a3.2.1 Rtca.c.

```
#include "turtle.h"
#include "t_alloc.h"
#include "t_io.h"
#include "t_list.h"
#include "t_random.h"
#include "t_utilities.h"
#include "t_datamanipulation.h"
#include "tensor3D.h"
#include "t_networks.h"

#define MAX_COLS_NUMBER 70

// Not used here but present womewhere else in the file. It could be elimated
// in the future

 LONGPAIR NW, N, NE;
 LONGPAIR  W, H,  E;
 LONGPAIR SW, S, SE;
 LONGPAIR *home,*work;

//Function declaration (prototyping)
void rtca(SHORTMATRIX *flow,DOUBLEMATRIX *ca, DOUBLEMATRIX *we);

int main(void)

{

double index;
char *inputfile,*weightsfile,*outputfile,*program="rtca";
extern char *WORKING_DIRECTORY;

SHORTMATRIX *m;
DOUBLEMATRIX *we,*ca;
FLOATVECTOR *pxl,*novalues;
FILE *input,*weights,*output;

printf("\nRTCA calculates the total contributing area of a networks using its drainage directions with random injections\n");
printf("Copyright 2006 by Riccardo Rigon and Matteo Convertino under the Free Software Foundation spirit\n\n");

WORKING_DIRECTORY=get_workingdirectory();

printf("ENTER THE INPUT FILE NAME\n");
inputfile=get_filename(WORKING_DIRECTORY,program);
input=t_fopen(inputfile,"r");
index=read_index(input,PRINT);
pxl=read_floatarray(input,PRINT);
novalues=read_floatarray(input,PRINT);
m=read_shortmatrix(input,"a",PRINT);

printf("ENTER THE WEIGHTS FILE NAME\n");
weightsfile=get_filename(WORKING_DIRECTORY,program);
weights=t_fopen(weightsfile,"r");
index=read_index(weights,PRINT);
pxl=read_floatarray(weights,PRINT);
novalues=read_floatarray(weights,PRINT);
we=read_doublematrix(weights,"a",PRINT);
ca=new_doublematrix(m->nrh,m->nch);
initialize_doublematrix(ca,0);

printf("ENTER THE OUTPUT FILE NAME\n");
outputfile=get_filename(WORKING_DIRECTORY,program);
output=t_fopen(outputfile,"w");

rtca(m,ca,we);

fprintf(output,"/** This_is_a_turtle_file created by RTCA */");
fprintf(output,"index{3}\n");
fprintf(output,"/** ------------------------------------ */");
fprintf(output,"1: float array pixels size");
write_floatarray_elements(output,pxl,80);
fprintf(output,"2: float array novalues {-1,0}\n");
```

```
fprintf(output,"3: double matrix rtca {%ld,%ld}\n",ca->nrh,ca->nch);
write_doublematrix_elements(output,ca,80);
t_fclose(output);
t_fclose(weights);
t_fclose(input);

printf("END OF PROCESSING\n");

getchar();
getchar();

return 1;

}



/* ------------------------------------------------------------------------------ */


/*          Inputs:  1) m shortmatrix of flowing directions 2) we matrix of weights
            Outputs: 1) ca matrix that will contain the values of contributing area [pixels]*/

void rtca(SHORTMATRIX *flow,DOUBLEMATRIX *ca, DOUBLEMATRIX *we)
{
/* char ch;*/
long i,j,punto[2];
// Just checking that the matrixes have the same dimensions
if(flow->nrh !=ca->nrh || flow->nch !=ca->nch || we->nrh !=ca->nrh || flow->nch !=ca->nch){
            t_error("Flow matrix and contributing area matrix have not the same dimensions");
}

//For any row in the contributing area, flow and weights matrixes
for(i=1;i<=ca->nrh;i++) {
//Say at which point of the computation we are
            meter(i,ca->nrh,10,"Lines of matrix ca parsed","\n");
//for any column of the contributing area, flow and weights matrixes
            for(j=1;j<=ca->nch;j++) {
// If the drainage direction is NOT 9 (we are not outside the basin)
                        if(flow->element[i][j]!=9 ){
// Select the point for the calculation of the contributing areas
                        punto[0]=i;punto[1]=j;
// Untill we are not at the outlet (which is marked by number 10)
                        while(flow->element[punto[0]][punto[1]]<9){
//Increment the value of the TCA matrix of the appropriate weights for all
//the point downstream the point chosen
                                    (ca->element[punto[0]][punto[1]])+=we->element[i][j];
//Obviously we move downstream (or downhill) one pixel by one pixel
                                    go_downstream(punto,flow->element[punto[0]][punto[1]],ca->nch);

                                }
//I we are at the outlet you have to add the value out of the while loop
                                if(flow->element[punto[0]][punto[1]]==10)(ca->element[punto[0]][punto[1]])+=we->element[i][j];
                        }
            }
}
}
```

a3.2.2 R-Metropolis.c.

```
#include "turtle.h"
#include "t_alloc.h"
#include "t_io.h"
#include "t_list.h"
#include "t_random.h"
#include "t_utilities.h"
#include "t_datamanipulation.h"
#include "tensor3D.h"
#include "t_networks.h"
```

/*! \mainpage r-metropolis

The program r_metropolis takes a network and optimize it according to a simulated anenaling algorithm. Main references for this work are
Rodriguez-Iturbe and Rinaldo, Fractal River Networks, chance and self-organization, CUP, 1997
and Rinaldo, A., I. Rodriguez-Iturbe and R. Rigon, Channel Networks, Annual Review of Earth and Planetary Sciences, 26, 289-327, 1998

\author Riccardo Rigon

\date 1997-2006, licensed under GPL

\version 0.875

*/

/*!

Name: r_metropolis, r_weighted_metropolis

```
double r_metropolis(SHORTMATRIX *,DOUBLEMATRIX *,SHORTMATRIX *,\
        double ,long, double, long );
double r_weighted_metropolis(SHORTMATRIX *,DOUBLEMATRIX *,SHORTMATRIX *,\
        double ,long, double, long );
```

General information: Same as metropolis and weightedmetropolis but taking a contributing area
as DOUBLEVECTOR instead that LONGVECTOR

\author Riccardo Rigon,
\date December 1998, modified August 2006 (with Matteo Convertino)

Inputs: 1) the matrix of flow directions; 2) the matrix of contributing areas; 3) the matrix
of active sites; 4) ex, the exponent in the energy expenditure formula; 5) the number of iteration to be performed;
6) the temperature used in the Metropolis algorithm; 7) an area threshold: pixels below
thisdo not eneter in the sum of energy expenditure

Returns: the energy expenditure of the final network
Needs: alloc.c, error.c, t_io.c, list.c, random.c, net.c
Related Routines: tca, eden
See Also:
Keywords: networks, metropolis algorithm
References: Fractal River Basins, I. Rodriguez-Iturbe and A. Rinaldo, 1997
Bugs & limitations:
*/

```
double r_metropolis(SHORTMATRIX *,DOUBLEMATRIX *,SHORTMATRIX *,\
        double ,long, double, double );

double r_weighted_metropolis(SHORTMATRIX *,DOUBLEMATRIX *,SHORTMATRIX *,\
        double ,long, double, double  );
```

/**

Name: reset_active_sites_counter

Version: 1.0

Synopsis: LONGPAIR *reset_active_sites_counter(LONGPAIR *ink,long as);

General information: Used by metropolis and weighted_metropolis.
before starting the Metropolis algorithm is set a matrix where
sites whose contributing area has been changed are marked. This is used to calculate the
variation of energy expenditure without parsing the whole contributing area matrix.
The number of active sites after each iteration is by itself an interesting quantity and
in some cases it is useful to mantain a linked list of the number of active sites.

Authors & date: Riccardo Rigon, November 1997

Inputs:

Returns: the list containing the number of active sites at a given time

Needs: alloc.c, error.c, t_io.c, list.c, random.c, net.c

Related Routines: tca, metropolis, eden

See Also:

Keywords: networks

References: Fractal River Basins, I. Rodriguez-Iturbe and A. Rinaldo, 1997

Bugs & limitations:

*/

LONGPAIR *reset_active_sites_counter(LONGPAIR *ink,long as);

/**

Name: r_randomchangeflow

Version: 1.0

Synopsis: void randomchangeflow(long *,long *, SHORTMATRIX *m,DOUBLEMATRIX *tca);

General information:Used by r_metropolis and r_weighted_metropolis. changes randomly the flow directions

Authors & Date: Riccardo Rigon, November 1997

Inputs: 1) 2) the row and column index of the point whose flow is going to be changed;
3) a pointer to the matrix of flow direction, 4) a pointer to the matrix of contributing areas

Returns: void

Needs: alloc.c, error.c, t_io.c, list.c, random.c, net.c

Related Routines: tca

See Also:

Keywords: networks

References: Fractal River Basins, I. Rodriguez-Iturbe and A. Rinaldo, 1997

Bugs & limitations:

*/

void r_randomchangeflow(long *,long *, SHORTMATRIX *m,DOUBLEMATRIX *tca);

/**

Name: active_sites

Version: 1.0

Synopsis: void active_sites(SHORTMATRIX *flow,SHORTMATRIX *active,
long x,long y, long a, long b,
long* as);

General information: Used by metropolis and weighted_metropolis,
marks the number of active sites, defined as those whose contributing area
has changed after the flow direction was moved from (x,y) to (a,b)

Authors & date: Riccardo Rigon, November 1997

Inputs: 1) the pointer to the flowing sirection matrix; 2) the pointer to the matrix of
active sites; 3) 4) the row and column indexes of the old flowing direction; 5) 6)
the row and column indexes of the new flowing direction; 5) a ponter to a varable containing
the number of active sites

Returns: void

Needs: alloc.c, error.c, t_io.c, list.c, random.c, net.c

Related Routines: tca

See Also:

Keywords: networks

References: Fractal River Basins, I. Rodriguez-Iturbe and A. Rinaldo, 1997

Bugs & limitations:

*/

```
void active_sites(SHORTMATRIX *flow,SHORTMATRIX *active,
                  long x,long y, long a, long b,
                  long* as);
```

/**

Name: r_energy_variation, r_weighted_energy_variation

Version: 1.0

Synopsis:

```
double r_energy_variation(SHORTMATRIX *flow,DOUBLEMATRIX *tca,SHORTMATRIX *active,
                  long x,long y, long a,long b,
                  double ex,long tr);

double weighted_energy_variation(SHORTMATRIX *flow,DOUBLEMATRIX *tca,SHORTMATRIX *active,
                  long x,long y, long a,long b,
                  double ex,long tr);
```

General information: Used by r_metropolis and r_weighted_metropolis.
After a flowing direction has been changed it is calculated the energy variation
caused. In weighted_energy_variation energy expenditure is calculated with weights
in the different directions specified by the global variable weights.

Authors & date: Riccardo Rigon, December 1998

Inputs: 1) the pointer to the flowing sirection matrix; 2) the pointer to the matrix of
contributing areas; 3) the pointer to the matrix of active sites;
4) 5) the row and column indexes of the old flowing direction; 6) 7)
the row and column indexes of the new flowing direction; 8) the exponent of
energy expenditure formula; 9) a threshold as in energy_expenditure

Returns: the variation in energy expenditure after the change of flow direction

Needs: alloc.c, error.c, t_io.c, list.c, random.c, net.c

Related Routines: tca, metropolis, eden

See Also:

Keywords: networks

References: Fractal River Basins, I. Rodriguez-Iturbe and A. Rinaldo, 1997

Bugs & limitations:

*/

```
double r_energy_variation(SHORTMATRIX *flow,DOUBLEMATRIX *tca,SHORTMATRIX *active,
                long x,long y, long a,long b,
                double ex,double  tr);

double r_weighted_energy_variation(SHORTMATRIX *flow,DOUBLEMATRIX *tca,SHORTMATRIX *active,
                long x,long y, long a,long b,
                double ex,double  tr);
```

/**

Name: reset_active_and_set_tca, reset_active

Synopsis:

```
void r_reset_active_and_set_tca(SHORTMATRIX *flow,DOUBLEMATRIX *tca,SHORTMATRIX *active,
                long x,long y, long a,long b);

void reset_active(SHORTMATRIX *flow,SHORTMATRIX *active, long x,long y,
long a,long b);
```

General information: Used by r_metropolis and r_weighted_metropolis.
A change of flow direction can be accepted or rejected according to the Metropolis
rules. If accepted the contributing area matrix must be updated. In any case the active sites matrix must be
zeroed for the new iteration. The two routine accomplish these tasks.

Authors & Date: Riccardo Rigon, November 1997

Inputs: 1) the pointer to the flowing sirection matrix; 2) the pointer to the matrix of
contributing areas; 3) the pointer to the matrix of active sites;
4) 5) the row and column indexes of the old flowing direction; 6) 7)
the row and column indexes of the new flowing direction;

Returns: void

Needs: alloc.c, error.c, t_io.c, list.c, random.c, net.c

Related Routines: tca

See Also:

Keywords: networks

References: Fractal River Basins, I. Rodriguez-Iturbe and A. Rinaldo, 1997

Bugs:

*/

void r_reset_active_and_set_tca(SHORTMATRIX *flow,DOUBLEMATRIX *tca,SHORTMATRIX *active,
long x,long y, long a,long b);

void reset_active(SHORTMATRIX *flow,SHORTMATRIX *active, long x,long y,
long a,long b);

/**

Name: r_loopQ

Version: 1.0

Synopsis: short loopQ(long ,long , long , long ,SHORTMATRIX *, DOUBLEMATRIX *);

General information: Used by metropolis and weighted_metropolis,
when a flowing direction is changed it must be avoided to create loops.

Authors & Date: Riccardo Rigon, November 1997

Inputs: 1) 2) the row and column indexes of the old flowing direction; 3) 4)
the row and column indexes of the new flowing direction;
5) the pointer to the flowing sirection matrix; 6) the pointer to the matrix of
contributing areas.

Returns: 1 if there is a loop 0 otherwose

Needs: alloc.c, error.c, t_io.c, list.c, random.c, net.c

Related Routines: tca

See Also:

Keywords: networks

References: Fractal River Basins, I. Rodriguez-Iturbe and A. Rinaldo, 1997

Bugs & limitations:

*/

short r_loopQ(long ,long , long , long ,SHORTMATRIX *, DOUBLEMATRIX *);

LONGPAIR NW, N, NE;
LONGPAIR W, H, E;
LONGPAIR SW, S, SE;
LONGPAIR *home,*work;

/*---
---*/

```
int main(void )
{

long i/*! \param iterator */,k/*! \param iterator */,l/*! \param iterator */;
long itr=100000/*! \param number of iterations */,tca=0 /*! \param value of the contributing area */;
long ensemble=10 /*! \param number of OCNs in the same run */,ensembleb=10 /*! \param ??? */;
char *initial,*initialflow,*initialarea,inputflow[64*FILENAME_MAX],inputarea[64*FILENAME_MAX];
char  oldoutputflow[FILENAME_MAX],oldoutputarea[FILENAME_MAX],*ou="sample.out",*ouflow,*ouarea;
char outputflow[FILENAME_MAX]="",outputarea[FILENAME_MAX]="",*Tfile="ensemble.in";
char *message="------------------------------------------------------------------------";
char *experimentlabel,*program="r_metropolis";
extern char *WORKING_DIRECTORY /*! \param it contains the working directory*/;
double exa=0.5 /*! \param Exponent for the calculation of the Energy dissipation*/,energy=0/*! \param energy expenditure value*/;
long tmp /*! \param tmp */;
short D8 /*! \param type of simulation: D8 = 1 means with weighted topology, D8=0 means with equal weight of the D8 directions */;

SHORTMATRIX *m /*! \param Matrix containing the flow directions */,*active /*! \param Matrix containing the sites that, at each iteration step are
modified*/;
DOUBLEMATRIX  *ca/*! \param Matrix containing the total contributing areas */;
DOUBLEVECTOR *T/*! \param temperatures used in the simulated annealing algorithm*/;
LONGVECTOR  *ITR/*! \param Contains the index of the colims of the temperatures file*/;
SHORTVECTOR *PRT/*! \param Contains 0 if the simulation result of the temperature must be printed*/;
FLOATVECTOR *FLFLOW,*FLAREA,*NOFLOW,*NOAREA;

HEADER Th;
FILE  *Tstream,*ostream,*oostream, *istream,*iistream;

printf("\nr_METROPOLIS uses the Metropolis algorithm to obtain OCNs\n");
printf("Copyright 1997-2006 by Riccardo Rigon under the GPL\n\n");

// Initialization of the program
work=topology();
WORKING_DIRECTORY=get_workingdirectory();

printf("ENTER THE INITIAL NETWORK: ");
initial=get_filename(WORKING_DIRECTORY,program);
initialflow=join_strings(initial,".flow");
initialarea=join_strings(initial,".area");

printf("ENTER THE OUTPUT NETWORKS NICKNAME: ");
ou=get_filename(WORKING_DIRECTORY,program);
ouflow=join_strings(ou,".flow");
ouarea=join_strings(ou,".area");

printf("ENTER THE TEMPERATURES FILE NAME: ");
Tfile=get_filename(WORKING_DIRECTORY,program);
Tstream=t_fopen(Tfile,"r");
read_index(Tstream,NOPRINT);
experimentlabel=readandstore_comment(Tstream,TEST,MAXBUFFERSIZE);
read_matrixheader(Tstream,&Th);
printf("%ld\n",Th.dimensions[1]);
printf("ENTER THE NUMBER OF ITERATIONS: \n");
itr=get_parameter(WORKING_DIRECTORY,program);


printf("ENTER THE EXPONENT FOR ENERGY EXPENDITURE: \n");
exa=get_parameter(WORKING_DIRECTORY,program);
printf("ENTER A THRESHOLD FOR ENERGY EXPENDITURE COMPUTATION: \n");
tca=get_parameter(WORKING_DIRECTORY,program);
printf("ENTER '1' FOR WEIGHTED DIRECTIONS '0' OTHERWISE: \n");
D8=get_parameter(WORKING_DIRECTORY,program);

T=new_doublevector(Th.dimensions[1]);
ITR=new_longvector(Th.dimensions[1]);
PRT=new_shortvector(Th.dimensions[1]);
if(itr <= 0){
        t_error("Obsolete scheme");
}else {

        for(i=1;i<=T->nh;i++){
           fscanf(Tstream,"%ld",&tmp);
                fscanf(Tstream,"%lf",&(T->element[i]));
                fscanf(Tstream,"%hd",&(PRT->element[i]));

   }
        initialize_longvector(ITR,itr);
```

```
}

print_doublevector_elements(T,10);
print_longvector_elements(ITR,10);
print_shortvector_elements(PRT,10);

t_fclose(Tstream);
// For any temperature
for(k=1;k<=T->nh;k++){

        if(k==1){
    strcpy(inputflow,initialflow);
    strcpy(inputarea,initialarea);
        }else{
                sprintf(inputflow,"%s_%3.3f%s",ou,T->element[k-1],".flow");
                sprintf(inputarea,"%s_%3.3f%s",ou,T->element[k-1],".area");

        }

        printf("READING FROM\n%s and\n%s\n",inputflow,inputarea);

        istream=t_fopen(inputflow,"r");
        iistream=t_fopen(inputarea,"r");
        read_comment(istream,TEST,MAXBUFFERSIZE,NOPRINT);
        read_comment(iistream,TEST,MAXBUFFERSIZE,NOPRINT);
        ensemble=read_index(istream,NOPRINT)/3;
        ensembleb=read_index(iistream,NOPRINT)/3;
        if(ensemble!=ensembleb) t_error("Flow directions and area files not matching");
   strcpy(oldoutputflow,outputflow);
   strcpy(oldoutputarea,outputarea);
        sprintf(outputflow,"%s_%3.3f%s",ou,T->element[k],".flow");
        sprintf(outputarea,"%s_%3.3f%s",ou,T->element[k],".area");
        ostream=t_fopen(outputflow,"w");
   oostream=t_fopen(outputarea,"w");
        printf("WORKING ON:\n%s and\n%s\n",outputflow,outputarea);
        write_turtle(ostream,program,inputflow);
        write_turtle(oostream,program,inputarea);
        fprintf(ostream,"index{%ld}\n",3*ensemble);
        fprintf(oostream,"index{%ld}\n",3*ensemble);
        fprintf(ostream,"%s\n",experimentlabel);
        fprintf(oostream,"%s\n",experimentlabel);

// For any OCN in the files

        for(l=1;l<=ensemble;l++){
                fprintf(ostream,"%ld: float array ocn {1,1,0,0,%3.3f}\n",3*l-2,T->element[k]);
                fprintf(ostream,"%ld: float array novalues {1,9}\n",3*l-1);
          FLFLOW=read_floatarray(istream,NOPRINT);
    NOFLOW=read_floatarray(istream,NOPRINT);
          FLAREA=read_floatarray(iistream,NOPRINT);
          NOAREA=read_floatarray(iistream,NOPRINT);
                m=read_shortmatrix(istream,"a",NOPRINT);
    ca=read_doublematrix(iistream,"a",NOPRINT);
        active=new_shortmatrix(m->nrh,m->nch);
                initialize_shortmatrix(active,0);
                printf("\nCOMPUTING SAMPLE %ld : T=%f\n ",l,T->element[k] );
//Run the simulations

                if(D8==0){
                        energy=r_metropolis(m,ca,active, exa,ITR->element[k],T->element[k],tca);
                }else if(D8==1){
                        energy=r_weighted_metropolis(m,ca,active, exa,ITR->element[k],T->element[k],tca);
                } else {
                        t_error("Option not yet implemented");
                }

// Finalize
    write_comment(ostream,message,80);
                fprintf(ostream,"%ld: short matrix flow {%ld, %ld}\n",3*l,m->nrh,m->nch);
          write_shortmatrix_elements(ostream,m,80);
                write_comment(ostream,message,80);
                fprintf(oostream,"%ld: float array ocn {1,1,0,0,%3.3f}\n",3*l-2,energy);
                fprintf(oostream,"%ld: float array novalues {-1,0}\n",3*l-1);
                fprintf(oostream,"%ld: double matrix tca {%ld,%ld} \n",3*l,ca->nrh,ca->nch);
                write_doublematrix_elements(oostream,ca,80);
                free_shortmatrix(m);
                free_shortmatrix(active);
```

```
                    free_doublematrix(ca);
                    free_floatvector(FLFLOW);
                    free_floatvector(FLAREA);
                    free_floatvector(NOFLOW);
                    free_floatvector(NOAREA);

            }


            t_fclose(ostream);
            t_fclose(istream);
            t_fclose(oostream);
            t_fclose(iistream);

            if(k>1){
                if(PRT->element[k-1]==0){

                                if(remove(oldoutputflow) ==-1) printf("Warning::Remove error: %s could not be removed",oldoutputflow);
                                if(remove(oldoutputarea) ==-1) printf("Warning::Remove error: %s could not be removed",oldoutputarea);

        }
            }




}


free_doublevector(T);
free_longvector(ITR);
free_shortvector(PRT);


printf("END OF COMPUTATION\n");

getchar();
getchar();

return 1;

}




/*---------------------------------------------------------------------------

---------------------------------------------------------------------------*/
double r_metropolis(SHORTMATRIX *m,DOUBLEMATRIX *ca,SHORTMATRIX *active,
            double ex,long itr, double T,double th)
{


long j /*! \param iterator */,rr/*! \param random number */,pp=439357/*! \param seed for the first random number generation */;
long qq=36887/*! \param seed for the second random generator*/,mtr/*! \param ancillary parameter needed for printing the status of the simulation*/;
long pt1[2],pt2[2],pt3[2],x,y,newx,newy,as=0;
long kk=1919,jj=-1319;
double de=0/*! \param Energy expenditure */,dediff=0 /*! \param energy Expenditure difference*/,ee,q=0;

//Initialization of the random generators (see  Numerical Recipes)
rr=ran1(&pp);
rr=ran2(&jj);
prand(pt3,m->nrh,m->nch,0);
rr=pt3[0]+pt3[1];
for(j=1;j<=rr;j++) {q=ran2(&kk);}
rr=100*q;
for(j=1;j<=rr;j++) {q=ran1(&qq);}

//Energy Expenditure of the initial network
ee=r_energyexpenditure(ca, ex, th);

printf("Initial Energyexpediture: %f\n",ee);
//Just in case a too low temperature is chosen (remember that we are using double precesion numbers)
```

```
if(T < 0.00000001) T=0.00000001;

//It iterate as you requested
for(j=1;j<=itr;j++){
            // It shows the progress of computation
   if(itr < 100){
            if( (j  % 10 )==0){
                        printf("Iterations  %ld/%ld\n", j,itr);
            }   }else{
                         mtr=ceil(itr/100);
                         if( (j  % mtr )==0){
                        printf("Iterations  %ld/%ld\n", j,itr);
            }}
   de=0;
//Generate a random position inside the basin
                prand(pt3,m->nrh,m->nch,1);
                x=pt3[0];y=pt3[1];
//While the point chosen is not INSIDE the basin
                        while( m->element[pt3[0]][pt3[1]]==9 || m->element[pt3[0]][pt3[1]]==10) {
                                                // generate it again
                                                prand(pt3,m->nrh,m->nch,1);
                                                x=pt3[0];y=pt3[1];
                        }
// It preserve the starting point in pt3 and use a new pt2 for the computations
                pt2[0]=pt3[0];
                pt2[1]=pt3[1];
// It goes downhill one pixel
            go_downstream(pt2,m->element[pt2[0]][pt2[1]],m->nch);
// It randomly changes the direction of flow. This new direction is stored in pt1
   r_randomchangeflow(pt3,pt1,m,ca);
// The new direction is stored again in newx and newy. It is a redundant step
                newx=pt1[0],newy=pt1[1];
// If the new direction is the same that the old one,
                if(newx==pt2[0] && newy==pt2[1]) {
// set the flas as to 0,
                                        as=0;
// otherwise
                } else {
//              find the sites that are affected by this change,
                     active_sites(m,active,x,y,newx,newy,&as);
// and estimate the the variation of energy expenditure caused by this change.
                           de=r_energy_variation(m,ca,active,x,y,newx,newy,ex,th);
// According to the Metropolis' algorithm, generate a new random number, and
                                        q=ran2(&kk);
// if           the random number obeys the following rule
                                        if(exp(-de/T)<=q) {
// cancel the changes made and reset the flag as to 0,
                                        reset_active(m,active,  x, y,  newx, newy);
                                        as=0;
// otherwise
                                        }else {
// accumulate the variation in energy expenditure;
                                                dediff+=de;
// reset the active site matrix to 0 and update the contributing area matrix;
                                                r_reset_active_and_set_tca(m, ca, active,  x, y,  newx, newy);
//finally update the flow directions to account for the change made
                                                m->element[x][y]=coords2int(x,y,newx,newy);
                                        }
            }


}

printf("Finale Energy Expenditure: %f\n",ee+dediff);

return ee+dediff;


}

/*---------------------------------------------------------------------------

---------------------------------------------------------------------------*/
double r_weighted_metropolis(SHORTMATRIX *m,DOUBLEMATRIX *ca,SHORTMATRIX *active,
            double ex,long itr, double T,double th)
{
```

```
long j,rr,pp=1313,qq=-1313;
long pt1[2],pt2[2],pt3[2],x,y,newx,newy,as=0;
long kk=1919,jj=-1919;
double de=0,dediff=0,ee,q=0;
//static char s2[]=".ee",ch;


printf("Starting the metropolis algorithm at T : %f\n",T);
rr=ran1(&pp);
rr=ran2(&jj);
prand(pt3,m->nrh,m->nch,0);
rr=pt3[0]+pt3[1];
for(j=1;j<=rr;j++) {q=ran2(&kk);}
rr=100*q;
for(j=1;j<=rr;j++) {q=ran1(&qq);}

ee=r_weighted_energyexpenditure(m,ca, ex, th);

if(T < 0.00000001) T=0.00000001;

for(j=1;j<=itr;j++){

   de=0;
                        prand(pt3,m->nrh,m->nch,1);
              x=pt3[0];y=pt3[1];
                        while( m->element[pt3[0]][pt3[1]]==9 || m->element[pt3[0]][pt3[1]]==10) {
                                                prand(pt3,m->nrh,m->nch,1);
                                                x=pt3[0];y=pt3[1];
                        }

              pt2[0]=pt3[0];
              pt2[1]=pt3[1];

            go_downstream(pt2,m->element[pt2[0]][pt2[1]],m->nch);
    r_randomchangeflow(pt3,pt1,m,ca);
              newx=pt1[0],newy=pt1[1];
              if(newx==pt2[0] && newy==pt2[1]) {
                                as=0;
              } else {
                  active_sites(m,active,x,y,newx,newy,&as);
                        de=r_weighted_energy_variation(m,ca,active,x,y,newx,newy,ex,th);
                                q=ran2(&kk);
                                if(exp(-de/T)<=q) {
                                reset_active(m,active,  x, y,  newx, newy);
                                as=0;
                                }else {
                                                dediff+=de;
                                                r_reset_active_and_set_tca(m, ca, active,  x, y,  newx, newy);
                                                m->element[x][y]=coords2int(x,y,newx,newy);
                                }
            }


}

printf("E: %f\n",r_weighted_energyexpenditure(m,ca, ex, th));
return ee+dediff;

}


/*-------------------------------------------------------------*/

LONGPAIR * reset_active_sites_counter(LONGPAIR *ink,long as )

{

LONGPAIR *pink /*! \param An ancillary LONGPAIR list */;

// Use the new pointer to perform the operations and preserve the head of the list in ink
pink=ink;

// Usual error catch
if(pink==NULL){
              t_error("This cannot  happen");
} else {
```

```
// Untill the end of the LONGPAIR list
		while( pink->next!=NULL){
			if(pink->i==as){
			 (pink->j)++;
			 return ink;
		} else {
			pink=pink->next;
		}


    }
   if(pink->i==as){
			 (pink->j)++;
			 return ink;
		}
		if(pink->next!=NULL){
			t_error("This cannot happen here");
		} else {
			pink->next=new_longpair();
			pink->next->i=as;
			pink->next->j=1;
			(pink->next)->next=NULL;
			return ink;

		}

 }

return ink;
}



/*-----------------------------------------------------------------------------
It changes the flow in a admissible direction. Inputs are: the  point
where the flow is to be changed and the flow matrix. Note: The flow is
changed picing up the first direction contained in the list "work". This
list is shuffled every time it is inspected. Because the point chosen to
be modified are randomly chosen in the domain the number of operations
required to obtain a new flow direction is random and the head of the list
can be considered random.
-----------------------------------------------------------------------------*/

void r_randomchangeflow(long *p,long *flow, SHORTMATRIX *m,DOUBLEMATRIX *tca)

{


long r,u=0;
long j/*! \param random number identigying the random direction of flow */;
long jj=1919;
extern LONGPAIR *work,*home;
LONGPAIR *now;

// Generate a random direction
j=floor(8*ran1(&jj));
// Rotate the directions list j times
work=rotate(work,j);
now=work;
// Transform the random number in a coordinate direction
r=m->element[p[0]+now->i][p[1]+now->j];
flow[0]=p[0];
flow[1]=p[1];
// Untill a fesible direction is chosen serch for a new direction
while(r==9 ||  r==0 || r==11 ||
     crossQ(p[0],p[1],p[0]+now->i,p[1]+now->j,m) || r_loopQ(p[0],p[1],p[0]+now->i,p[1]+now->j,m,tca)){
// Shuffle  the work  list  (which contains the directions)
      if(work->next==NULL) {
          work=prependto(home,work);
			home=NULL;
			u=1;
				break;
      }else{
			work=work->next;
			now->next=NULL;
			home=prependto(home,now);
				now=work;
```

```
                                        u=0;
                                r=m->element[p[0]+work->i][p[1]+work->j];
        }
}

if(u==1){
// This is if the unique direction possible is the one already chosen : thus the flowing direction is set to the old one.
                go_downstream(flow,m->element[flow[0]][flow[1]],m->nch);

}else{
// The new direction chosen is fine and set up
                flow[0]=p[0]+now->i;
                flow[1]=p[1]+now->j;
}

work=appendto(work,home);
home=NULL;



}

/*   */

void active_sites(SHORTMATRIX *flow,SHORTMATRIX *active,
                        long i,long j, long newi, long newj,
                        long* as)

{

long punto[2],ass=0;
//double dex=0;



active->element[i][j]=1;
ass++;
// The active sites, by definition are those which have a variation in their contributing area,
punto[0]=i;
punto[1]=j;
//either a decrease, after a flow diversion oif a point upstream
go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
while(flow->element[punto[0]][punto[1]]!=10){
                active->element[punto[0]][punto[1]]=1;
                ass++;
                go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);

}


// or an increase for a new inflow
punto[0]=newi;
punto[1]=newj;

while(flow->element[punto[0]][punto[1]]!=10 && active->element[punto[0]][punto[1]]!=1){
                active->element[punto[0]][punto[1]]=1;
                ass++;
                go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
}

if(active->element[punto[0]][punto[1]]==1){
                while(flow->element[punto[0]][punto[1]]!=10){
                                active->element[punto[0]][punto[1]]=0;
                                ass--;
                                go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
                }


}


(*as)=ass;



}
```

```
/* */

double r_energy_variation(SHORTMATRIX *flow,DOUBLEMATRIX *tca,SHORTMATRIX *active,
                    long i,long j, long newi,long newj,
                    double ex,double tr)

{

long punto[2];
double dex=0,bi,bo,tmp,diff;


//This is the contributing area of the chosen site: it is to substract in the old flowing direction and to be added in the new flowing direction
diff=tca->element[i][j];
// In the old direction
punto[0]=i;
punto[1]=j;
go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
while(active->element[punto[0]][punto[1]]==1 && flow->element[punto[0]][punto[1]]!=10){
// Once downstream, the value of the  contributing area is stored
            tmp=tca->element[punto[0]][punto[1]];
//  If the change under examination would be accepted, the contributing areaof the point downstream  would be
//  the actual area minus the area of the point which is upstream and has changed the flowing direction. The fact that after the diversion
//  the are of the point  (i,j) is below the threshold for energy expenditure calculation must be considered
            if((tmp-diff)>tr) bi=pow(tmp-diff,ex);
            else bi=0;
            if(tmp >tr) bo=pow(tmp,ex);
            else bo=0;
// But the contribution of the pixel examined to energy expenditure difference is the difference of the hypothetical
// new contributing areas minus the old value
            dex+=(bi-bo);
            go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
}

// In the new direction
punto[0]=newi;
punto[1]=newj;


while(active->element[punto[0]][punto[1]]==1 && flow->element[punto[0]][punto[1]]!=10){
            tmp=tca->element[punto[0]][punto[1]];
//  If the change under examination would be accepted, the contributing areaof the point downstream  would be
//  the actual area plus the area of the point which is upstream and has changed the flowing direction.

            if((tmp+diff)>tr) bi=pow(tmp+diff,ex);
            else bi=0;
            if(tmp >tr) bo=pow(tmp,ex);
            else bo=0;
            dex+=(bi-bo);
            go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);

}

// Return the variation in energy expenditure
return dex;


}

/* */

double r_weighted_energy_variation(SHORTMATRIX *flow,DOUBLEMATRIX *tca,SHORTMATRIX *active,
                    long i,long j, long newi,long newj,
                    double ex,double tr)

{

long punto[2];
double dex=0,bi,bo,diff,delta,tmp,df;
extern double weight[12];


df=tca->element[i][j];
diff=sqrt((tca->element[i][j])*weight[flow->element[i][j]]);
delta=diff;
```

```
punto[0]=i;
punto[1]=j;
go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
while(active->element[punto[0]][punto[1]]==1 && flow->element[punto[0]][punto[1]]!=10){
		tmp=tca->element[punto[0]][punto[1]];

		if((tmp-df)>tr) bi=pow(tmp-diff,ex);
		else bi=0;
		if(tmp >tr) bo=pow(tmp,ex);
		else bo=0;
		dex+=(bi-bo)*weight[flow->element[punto[0]][punto[1]]];
		go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
}


punto[0]=newi;
punto[1]=newj;
diff=sqrt(tca->element[i][j]*weight[coords2int(i,j,punto[0],punto[1])]);
delta-=diff;

while(active->element[punto[0]][punto[1]]==1 && flow->element[punto[0]][punto[1]]!=10){
		tmp=tca->element[punto[0]][punto[1]];
		if((tmp+df)>tr) bi=pow(tmp+diff,ex);
		else bi=0;
		if(tmp >tr) bo=pow(tmp,ex);
		else bo=0;
		dex+=(bi-bo)*flow->element[punto[0]][punto[1]];
		go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);

}


return dex+delta;


}







/* */

void r_reset_active_and_set_tca(SHORTMATRIX *flow,DOUBLEMATRIX *tca,SHORTMATRIX *active,
			long x,long y, long a,long b)

{

long punto[2];
double diff;


diff=tca->element[x][y];
//Starting from the old flow direction and going downstream, put to 0 all the 1 in the active sites matrix

punto[0]=x;
punto[1]=y;

go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
while(active->element[punto[0]][punto[1]]==1 && active->element[punto[0]][punto[1]]!=10){
   tca->element[punto[0]][punto[1]]-=diff;
   active->element[punto[0]][punto[1]]=0;
		go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
}


active->element[x][y]=0;
//Starting from the new chosen direction go downstream, put to 0 all the 1 in the active sites matrix and correct the contributing area

punto[0]=a;
punto[1]=b;


while(active->element[punto[0]][punto[1]]==1 && active->element[punto[0]][punto[1]]!=10){
   tca->element[punto[0]][punto[1]]+=diff;
```

```
   active->element[punto[0]][punto[1]]=0;
            go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
}

}




/*  */

void reset_active(SHORTMATRIX *flow,SHORTMATRIX *active, long x,long y,
long a,long b)
{

long punto[2];




punto[0]=a;
punto[1]=b;
//Starting from the new flow direction go downstream and put to 0 all the 1 in the active sites matrix
while(active->element[punto[0]][punto[1]]==1 && flow->element[punto[0]][punto[1]]!=10){
                        active->element[punto[0]][punto[1]]=0;
                        go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
                        }

punto[0]=x;
punto[1]=y;
go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
//Starting from the old flow direction go downstream and put to 0 all the 1 in the active sites matrix

while(active->element[punto[0]][punto[1]]==1 && flow->element[punto[0]][punto[1]]!=10){
                        active->element[punto[0]][punto[1]]=0;
                        go_downstream(punto,flow->element[punto[0]][punto[1]],flow->nch);
                        }

active->element[x][y]=0;

}


/*-------------------------------------------------------------------------*/

short r_loopQ(long i,long j, long newi, long newj,SHORTMATRIX *m,DOUBLEMATRIX* tca)
{

long punto[2];


punto[0]=newi;
punto[1]=newj;


if(tca->element[newi][newj] < tca->element[i][j]){
        while((m->element[punto[0]][punto[1]])!=10 && tca->element[punto[0]][punto[1]] <= tca->element[i][j]) {
                        // check if a new direction is chosen which causes a loop in the drainage directions: going downstream the while{ } cross the
starting point
                        if(punto[0]==i && punto[1]==j) return 1;
                        else  go_downstream(punto,m->element[punto[0]][punto[1]],m->nch);
        }


}

// If we  have been arrived to the outlet, no drainge directionsm loops has been created
return 0;

}
```

a3.2.3 RextractOCN.c.

```
#include "turtle.h"
#include "t_alloc.h"
#include "t_io.h"
#include "t_list.h"
#include "t_random.h"
#include "t_utilities.h"
#include "t_datamanipulation.h"
#include "tensor3D.h"
#include "t_networks.h"

#define MAX_COLS_NUMBER 70

// Not used here but present somewhere else in the file. It could be elimated
// in the future

 LONGPAIR NW, N, NE;
 LONGPAIR  W, H,  E;
 LONGPAIR SW, S, SE;
 LONGPAIR *home,*work;

/**

Name: extract_ocn

\version: 0.99

\brief It estracts the networks from an OCN (or any fluidturlte DEM) according to a given area threshold.

\author: Riccardo Rigon, Matteo Convertino August 2006

Inputs: 1) The flow directions file name; 2) The contributing area file name; 3) The output file; 4) The area threshold file

Returns: A file containing the information needed to eventually draw the river network with an external program

Keywords: OCNm, Contributing area

References: I. Rodriguez_Iturbe and A. Rinaldo, Fractal River Basins, CUP 1997

Bugs: None


*/

/*--------------------------------------------------------------------------*/
int main(void )
{

long i/*! \param loop iterator */,j/*! \param loop iterator*/,index/*! \param Number of blocks in turtle files*/,indexa,flw[2],cnt/*! \param Counter */;
char *dirname/*! \param Name of the drainage direction file */,*areaname/*! \param Name of the Contributing areas File*/, *output/*! \param Name
of the o utput file */,*program="ExtractOCN"/*! \param  Name of the program: used for the search of the ExtractOCN.inpts file*/;
char *processed/*! \param Name of the files processed*/;
extern char *WORKING_DIRECTORY/*! \param Working Directory */;
double ath /*! \param Area threshold in pixels*/;

SHORTMATRIX *m=NULL/*! \param Matrix containing the drainage directions*/;
DOUBLEMATRIX  *rtca=NULL/*! \param Matrix of contributing areas*/;
FLOATVECTOR *U=NULL/*! \param Novalues for the drainage directions*/,*V=NULL/*! \param ??? */,*W=NULL/*! Novalues for the
contributing areas \param */,*Z=NULL/*! \param ??? */;

FILE *dirfile/*! Drainage directions file pointer */,*areafile/*! Contributing area file pointer */,/*! Output file pointer */*ostream;


printf("\nEXTRACT OCN extracts an OCN network according to a given area threshold\n");
printf("\n and produces a file which can be read by other programs to draw the network.\n");
printf("\nThe contributing area is assumed to be  a double precision number\n");
printf("Copyright 2006 by Riccardo Rigon and Matteo Convertino under GPL\n\n");

// INITIALIZATION

WORKING_DIRECTORY=get_workingdirectory();
```

```
printf("ENTER THE FLOW DIRECTIONS FILE NAME: \n");
dirname=get_filename(WORKING_DIRECTORY,program);
dirfile=t_fopen(dirname,"r");
read_comment(dirfile,TEST,MAXBUFFERSIZE,PRINT);
index=read_index(dirfile,PRINT);
U=read_floatarray(dirfile,PRINT);
W=read_floatarray(dirfile,PRINT);

printf("ENTER THE TCA FILE NAME: \n");
areaname=get_filename(WORKING_DIRECTORY,program);
areafile=t_fopen(areaname,"r");
read_comment(areafile,TEST,MAXBUFFERSIZE,PRINT);
indexa=read_index(areafile,PRINT);
if(indexa!=index) t_error("INPUT FILES NOT MATCHING\n");
V=read_floatarray(areafile,PRINT);
Z=read_floatarray(areafile,PRINT);



printf("ENTER THE OUTPUT DIRECTIONS FILE: \n");
output=get_filename(WORKING_DIRECTORY,program);
ostream=t_fopen(output,"w");




m=read_shortmatrix(dirfile,"a",PRINT);
processed=join_strings(join_strings("\n",dirname),join_strings("\n",areaname));
printf("ENTER A MINIMUM AREA THRESHOLD\n");
scanf("%lf",&ath);
rtca=read_doublematrix(areafile,"a",PRINT);

// RUN

// It counts the points above the area threshold value.
cnt=0;
for(i=1;i<=rtca->nrh;i++){
        for(j=1;j<=rtca->nch;j++){
                if(m->element[i][j]!=0 && m->element[i][j]!=9 && rtca->element[i][j]>ath) {
                        cnt++;
                }
        }
}

printf("Selected pixels are: %ld\n",cnt);

// The operation above is necessary to know how many rows there are in the output file.
write_turtle(ostream,program,processed);
fprintf(ostream,"index{2,XY}\n");
fprintf(ostream,"1: float array pixel sizes and threshold {%f,%f,%f}\n",U->element[1],U->element[2],ath);
fprintf(ostream,"2: double matrix channel network coordinates {%ld,%d}\n",cnt-1,5);

//For any row, and
        for(i=1;i<=m->nrh;i++){
// For any column,
                for(j=1;j<=m->nch;j++){
                        if(m->element[i][j] < 9 && rtca->element[i][j] > ath){
// starting from any point inside the basin,
                                flw[0]=i;
                                flw[1]=j;
// and going downstream,
                                go_downstream(flw,m->element[flw[0]][flw[1]],m->nch);
//print the networks characteristics
                                fprintf(ostream,"%ld\t%ld\t%ld\t%ld\t%f\n",j,m->nrh-i+1,flw[1],m->nrh-flw[0]+1,rtca->element[i][j]);
                        }
                }
        }

 // FINALIZE AND EXIT

free_doublematrix(rtca);
free_shortmatrix(m);
free_floatvector(U);
free_floatvector(V);
free_floatvector(Z);
t_fclose(areafile);
```

```
printf("END OF PROCESSING");

getchar();
getchar();

return 1;

}
```