A cellular automata land use model for the R software environment (Description and user documentation)

Richard Hewitt, Jaime Díaz Pacheco and Borja Moya Gómez

Contents

1. Introduction

SIMLANDER stands for SIMulation of LAnd use changE using R, and is a prototype Cellular Automata (CA) land use model built for the R software environment. It is a completely free experimental system without any guarantee which you are invited to play with to your heart's content. In fact, it's not really a system, just a collection of simple routines for the R software environment collected into a script. If you do find it useful, we'd appreciate it if you could reference us in your work:

Hewitt, R., Díaz Pacheco, J. and Moya Gómez, B., (2013), A cellular automata land use model for the R software environment, https://simlander.wordpress.com.

Please go to https://simlander.wordpress.com/ to download the latest version of SIMLANDER

See end of this document for updates on new versions

Version documentation

version 1.0.1 – simple main script, finished August 2013
version 1.0.2 – altered inner loop as per recoommendations from Graeme Hill and corrected random code February 2014
version 1.0.3 – corrected demand allocation problem May 2014
version 1.0.4 – modified land allocation to allocate all land anew at each timestep May 2014
version 1.0.5 – some minor changes and tidying up, October 2016-March 2017

2. Quick Start Documentation

2.1 How to run the simulation

1. Copy the " R-working-dir" folder to a location on your computer where you want to work.
2. Open an R console. (Rcmdr isn't necessary).
3. Install the R packages "raster", "aod", and "sp":

```
install.packages("raster")
library(raster, pos=4)
install.packages("aod")
library(aod, pos=4)
install.packages("sp")
library(sp, pos=4)
```

4. Set your R working directory to wherever you have copied the R-working-dir folder, in my case it was as follows:

```
setwd("/media/hd_sdb1/richard/SIMLANDeR/R_CA_model/R-working-dir")
```

(the path may require some tweaking to work on a windows system, see note below)

5. Run, from the R console (DON'T run this from Rcmdr, you won't see the progress bar!)

```
source("SIMbasic.R")
```
#note SIMbasic.R is now "SIM1.0.5.R" in the latest version.

If all is fine, you will have a pause of about 3 minutes where the console window does nothing. Then, a progress bar will appear, at "0" percent.

If it doesn't do anything at all, the most probable cause is that the working directory is not set correctly, and for this reason, the "SIMbasic.R" script file can't be found.
Check with getwd()

```
> source("SIMbasic.R")
Loading required package: sp
raster 2.0-12 (1-September-2012)
  |                                                                      |   0%
```

After a while (about 2 minutes in my machine), the progress bar will begin to update the progress of the simulation.

```
> source("SIMbasic.R")
Loading required package: sp
raster 2.0-12 (1-September-2012)
  |===                                                                   |   5%
```

The whole simulation, on my laptop, which is not particularly powerful, took about 40 minutes to simulate land use change over 43 timesteps (1956-1999).

When the simulation is finished, type ls() on the command line to see the various maps (one for each year) and the final calibration date, simlu99.

```
rich@rich-Satellite-L500: ~
File  Edit  View  Search  Terminal  Help
  |======================================================================| 100%
> ls()
 [1] "a"                 "acc"               "accessibility"
 [4] "ch56_99"           "change56_99"       "distriv"
 [7] "distriv_null"      "distroad"          "distroad_null"
[10] "elev"              "elev_null"         "fun"
[13] "fun3"              "funacc"            "i"
[16] "j"                 "landuse1957"       "landuse1958"
[19] "landuse1959"       "landuse1960"       "landuse1961"
[22] "landuse1962"       "landuse1963"       "landuse1964"
[25] "landuse1965"       "landuse1966"       "landuse1967"
[28] "landuse1968"       "landuse1969"       "landuse1970"
[31] "landuse1971"       "landuse1972"       "landuse1973"
[34] "landuse1974"       "landuse1975"       "landuse1976"
[37] "landuse1977"       "landuse1978"       "landuse1979"
[40] "landuse1980"       "landuse1981"       "landuse1982"
[43] "landuse1983"       "landuse1984"       "landuse1985"
[46] "landuse1986"       "landuse1987"       "landuse1988"
[49] "landuse1989"       "landuse1990"       "landuse1991"
[52] "landuse1992"       "landuse1993"       "landuse1994"
[55] "landuse1995"       "landuse1996"       "landuse1997"
[58] "landuse1998"       "landuse1999"       "lu03"
[61] "lu07"              "lu56"              "lu99"
[64] "model_nhood"       "model_random"      "model_suitability"
[67] "model_TTP_2"       "n"                 "newdata"
[70] "nhood"             "pb"                "rainfall"
[73] "rain_null"         "random"            "ranones"
[76] "ranstudy"          "roadsco"           "simlu99"
[79] "slope"             "slope_null"        "slope_suit"
[82] "T1"                "Tchange"           "temper"
[85] "temp_null"         "total"             "TTP"
[88] "w"                 "weibull"           "x"
```

At the moment the maps can only be viewed inside R with `plot("landuse1957")`

It would be very easy to ammend the script both to output .ascii or .grd files, but I haven't done it yet. As you see (file"animationlu56-99.gif") I have created an animation from SIMbasic.R (but your simulation will be slightly different of course because of the random factor). I did this exporting each map by hand and then animating them using a very simple command-line linux program called imagemagick. If you look at the animation you will see that the model clearly works as it should do, but is producing quite clumped patterns. Should be easy enough to adjust by playing with the neighbourhood weights (variable "w") and the random exponent.

To write out the rasters themselves, use:
rf <- writeRaster(landuse1999, format="ascii", filename="sim1_lu99.asc")

In the other simulation SIMhighrandom.R I have set the random exponent to 0.9. This produces a very different effect, but still relatively clumped, because of the neighbourhood effect.

Ideally, this operation would be done from the script, or even, from a separate script, so the user doesn't have to export all the maps as images and create his own animation. Again, this would be a very simple modification to the script.

3. Full Documentation

3.1 Launch R

3.2 Install the required libraries and packages.

install raster library
install aod library

```
install.packages("raster")
library(raster, pos=4)
install.packages("aod")
library(aod, pos=4)
```

IMPORTANT NOTE FOR WINDOWS USERS:
R gets confused if you use a path in your code like

c:\mydocuments\myfile.txt

This is because R sees "\" as an escape character. Instead, use

c:\\my documents\\myfile.txt
c:/mydocuments/myfile.txt

Either will work.

From http://www.statmethods.net/interface/workspace.html

3.3 Set the R Workspace

```
getwd()  print the current working directory - cwd
ls()  list the objects in the current workspace
```

```
setwd(mydirectory)  change to mydirectory
setwd("c:/docs/mydir")  note / instead of \ in windows
setwd("/usr/rob/mydir")  on linux
```

3.4 Bring in all the raster data

Data must be in ESRI ascii format, with the same number of columns and rows and the same cellsize (100 in this case)

First the land use maps. They are binary, with 1 representing urban land, and 0 representing non-urban land. The whole of the Doñana natural area, which is protected with urban development strictly restricted, is cut out and stored as "no data"

if you haven't set a working directly, bring it in from its location elsewhere:

```
lu56 <
raster("/media/hd_sdb1/richard/PhD/change_analyst/R_logistic_regression_model/lu56.
asc")
```

If its already in your R workspace, it's much easier:

```
lu56 <- raster("lu56.asc")
lu99 <- raster("lu99.asc")
lu03 <- raster("lu03.asc")
lu07 <- raster("lu07.asc")
```

have a look at them:

```
plot(lu56)
plot(lu99)
plot(lu03)
plot(lu07)
```

bring in some maps to use for calculating accessibility and suitability.
---
distriv - euclidean distance from water courses
distroad - euclidean distance from roads
elev - elevation
rainfall - Average of mean annual precipitation between 1971 and 2000
slope - slope value in degrees
temp - mean annual temperature in celsius

```
distriv <- raster("distriv.asc")
distroad <- raster("distroad.asc")
elev <- raster("elevation.asc")
rainfall <- raster("precip_71_00.asc")
slope <- raster("slope.asc")
temper <- raster("tempmedan.asc")

plot(distriv)
plot(distroad)
plot(elev)
plot(rainfall)
plot(slope)
plot(temper)
```

At present, zoning is included in the land use maps. Areas that are inside the natural protected area have null values (NA). They are outside the map, so no transitions can be calculated there.

3.5 Calculate a change map (not necessary, but interesting and a useful check on data)

Overlay maps for the calibration dates (initial map = 1956, 1st calibration map= 1999) to produce a change map between the two dates.

```
change56_99 <- lu99-lu56
```

plot it to make sure it looks ok

```
plot(change56_99)
```

3.6 Clean up the data a bit (remove unwanted values)

a weird error somewhere generates a few cells of value -1. This is no good, because land use change between 1956 and 1999 has to be a binary response variable (e.g. 1= change, 0=no change)

```
fun <- function(x) { x[x<0] <- NA; return(x) }
ch56_99<-calc(change56_99, fun)

summary(ch56_99)
```

just to check that we have successfully removed the -1 values from the map and made them No-Data (NA) instead.

Ok, here we realised that we hadn't created null values in the variable maps where the protected areas should be, just on the change map. In some of the forums people say it shouldn't really matter. I got rid of them anyway, just to be on the safe side.

```
distroad_null <- mask(distroad,ch56_99)
distriv_null <- mask(distriv,ch56_99)
elev_null <- mask(elev,ch56_99)
rain_null <- mask(rainfall,ch56_99)
slope_null <- mask(slope,ch56_99)
temp_null <- mask(temper,ch56_99)
```

If we wanted to do regression operations or some other kind of mathematical/statistical operation on the maps, we can extract the values from the map using the getValues command (and then return them later). We don't need this now, but here's how to do it anyway (for a logistic regression, for example). Once the operation has been carried out, we can just return the results back to the map.
as follows:

```
chgevals <-getValues(ch56_99)

summary(chgevals)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
      0       0       0       0       0       1  599750

distroadvals<-getValues(distroad_null)
distrivval<-getValues(distriv_null)
elevval<-getValues(elev_null)
slopeval <-getValues(slope_null)
rainval<-getValues(rain_null)
tempval<-getValues(temp_null)
```

3.7 Applying the neighbourhood rule to the initial map

Let *w* be a moving window filter of dimensions 5x5 cells (500 x 500 metres here).

```
w <- matrix(c(0,0,50,0,0,0,50,50,50,0,50,50,500,50,50,0,50,50,50,0,0,0,50,0,0),
nr=5,nc=5)

w
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0   50    0    0
[2,]    0   50   50   50    0
[3,]   50   50  500   50   50
[4,]    0   50   50   50    0
[5,]    0    0   50    0    0


n <- focal(lu56, w=w)
plot(n)
```

This command generates a neighbourhood transition potential map from lu56. Have a look at this useful webpage:

http://scrogster.wordpress.com/2012/10/05/applying-a-circular-moving-window-filter-to-raster-data-in-r/

But it has the unwanted side effect of increasing the width of the nodata areas, thus massively increasing the size of the feature land use classes and the zoned area. We seriously don't want this. BUT, luckily, its easily fixed:

```
nhood <- cover(n, lu56)
```

The cover operation (see Raster package manual) fills the null values in the first map with the values in the second, so now we have a final neighbourhood potential map for the transition period 1956-57

```
plot(nhood)

zoom(nhood)

model_nhood <- nhood   #rename to model_nhood so we know this is our final
neighboorhood potential map.
```

Its a good idea to list what we've got in the workspace and delete anything we don't want:

```
ls ()
[1] "lu03"        "lu07"        "lu56"        "lu99"        "model_nhood"
[6] "n"           "nhood"       "w"
```

to remove something, just enter:

```
remove(map)
```

We can plot the map and zoom around to see what the result it. Looks good so far. Just plot the map again and it will zoom back to the original extent.

Notes on the neighbourhood rule, 14/02/2014

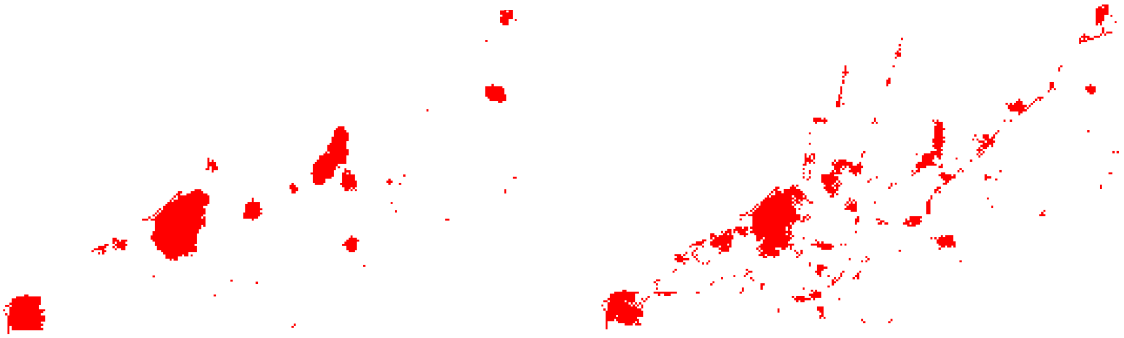
if we replace

```
w <- matrix(c(0,0,50,0,0,0,50,50,50,0,50,50,500,50,50,0,50,50,50,0,0,0,50,0,0),
nr=5,nc=5)
```

with:

```
w <-
matrix(c(0,0,0.5,0,0,0,0.5,0.5,0.5,0,0.5,0.5,5,0.5,0.5,0,0.5,0.5,0.5,0,0,0,0.5,0,0)
, nr=5,nc=5)
```

The other parameters (e.g. accessibility, suitability and randomness) play a much larger part in determining the areas that are most likely to change. In this case, it makes for a much more realistic simulation. Compare:



At left, neighbourhood rules as normal. At right, neighbourhood rules adjusted downwards by a factor of 100.

3.8 Generating the parameter **v,** *a skewed random distribution (Weibull).*

In Matlab:

http://www.taygeta.com/random/weibull.xml

we read that:

WeibullRandomNumbers = scale.*(-log(1-rand(noOfRandomNumbers,1))).^(1/shape)

at http://www.taygeta.com/random/weibull.xml it is explained that to use the weibull distibution first we need to generate a random number, x, which comes from a uniform distribution (in the range from 0 to 1)
this is very easy, we just use runif for one number, accepting the defaults, which
returns what we want, a random number between 0 and 1

```
x <-runif(1)
x
```

OK, here's how it worked in R:

```
#begin random block

x <-runif(862099) #because there are 862099 cells in the map

weibull <- 1+(-log(1-(x)))*exp(1/2)

funselect <- function(x) { x[x!=0] <- NA; return(x) } #extract only vacant
```

```
areas (value 0) so that existing functional land uses are not    randomized. Set
everything else to NA

      vacants <- calc(T1, funselect)  #BE CAREFUL! NEED TO TAKE VACANTS FROM T1,
NOT LU56

      random <- lu56 #just copying lu56 as a skeleton map

      values(random)<-weibull

      model_random <- mask(random, vacants) #generating a mask to apply NAs to the
random layer.

      model_random <- cover(model_random,T1) #cover to fill the NAs back in with
the original values from T1.

      #end random block

      #-----------------------
```

NB: this random block replaces the original one in the script version 1.0.0

3.9 Generating an accessibility map

Now we need to reclassify the two accessibility maps. At the moment they are just simple distance maps, so the closer each cell is to a road or a river, the lower the value. But, for accessibility, we need precisely the opposite. We can obtain this using White et al's (1997) local accessibility equation:

$$a_j = 1 + \left(\frac{D}{\delta_j}\right)^{-1}$$

(Eq. 2)
(from White *et al* 1997)

where D is the euclidean distance from the cell to the nearest cell in the network, and δ_j is the coefficient expressing the importance of accessibility for the desirability of the cell for land activity *j*; a_j is then inserted as a coefficient in equation 2 (see above).

To produce the coefficients, we need to reclassify the accessibility map

```
summary(distroad_null)

Min.         0
1st Qu.    424
Median    1118
Mean      1966
3rd Qu.   2326
Max.     13710
NA's    600147

roadsco <- reclassify(distroad_null,c(-Inf,100,1,100,300,0.9,300,800,0.8,
```

```
800,1500,0.7,1500,2500,0.6,2500,5000,0.5,5000,8000,0.4,8000,10000,0.3,10000,12000,0
.2,12000,Inf,0.1))
```

This gives a reasonably dispersed distance effect with a halving distance of 2.5km.

```
accessibility <- 1+((distroad_null/roadsco)^-1)
```

Works fine. Note that areas where roads already exist (distance 0) take unmeasurably large values, (Inf) (and therefore appear white). This should be no particular problem, it will just mean that all urban cells will try to sit on top of roads. Since roads are not explicitly modelled, this isn't a problem.

If we wanted the maximum value in this map to be 2, the same as at distance 1, we can set it with a map calculation:

```
funacc <- function(x) { x[x>2] <- 2; return(x) }
acc<-calc(accessibility, funacc)
```

Thus every cell which had a value greater than 2 gets a value of 2.

```
model_accessibility <- accessibility     #change the name to match the others we
will use for the Transition Potential computation.
```

3.10 Generating a suitability map.

We are going to use use slope, and simply reclassify it so that the lowest slopes are the most suitable

```
slope_suit <- reclassify(slope_null,c(-
Inf,0,1,0,5,0.9,5,10,0.8,10,15,0.5,15,20,0.3,20,Inf,0.1))
plot(slope_suit)
model_suitability <- slope_suit
```

3.11 Calculating transition potential

Now we can calculate the total transition potential for the first step, 1956-57

```
model_TTP <-
(model_accessibility)*(model_suitability+1)*(model_nhood+1)*(model_random)
```

Note that we added 1 already to accessibility and random.

This TTP plot contains infinite values, maybe we don't want these. To get rid of them, just recompute using "acc" instead of model_accessibility

```
model_TTP_2 <- (acc)*(model_suitability+1)*(model_nhood+1)*(model_random)
```

Ok, now we need to calculate a new land use map for 1957, on the basis of the TTP we have just computed. So, first we need to know the demand.

```
freq(ch56_99, value=1)
```

```
value  count
[1,]     0 255597
[2,]     1   6752
[3,]    NA 599750
```

So we know that we have to locate 6752 cells between 1956 and 1999, that's 157 cells at each timestep!

3.12 Putting it all together

Now we just need to use a couple of loops to join all this together!

Here's the model design:

SIMLANDeR model design

1. input layers lu initial (lu56), land use first calib (lu99). Land use cats are 0 urban, everything else 1, except features and zoning, which is NA
2. cross these two layers to produce change map.
3. demand cells can be queried with freq(change map)
4. calculate suitability and accessibility maps
5. T1 <- copy of lu56
5. Begin loop to iterate 43 times
    ---------- take T1 and apply nhood rules to create model-nhood
    ----------create random map, with urban filled as 1 and everything else randomised.
    ----------calculate TTP from N,A,S + R
    ----------extract values as TTP
  ----------fill NA values with zeroes
        (Implement G.Hill's modification)
      select top 157 values for replacement
      find the cut-off
        all values not -9999 take value 1
        newchange <- fill skeleton map lu56 with new values
        T+1change <- newchange + T1
        T1 <- T+1change <- replace all values in T+1change with 1 (urban)
6. End Loop
7. Print "finished after" & i & "transitions"
8. simlu99 <- T1
END

And here's the final script (version 1.0.5)

```
#begin script
#set workspace and libraries
#setwd("/media/hd_sdb1/richard/SIMLANDeR/R_CA_model/R-working-dir")
require(raster)
#begin import data
lu1 <- raster("lu56.asc")
```

```
lu2 <- raster("lu99.asc")
lu3 <- raster("lu03.asc")
lu4 <- raster("lu07.asc")
distriv <- raster("distriv.asc")
distroad <- raster("distroad.asc")
elev <- raster("elevation.asc")
rainfall <- raster("precip_71_00.asc")
slope <- raster("slope.asc")
temper <- raster("tempmedan.asc")
#end import data
#-----------------------
#begin tidy up a bit - calculate a change map, set null values for features and
zoned areas
change56_99 <- lu2-lu1
fun <- function(x) { x[x<0] <- NA; return(x) }
ch56_99<-calc(change56_99, fun)
distroad_null <- mask(distroad,ch56_99)
distriv_null <- mask(distriv,ch56_99)
elev_null <- mask(elev,ch56_99)
rain_null <- mask(rainfall,ch56_99)
slope_null <- mask(slope,ch56_99)
temp_null <- mask(temper,ch56_99)
#end tidy up
#-----------------------
#begin calculate accessibility
roadsco <- reclassify(distroad_null,c(-
Inf,100,1,100,300,0.9,300,800,0.8,800,1500,0.7,1500,2500,0.6,2500,5000,0.5,5000,800
0,0.4,8000,10000,0.3,10000,12000,0.2,12000,Inf,0.1)) #setting the coefficients
accessibility <- 1+((distroad_null/roadsco)^-1)
funacc <- function(x) { x[x>2] <- 2; return(x) }   #to make sure we do not have
infite values at distance 0
acc<-calc(accessibility, funacc)
#end calculate accessibility
#-----------------------
#begin calculate suitability
slope_suit <- reclassify(slope_null,c(-
Inf,0,1,0,5,0.9,5,10,0.8,10,15,0.5,15,20,0.3,20,Inf,0.1))
model_suitability <- slope_suit
#end calculate suitability
#-----------------------
#set weights matrix for neighbourhood
w <- matrix(c(0,0,50,0,0,0,50,50,50,0,50,50,500,50,50,0,50,50,50,0,0,0,50,0,0),
nr=5,nc=5)
#-----------------------
#preparations for simulation
T1 <- lu1  #set T1 to first map lu1. Change this if we want to start from a
different map.
print (paste("simulation starting at ", Sys.time(), sep=""))
startyear <- 1956
endyear <- 1999
total <- (endyear)-(startyear)
#------------------------------calculate demand from lu1 and lu2 (must have 2 rows
and 2 columns with urban land in row 2, column 2)
dflu1 <- as.data.frame(freq(lu1))
dflu2 <- as.data.frame(freq(lu2))
```

```
urbdemand <- as.numeric(dflu1[2,2])
finaldemand <- as.numeric(dflu2[2,2])
andem <- (finaldemand - urbdemand)/total
andem <- round(andem,0)
#------------------------------end calculate demand from lu1 and lu2
print("set demand, starting timer..")
ptm <- proc.time()
#end preparations
#-----------------------
pb <- txtProgressBar(min = 0, max = total, style = 3)
for (i in 1:total){    #begin running simulation
urbdemand1 <- urbdemand + andem*i
      #begin neighbourhood block
      n <- focal(T1, w=w)
      nhood <- cover(n, T1)
      model_nhood <- nhood
      #end neighbourhood block
      #-----------------------
      #begin random block
      x <-runif(ncell(T1)) #corrected to calculate directly from number of cells in
the map
      weibull <- 1+(-log(1-(x)))*exp(1/2)
      funselect <- function(x) { x[x!=0] <- NA; return(x) } #extract only vacant
areas (value 0) so that existing functional land uses are not    randomized. Set
everything else to NA
      vacants <- calc(T1, funselect)  #BE CAREFUL! NEED TO TAKE VACANTS FROM T1,
NOT LU56
      random <- lu1 #just copying lu1 as a skeleton map
      values(random)<-weibull
      model_random <- mask(random, vacants) #generating a mask to apply NAs to the
random layer.
      model_random <- cover(model_random,T1) #cover to fill the NAs back in with
the original values from T1.
      #end random block
      #-----------------------
      #begin transition potential calculation
      model_TTP <- (acc)*(model_suitability+1)*(model_nhood+1)*(model_random)
      fun <- function(x) { x[is.na(x)] <- 0; return(x)}
      TTP <- calc(model_TTP, fun)
      #Select the highest 164 values from the TTP map
      #r1demand <- 164  #this is the number of cells we want to allocate from r1
      x <- as.matrix(TTP)
      n <- urbdemand1  #the demand calculated as above
      x2 <-  sort(-x, partial = n)
      x2h <-  -sort(x2[1:n])
      ix <- which(x %in% x2h)
      rowsT1 <- nrow(T1)
      colsT1 <- ncol(T1)
      ro <- ix %% rowsT1
      ro <- ifelse(ro == 0L, rowsT1, ro)
      co <- 1 + ix %/% rowsT1
      x3 <- x[ix]
      d <- data.frame(row = ro, col = co, x = x3)
      result <- d[rev(order(d$x)), ]
      #-----------------------
```

```
      #test for duplicates that inflate the number of cells allocated
      difftrans <- (length(result$x)-n)
            if (difftrans > 0) {
            result2 <- head(result,-difftrans) #remove the duplicates from the end
of the file (the weakest candidate cells)
            result <- result2
            }
      #turn selected n values in the dataframe into a matrix and then to a raster
      x.mat <- matrix(0, rowsT1, colsT1) #create a matrix with the right number of
rows and columns and fill with 0 values
      #x.mat[cbind(result$row,result$col)] <-result$x   #works just fine.
      x.mat[cbind(result$row,result$col)] <-1   #but actually we want values to be
1 (urban land), not the TP value.
      #which(!is.na(x.mat), arr.ind =TRUE)  #pick out the non-NA values. Useful for
testing that this has actually worked.
      r <- raster(x.mat)
      extent(r) <- extent(T1)
      newdata <- r
      newdata <- mask(newdata, lu1) #generating a mask to apply NAs to the final
map layer.
      T1 <- newdata  #directly allocated all the cells at their most favourable
locations according to TTP

      filen <- paste("lu", (startyear + i), ".png", sep="")
      #filenasc <- paste("lu", (startyear + i), ".asc", sep="")
      #LUout <- writeRaster(T1, filename=(filenasc), format="ascii",
overwrite=TRUE)
      #filen <- paste(filen,".png", sep="")
      png(filename = paste(filen),width = 1200, height = 1200, bg="white") #Plot
each land use map to be able to make an animation.
      #plot(T1)
      #plot(T1,breaks=breakpoints,col=colors,main=(startyear+ii))
      plot(T1,main=(startyear+i))
      dev.off()

      removeTmpFiles(h=5)
      Sys.sleep(0.1)
      setTxtProgressBar(pb, i)
      print (paste("FINISHED: landuse simulation for ", (startyear + i), sep=""))
      print(proc.time() - ptm)
}
close(pb)
png(filename = paste("lu",startyear,".png",sep=""),width = 1200, height = 1200,
bg="white") #Plot initial map also to animations file
plot(lu1, main = paste("initial land use map - ",startyear,sep=""))  #lu1 for
calibration, lu2 for simulation phase
dev.off()
#eval_anim <- paste("system ('convert -delay 200 -loop 0 *.png
lu",startyear,"_",endyear,".gif')",sep="")   #creates an animated gif using
imagemagick, comment back in if you are on a linux system and have imagemagick
installed
#eval(parse(text=eval_anim))  #comment back in if you are on a linux system and
have imagemagick installed
#simlu2011 <- T1
sim99 <- T1
```

```
sss <- stack(lu2,sim99)
#name1 <- paste("initial land use ",startyear,sep="")
name1 <- paste("real land use ",endyear,sep="")
name2 <- paste("simulated land use ",endyear,sep="")
names(sss) <- c(name1,name2)
print(plot(sss)) #without the print command, the output will not appear on the
screen
print (paste("simulation complete at ", Sys.time(), sep=""))
#print(plot(sss,breaks=breakpoints,col=colors))   #same as above, but just in case
you have colors and breakpoints defined
#END OF SCRIPT--
```

4. Tips and tricks

4.1 Cleaning out the workspace

```
rm(list = ls()[grep("^x", ls())])

remove everything in the workspace that begins with "X"

rm(list = ls()[grep("^*", ls())])
This cleans the directory out completely.
```

4.2 Using patch statistics to compare simulations and real maps

Install SDMTools:

```
install.packages("SDMTools")

ps.data = PatchStat(lu56)

> ps.data

  patchID n.cell n.core.cell n.edges.perimeter n.edges.internal   area

1       0 261553      231791             25398          1020814 261553
2       1   1411         270              2032             3612   1411

  core.area perimeter perim.area.ratio shape.index frac.dim.index

1    231791     25398        0.0971046    12.41349       1.403857
2       270      2032        1.4401134    13.36842       1.718267

  core.area.index

1       0.8862104
2       0.1913536
```

Analyze the real maps:

```
 ps.data07 = PatchStat(lu07)

> write.csv(ps.data07, file="patch2007.csv", na='NA')
```

The calibrations:

```
ps.data_sim1_99 = PatchStat(sim1_lu99)

> write.csv(ps.data_sim1_99, file="patch_sim1_99.csv", na='NA')
> ps.data_sim2_99 = PatchStat(sim2_lu99)
> write.csv(ps.data_sim2_99, file="patch_sim2_99.csv", na='NA')
> ps.data_sim3_99 = PatchStat(sim3_lu99)
> write.csv(ps.data_sim3_99, file="patch_sim3_99.csv", na='NA')
```

And the future simulation 2050:

```
 ps.datasimlu2050 = PatchStat(simlu2050)

> write.csv(ps.datasimlu2050, file="patch_simlu2050.csv", na='NA')
```

5. Updates

5.1 Version 1.0.3

In Version 1.03, the selection of highest values from the transition potential map has been rewritten. Instead of extracting values through a GetValues command and then sorting, which I found difficult to code properly, the data are converted to a matrix, output to a dataframe (primarily to facilitate checking but also to make it easy to write out an excel file if desired) and then passed back to matrix again (almost certainly could be done in fewer lines of code, but it does work). A new raster is then generated with the highest values as "1" (urban land). This is the replacement code:

```
#Select the highest 164 values from the TTP map
        #r1demand <- 164  #this is the number of cells we want to allocate from r1
        x <- as.matrix(TTP)
        n <- urbdemand1  #the demand, calculated by a countdown procedure, above
        x2 <-  sort(-x, partial = n)
        x2h <-  -sort(x2[1:n])
        ix <- which(x %in% x2h)
        ro <- ix %% 1151
        ro <- ifelse(ro == 0L, 1151, ro)
        co <- 1 + ix %/% 1151
```

```
x3 <- x[ix]
d <- data.frame(row = ro, col = co, x = x3)
result <- d[rev(order(d$x)), ]
#-----------------------
        #turn selected 164 values in the dataframe into a matrix and then to a raster
        x.mat <- matrix(0, 1151, 749) #create a matrix with the right number of rows and
columns and fill with 0 values
        #x.mat[cbind(result$row,result$col)] <-result$x   #works just fine.
        x.mat[cbind(result$row,result$col)] <-1   #but actually we want values to be 1 (urban
land), not the TP value.
        #which(!is.na(x.mat), arr.ind =TRUE)  #pick out the non-NA values. Useful for testing
that this has actually worked.
        r <- raster(x.mat)
        e <- extent(c(150938,225838,4070192,4185292))
        extent(r) <- e
```

As in previous versions, new land to be allocated is calculated by: (urban cells in final map-urban cells in initial map/number of years). The new land to be allocated at each time step is then added to the existing urban land from the previous step. If this is not what you want, look at version 1.0.4.

Richard Hewitt 17-may-2014

5.2 Version 1.0.4

In Version 1.04, the principal change relates to the way the land is allocated at each time step.

Instead of adding the new land (calculated by urban cells in final map-urban cells in initial map/number of years) to be allocated at each time step (e.g as in version 1.03) now the model allocates all of the urban land cells in the map at each time step, according to the results of the transition potential calculation. The total urban land in each time step is calculated by taking the cell count for urban from the initial map and then increasing the cells allocated at each time step until the final desired demand is reached.

This means that each time step is effectively a blank sheet, and that urban land can, if necessary disappear between time steps. Though there may be few cases in which this is desirable, this is much closer to the way that original CA models (like the game of life) work.

Richard Hewitt 17-may-2014

5.3 Version 1.0.5

In Version 1.05, the main changes are in the way the demand is calculated, and a lot of general tidying up. The script should be alot easier to follow and contains much less redundant code

Also added the following code to remove the over-allocation issue

```
      #test for duplicates that inflate the number of cells allocated

      difftrans <- (length(result$x)-n)

            if (difftrans > 0) {

            result2 <- head(result,-difftrans) #remove the duplicates from the end
of the file (the weakest candidate cells)

            result <- result2

            }
```

Richard Hewitt 28-March-2017