++)
        {
            sum = sum+Accum(ant[index[position]][j],ant,nprev,index,m,n);
        }
        return(sum+nprev[position]);
    }
    return(0);
}

/*technical functions*****************************************************************/

//b is a vector with the altitude of the neighbours of i
//in the cases of out of limits or NODATA, it returns 10000, since a minimum is sought
//we want this points to not take importance
void    PointsNear(float *altitude, int i, float *b,int m,int n)
{
    int *positions;
    short j;

    PosofFloat(altitude,pos,i,m,n);
    for(j=0;j<9;j++)
    {
        if(pos[j]==-1) b[j] = 10000;
        else b[j] = altitude[pos[j]];
    }
```

```
    return;
}

//positions returns -1 when the neighbours are outside the limits or when the altitude
//layer is NODATA in those points
void    PosofFloat(float *alt,int *positions,int pos,int m,int n)
{
    int i;
    Positions(positions,pos,m,n);
    for(i=0;i<9;i++)
    {
        if(positions[i]!=-1) if(alt[positions[i]]==NODATA) positions[i]=-1;
    }
    return;
}
//return the positions of the neighbours of pos
//As before, if the neighbours are outside the limits are set to -1
void    Positions(int *positions,int pos,int m,int n)
{
    int x,y;
    x = pos%n;
    y = (pos -x)/n;
    positions[0] = pos-n-1;
    positions[1] = pos-n;
    positions[2] = pos-n+1;
    positions[3] = pos-1;
    positions[4] = pos;
    positions[5] = pos+1;
    positions[6] = pos+n-1;
    positions[7] = pos+n;
    positions[8] = pos+n+1;

    if(x==0 && y==0)
    {   positions[0] = -1;positions[1] = -1;positions[2] = -1;positions[3] = -1; positions[6] = -1; }
    if(x>0 && x<n-1 && y==0)
    {   positions[0] = -1;positions[1] = -1;positions[2] = -1; }
    if(x==n-1 && y==0)
    {   positions[0] = -1;positions[1] = -1;positions[2] = -1;positions[5] = -1; positions[8] = -1; }
    if(x==0 && y>0 && y<m-1)
    {   positions[0] = -1;positions[3] = -1;positions[6] = -1;    }
    if(x==n-1 && y>0 && y<m-1)
    {   positions[2] = -1;positions[5] = -1;positions[8] = -1;    }
    if(x==0 && y==m-1)
    {   positions[0] = -1;positions[3] = -1;positions[6] = -1;positions[7] = -1; positions[8] = -1; }
    if(x>0 && x<n-1 && y==m-1)
    {   positions[6] = -1;positions[7] = -1;positions[8] = -1;    }
    if(x==n-1 && y == m-1)
    {   positions[2] = -1;positions[5] = -1;positions[6] = -1;positions[7] = -1; positions[8] = -1; }
    return;
}
//return which is the minimum of a vector of 9 real components
//if the 4th position does not contain the minimum, then the returned minimum is the first.
//otherwise, always 4 is returned
int     WhichMin(float *b)
{
    int i,mini,j=0;
    float mn;
    mn = 10000;
    j=0;
    for(i=0;i<9;i++)
    {
        if(b[i]<mn && i!=4) {mn = b[i]; mini=i;}
    }
    if(mn>=b[4]) return(4);
    return(mini);
}
//given the position of the flow direction in terms of a 9-neighbours vector,
//is returned the code of SFD8
int     Dir(int k)
{
    if(k==0)    return(32);
    if(k==1)    return(64);
    if(k==2)    return(128);
    if(k==3)    return(16);
    if(k==4)    return(0);
    if(k==5)    return(1);
    if(k==6)    return(8);
    if(k==7)    return(4);
    if(k==8)    return(2);
}
```

```
//given a position i of the matrix, it returns 0 if is not on the border of the matrix
//and return 1 if yes
int     Isintheborder(int i,int m,int n)
{
    int x,y;
    x = i%n;
    y = (i -x)/n;
    if(x==n-1 || x==0 || y==m-1 || y==0) return(1);
    return(0);
}
//given the nprev information, it assign the index vector, containing:
//index[i] is -1 if i has not previous, and i is the index[i]-th element with previous
//if i has previous
void    Setindex(int *index,short *nprev,int m,int n)
{
    int i,j;

    j = 0;
    for(i=0;i<m*n;i++)
    {
        if(nprev[i]>0) {j++; index[i] = j-1;}
        else index[i] = -1;
    }
}


/*reading and writing functions*****************************************************************/
void    InitDataBinnary(float *a, FILE *f,int m, int n)
{
    int i;
    i=fread(a,4,m*n,f);
    return;
}
void    WriteMatrixDecimalCSVshort(short *a,int m,int n,FILE *f)
{
    int i,j;
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
            fprintf(f,"%4d,",a[n*i+j]);
        fprintf(f,"\n");
    }
    return;
}
void    WriteMatrixDecimalCSVint(int *a,int m,int n,FILE *f)
{
    int i,j;
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
            fprintf(f,"%4d,",a[n*i+j]);
        fprintf(f,"\n");
    }
    return;
}
void    WriteDataBinnaryShort(short *a,FILE *f,int m, int n)
{
    fwrite(a,sizeof(short),m*n,f);
    return;
}
void    WriteDataBinnary(int *a,FILE *f,int m, int n)
{
    fwrite(a,sizeof(int),m*n,f);
    return;
}
void WriteNeighbours(float *b,int i,int n)
{

    printf("%2.7f,%2.7f,%2.7f\n",b[i-n-1],b[i-n],b[i-n+1]);
    printf("%2.7f,%2.7f,%2.7f\n",b[i-1],b[i],b[i+1]);
    printf("%2.7f,%2.7f,%2.7f\n\n",b[i+n-1],b[i+n],b[i+n+1]);
    return;
}
#endif
```