# Generalized Proximity Matrixes with aRT

Pedro Ribeiro de Andrade

April 18, 2011

## Contents



## 1 Introduction

This vignette describes how to create Generalized Proximity Matrixes (GPM) within aRT. GPM is based on the idea that Euclidean spaces are not enough to describe the relations that take place within the geographical space. For more information on GPM, see *Aguiar et al. (2003); Modeling Spatial Relations by Generalized Proximity Matrices. Proceedings of V Brazilian Symposium in Geoinformatics (GeoInfo'03).*

GPM is composed by a set of strategies that try to capture such spatial warp, computing operations over sets of spatial data. Some strategies have been implemented within aRT. In the next sections, we will describe the basic structure of the implementation and present some examples of creating proximity matrixes. Before starting, we will read some spatial data.

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

> db=openDb(con, "gpm")
> llotes = openLayer(db, "lotes")
> lcomunidades = openLayer(db, "comunidades")
> lrodovias = openLayer(db, "rodovias")
> rodovias    = getLines(lrodovias)
> comunidades = getPoints(lcomunidades)
> lotes       = getPolygons(llotes)
```

The database, shown in Figure 1, contains:

1. a set of lines, representing roads;
2. a set of points, representing the center of communities;
3. a set of polygons, representing farms.

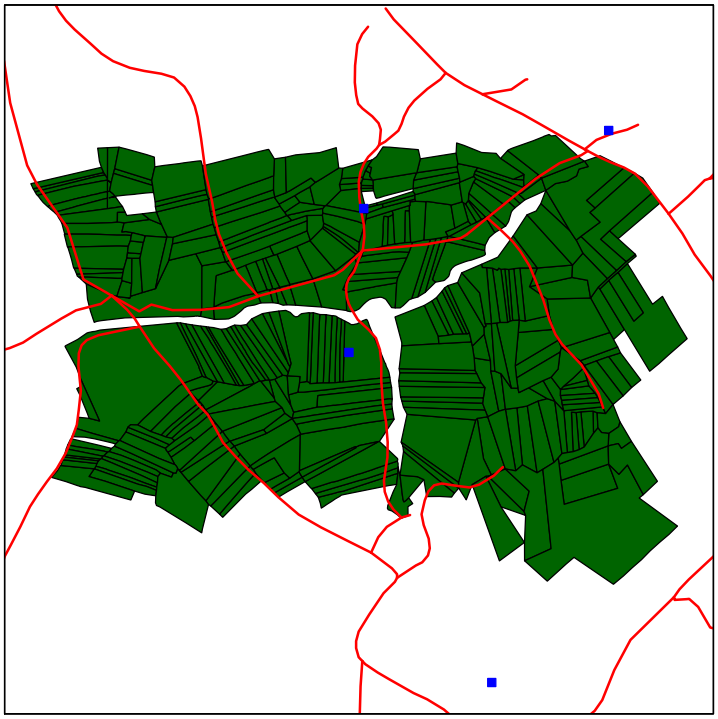

Figure 1: Database that will be used to compute spatial relations.

From the set of polygons, we create a set of points with their centroids and a layer of cells, shown in Figure 2.

```
> lcells = createLayer(llotes, "cells", 150) ## ORIGINAL IS 150
> cells  = getCells(lcells, slice=3000) # le de 3000 em 3000, fica bem mais rapido
> centroids = getOperation(llotes, operation="centroid")
```

## 2 Intersection area

The first example of GPM starts with a strategy that uses the intersection area to create relations between cells and polygons. A cell is connected to the polygon

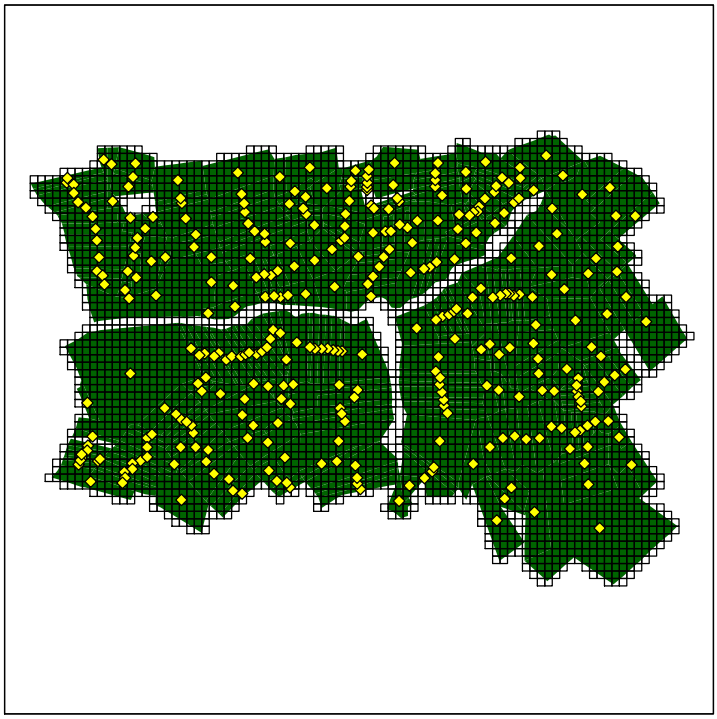

Figure 2: Data generated form the layer of polygons.

that has the largest intersection area. `connectToBiggerIntersectionArea()` can be used to compute this operation. It gets a set of cells and a layer of polygons as arguments and returns a table with two columns: the object id of the polygon with largest intersection area and the intersection area itself.

```
> mytable = connectToBiggerIntersectionArea(cells, llotes)
> get_property_from_cell = function(id)
+ {
+     list(ids=mytable[id,"father"], area = mytable[id,"area"])
+ }
```

After creating the function that generates the neighbors of a given object, the GPM can be created straightforwardly by calling `createGPM()`. It takes two arguments: the database layer with the dataset and the function that creates the neighborhood. The GPM stores the neighborhoods of all objects, with other atributes according to the adopted strategy, such as the 'area,' in this case.

```
> gpmcellsprop = createGPM(lcells, get_property_from_cell)
> as.data.frame(gpmcellsprop[1:2])
```

```
  C68L00.ids C68L00.area C69L00.ids C69L00.area
1        961    2744.903        961    9396.597
```

This GPM, shown in Figure 3, can be saved as a GAL[1] file by using `saveGAL`.

```
> saveGAL(gpmcellsprop, "cell-neighborhood.gal", "cells")
```

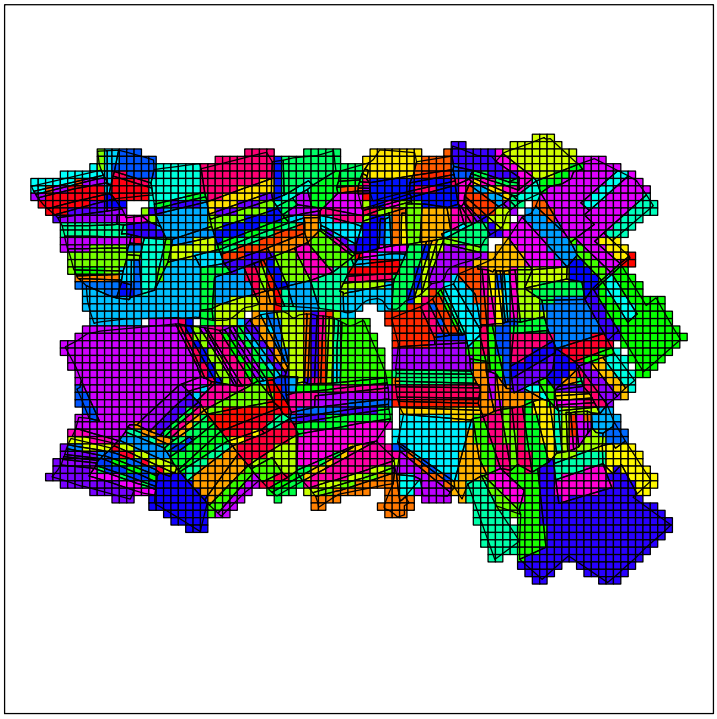

Figure 3: Relations from cells to the polygon with larger intersection area.

# 3 Euclidean distance

The second strategy uses the centroids to create relations between points that are closer than 1000m. To accomplish that, we use `getNeighborsEuclidean-MaxDistanceFunction()` to generate a function that returns the neighbors within a given distance. Finally we use `createGPM()` from the layer of polygons to generate the results shown in Figure 4.

[1]For more information on GAL format, see `http://geodacenter.asu.edu/software/documentation`.

```
> get_neighbors_lotes   = getNeighborsEuclideanMaxDistanceFunction(centroids, 1000)
> gpmdistance  = createGPM(llotes, get_neighbors_lotes)
```

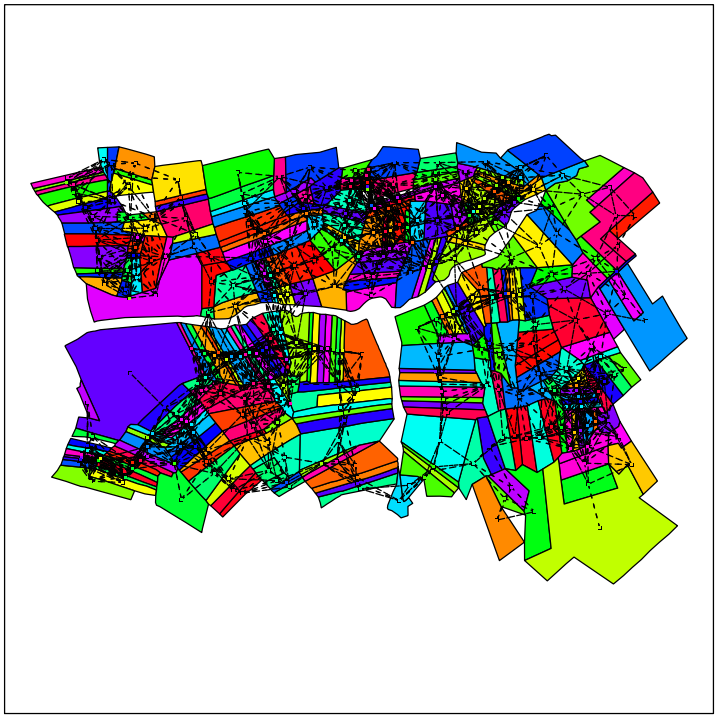

Figure 4: Neighborhoods of centroids within 1000m of radius.

# 4 Networks

The last strategy presented in this vignette computes neighborhoods based on the distance through a given network represented by a set of lines. The original data has to be very well represented, with the starting and ending points of two lines being connected to one another when they share the same position in space. In this type of network, it is possible to enter and leave the roads in any position. `createOpenNetwork()` is then used to generate the network. It takes as arguments the destination (reference) points, the lines that will be used to represent the network, and a function that computes the distance on the network given the length of the lines. The code below creates a network that reduces the distance within the network by one third of the Euclidean distance.

```
> network = createOpenNetwork(comunidades, rodovias, function(d) d/3)
> get_neighbors_net  = getNeighborsOpenNetworkFunction(centroids, network)
```

```
> gpmnetwork   = createGPM(llotes, get_neighbors_net)
```

Figure 5 shows the polygons drawn with the color of the closest point through the network. There is a current known limitation in the current version of `createOpenNetwork()`, that does not work properly when the entry point on the network for a given point is the start or end of a line segment.

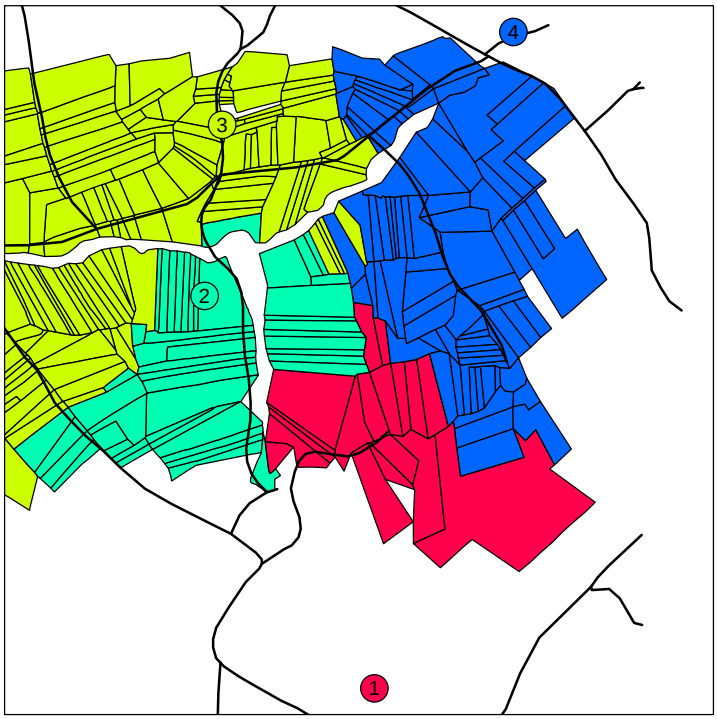


Figure 5: A non-squared cellular space covering the box of the polygonal set.