Fill Cells with aRT

Pedro R. Andrade

April 4, 2011

Contents

1 Introduction

Fill cell functions calculate attribute values for tables associated with layers of cells. The goal is to homogenize information from different sources in different formats (vector data, rasters as well as other cellular layers), aggregating them into a single spatial (and possibly temporal) data source. Different operators are available according to the geometrical representation and semantics of the input data.

In the current TerraLib version, cells are rectangular. They may have a resolution of, for instance 1m x 1m, 500m x 500m, 100km x 200km, according to the application needs. A single cellular layer may be associated to different tables, which can be static (attributes that do not change over time) or dynamic (attributes that change over time).

To execute the steps in this tutorial, some spatial data is required:

```
> require(aRT)
> con=openConn(name="default")

> db=openDb(con, "rondonia")
> lcenso     = openLayer(db, "censo")
> tcenso     = openTheme(db, "censo")
> tcenso

Object of class aRTtheme
```

```
Theme: "censo"
Layer: "censo"
View: "censo"
Number of polygons: 68
Table: "censo"
Attributes: "MSLINK", "AREA_1", "PERIMETRO_", "CODIGO",
            "NOMEMUNI", "RENDIMENTO", "NUMERO_PES",
            "RENDAPCAPI", "DENS_POP"

> lroads    = openLayer(db, "roads")
> troads    = openTheme(db, "roads")
> troads

Object of class aRTtheme

Theme: "roads"
Layer: "roads"
View: "roads"
Number of lines: 22
Table: "roads"
Attributes: "SPRPERIMET", "SPRCLASSE", "OBJET_ID_5",
            "object_id_2"

> lpoints = openLayer(db, "urban")
> tpoints = openTheme(db, "urban")
> tpoints

Object of class aRTtheme

Theme: "urban"
Layer: "urban"
View: "urban"
Number of points: 9
Table: "urban"
Attributes: "SPRROTULO", "SPRNOME", "MSLINK", "MAPID",
            "CODIGO", "AREA_1", "PERIMETRO_", "GEOCODIGO",
            "NOME", "SEDE", "LATITUDESE", "LONGITUDES",
            "AREA_TOT_G", "OBJET_ID_2", "object_id_3"
```

2 Creating layers of cells

A layer of cells is created from another database layer. There are two strategies
to create cells, depending on the type of spatial data of reference. The default
way of creating cells is from layers of polygons. The code below creates the cells
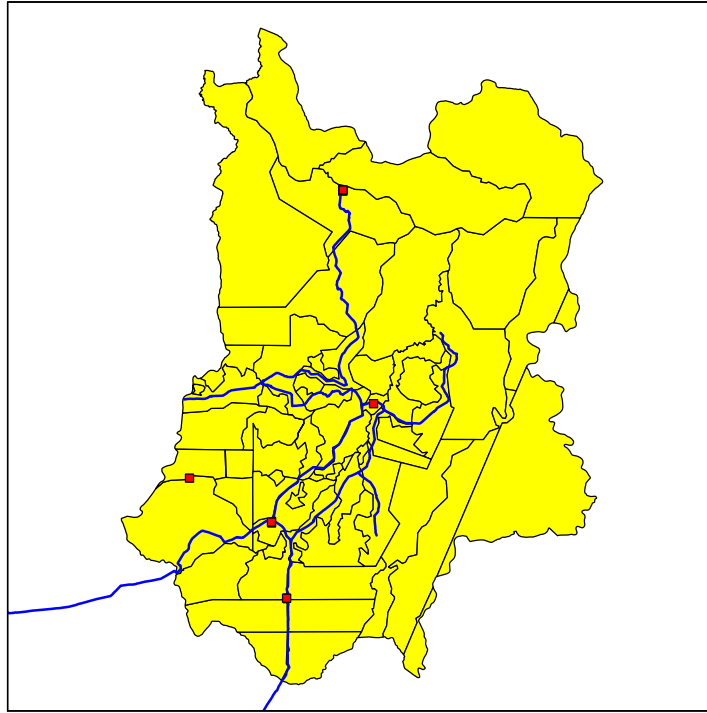shown in Figure 2.

Figure 1: Points, lines, and polygons used in this vignette.

```
> lcells = createLayer(lcenso, "cells", rx=4000)
> lcells

Object of class aRTlayer

Layer: "cells"
Database: "rondonia"
Number of cells: 824
Projection Name: "NoProjection"
Projection Datum: "Spherical"
Tables:
     "cells": static
```

The default options create squared cells from the polygons. Bounding boxes of the whole set of polygonal geometries can also be used to create the layer of cells, using the argument `all`. In the next example, we create a layer of non-squared cells by informing different resolutions for x and y:

```
> lcells2 = createLayer(lcenso, "cellsbox", rx=10000, ry=5000, all=TRUE)
```
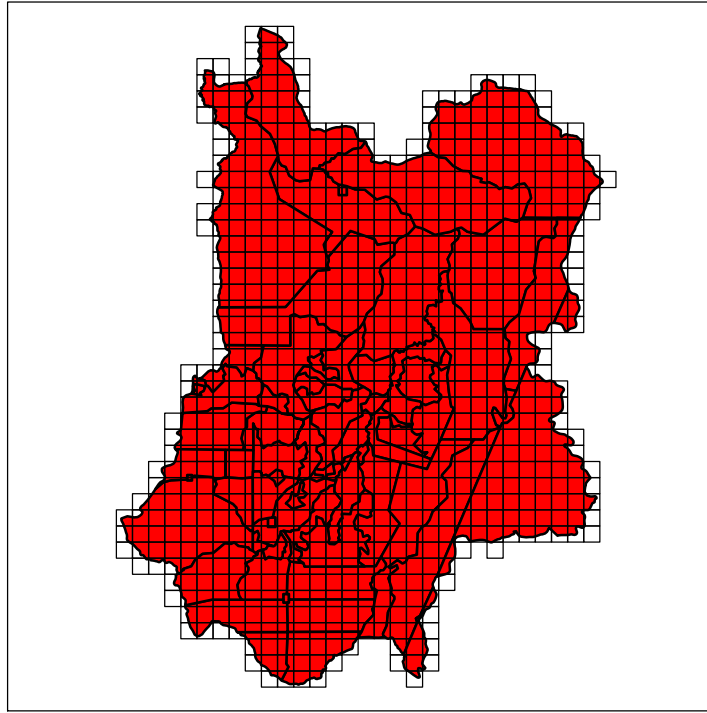
Figure 2: Cells covering the layer of polygons.

Cells created this way cover the entire area of bounding box covering the reference layer, similarly to a raster with a constant number of cells in each row and column. The result is shown in Figure 3. In both cases, a static table with the same name of the layer of cells is created automatically.

3 Fillling cell attributes

Cell attributes are created directly from the table of the layer of cells. The first step to fill cell attributes is to open the table:

```
> tcells = openTable(lcells, "cells")
> tcells

Object of class aRTtable

Table: "cells"
Type: static
Layer: "cells"
```
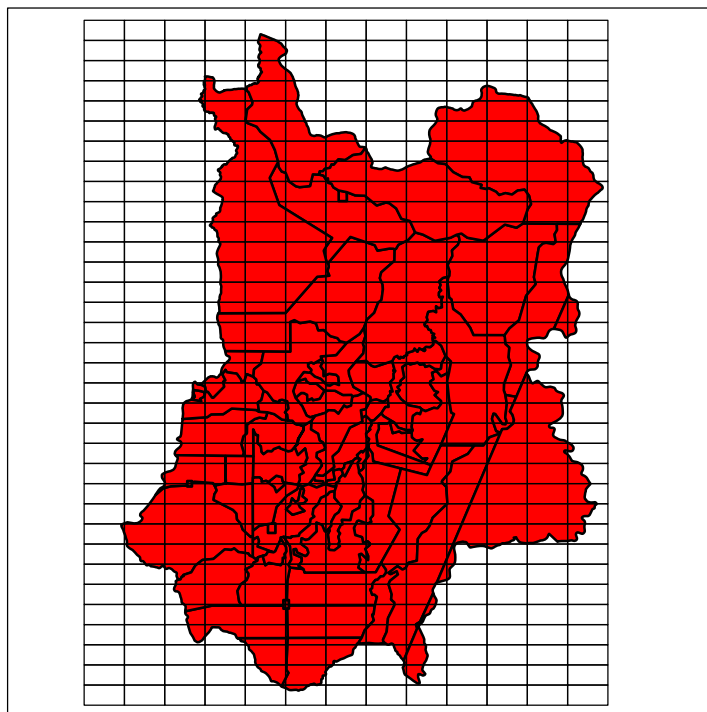
Figure 3: A layer of non-squared cells covering the bounding box of the polygons.

```
Rows: 824
Attributes:
    object_id0: character[48] (key)
    Col: integer
    Lin: integer
```

To create attributes using TerraLib functionalities, we use `createAndFill-Column()`. This function may take different arguments depending on the strategy used to fill the cells. The first argument is the table where the result will be stored, the second is the name of the attribute to be created, the third is the theme with the data used to compute the attribute, and the other depend on the operation. For example, if one wants to create an attribute based on the minimum distance from the centroid of the cells to a set of points, we can use strategy "distance".

```
> # repare aqui que, caso o tema possua apenas pontos, nao eh necessario usar o argumento 'g
> createAndFillColumn(tcells, newatt="dpoints", target=tpoints, geometry="point", strategy='
```

Operations such as distance may be used with any geometric type as parameter. However, some operations can be used with only one geometric type. For

example, the length of lines covering each cell can be created by calling:

```
> createAndFillColumn(tcells, "length_roads", target=troads, strategy="length")
```

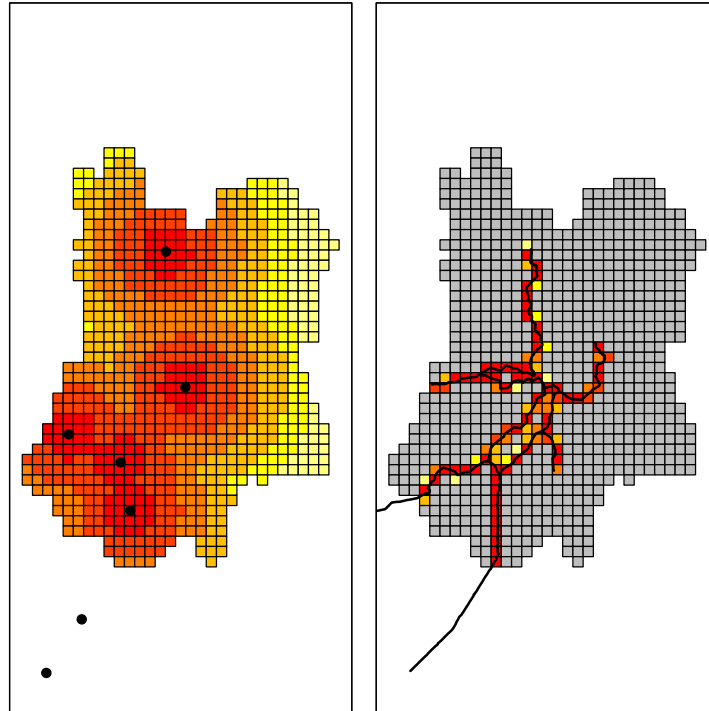The results of these two operations are shown in Figure 4.

Figure 4: Results of 'distance' and 'length' operations.

Some strategies use not only geometries but also attributes related to those geometries. To create an attribute that computes the maximum value of a given geometry that has some overlap with the cells just call:

```
> createAndFillColumn(tcells, "maxrenda", tcenso,  att="RENDIMENTO", strat="maximum")
```

Table 1, in the end of this vignette, describes each operation available for filling cells directly from `createAndFillColumn()`. Most of the strategies are intuitive, but two deserve more attention, "sumwba" and "averagewba," which mean "sum weighted by area" and "average weighted by area," respectively. The first spreads a given attribute of a set of polygons through the cells. This operation can be used for population data, because it keeps the overall sum of the data in both sets. The second operation if useful to work with attributes representing averages. The algorithm spreads averages of a set of polygons to

the cells, recomputing each average proportionally to the intersection area. The results of these operations are shown in Figure 5.

```
> createAndFillColumn(tcells, "pessoas",  tcenso,   att="NUMERO_PES", strat="sumwba")
> createAndFillColumn(tcells, "rendacap", tcenso,   att="RENDAPCAPI", strat="averagewba")
```
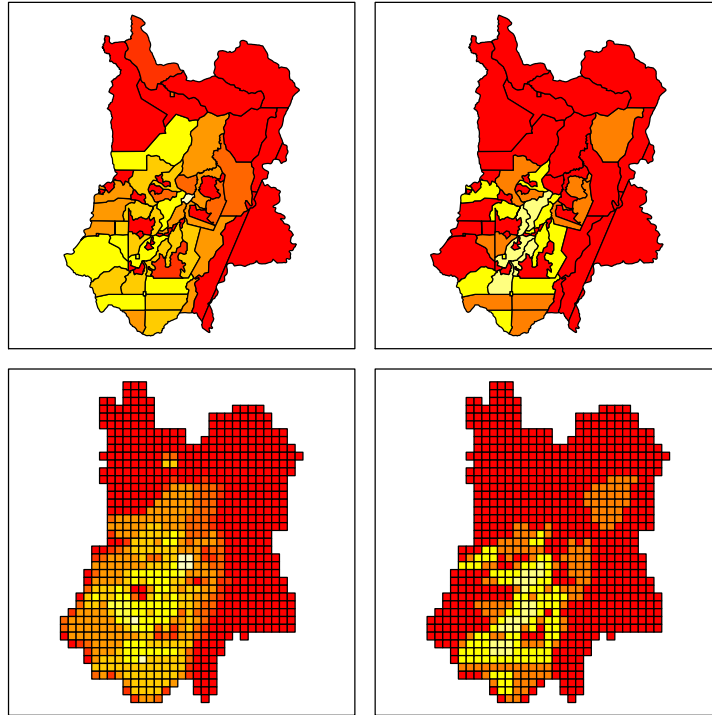

Figure 5: Results of the operations "sumwba" (left) and "averagewba" (right).

Finally, to visualize the data it is necessary to create a theme using the table that contains the attributes created.

```
> theme = createTheme(lcells, "mytheme")
> theme

Object of class aRTtheme

Theme: "mytheme"
Layer: "cells"
View: "mytheme"
Number of cells: 824
Table: "cells"
```

```
Attributes: "object_id0", "Col", "Lin", "dpoints", "length_roads",
           "maxrenda", "pessoas", "rendacap"

> cells = getData(theme)
> #polygons = getData(tcenso)
> #for(i in 1:length(d@polygons))
> #    d@polygons[[i]]@labpt = c(0,0)
>
> #d@data = d@data[,-(1:3)] # removing the attributes of type string
>
> #for(i in 1:dim(d@data)[2]) # normalizing the attributes
> #{
> #    mmax = max(d@data[,i])
> #    d@data[,i] = d@data[,i]/mmax
> #}
```

The figures plotted in this vignette use `plotColoured()`, described and used as follows:

```
> plotColoured = function(spatialdata, attribute, slices, colors)
+ {
+     df = spatialdata@data
+
+     vcolors=rep(colors[1], dim(df)[1])
+
+     for(i in 1:length(slices))
+     {
+         vcolors[which(df[,attribute] > slices[i])] = colors[i]
+     }
+     plot(spatialdata, col=vcolors)
+     box()
+ }
> plotColoured(cells, "dpoints", c(0, (1:5)*10000), heat.colors(6))
```

4 Other operations

Aditionally, it is possible to create new attributes directly using `createColumn()` and `updateColumns()`, instead of `createAndFillColumn()`. This way, you can use all R functionalities to generate attributes according to the objectives of the work. For further information on how to create attributes of tables, read the vignette "Tables and Queries With aRT," available within the package.

Table 1: Different operations available for `createAndFillColumn()`.

Operation	Description
area	Total area of overlay between the cell and a layer of polygons.
average	Average of an attribute of the objects that have some intersection with the cell, without taking into account their geometric properties.
averagewba	Average weighted by area, based on the proportion of the intersection area. Useful when you want to distribute atributes that represent averages, such as per capita income.
count	Number of objects that have some overlay with the cell (requires argument geometry).
distance	Distance to the nearest object of a chosen geometry (requires argument geometry).
length	Total length of overlay between the cell and a layer of lines.
majority	More common value in the objects that have some intersection with the cell, without taking into account their geometric properties.
maximum	Maximum value of an attribute among the objects that have some intersection with the cell, without taking into account their geometric properties.
minimum	Minimum value of an attribute among the objects that have some intersection with the cell, without taking into account their geometric properties.
percentage	Percentages of each class of a raster data. It creates one attribute for each class of the raster.
presence	Boolean value pointing out whether some object has an overlay with the cell.
stdev	Standard deviation of an attribute of the objects that have some intersection with the cell, without taking into account their geometric properties.
sum	Sum of an attribute of the objects that have some intersection with the cell, without taking into account their geometric properties.
sumwba	Sum weighted by area, based on the proportion of the intersection area. Useful when you want to preserve the total amount in both layers, such as population size.
xdistance	Approximated distance to the nearest object of a chosen geometry (requires arguments geometry and maxerror).