Generalized Proximity Matrixes with aRT

Pedro Ribeiro de Andrade, Raian Vargas Maretto

March 21, 2012

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.

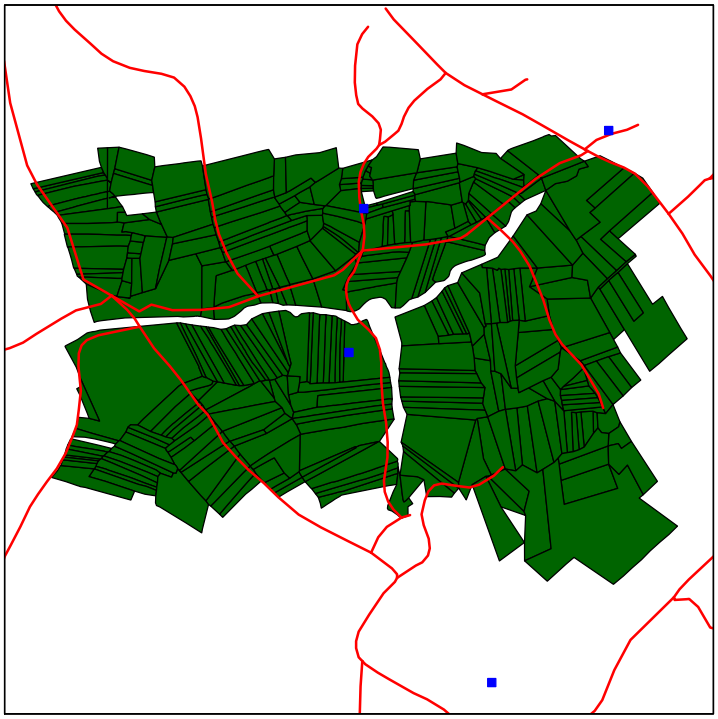

Figure 1: Database that will be used to compute spatial relations.

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")
```

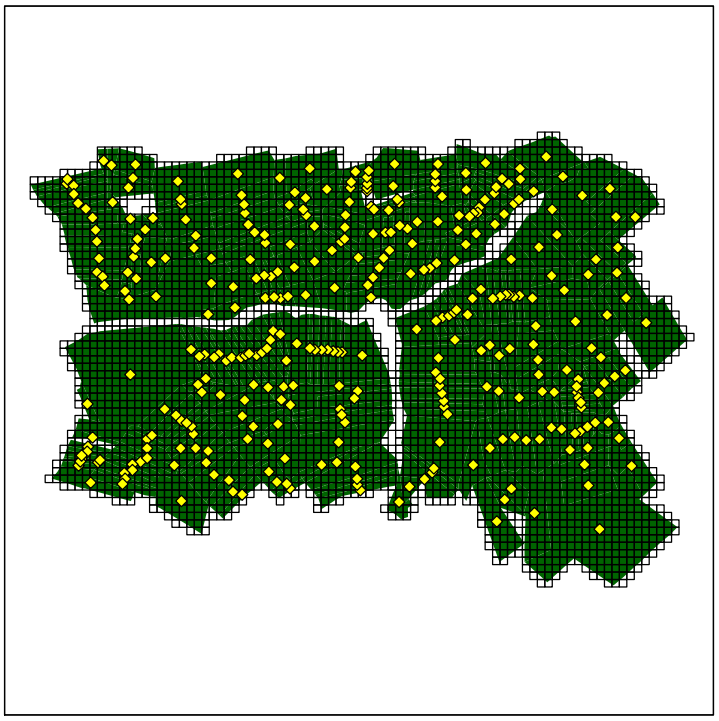

Figure 2: Data generated form the layer of polygons.

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 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])

  C00L06.ids C00L06.area C00L07.ids C00L07.area
1       1727    1018.821       1727    11842.78
```

This GPM, shown in Figure 3, can be saved as a .gpm, GAL or GWT file by using `saveGPM()`, presented in more details in section 8. The code below saves it in a .gpm file:

```
> saveGPM(gpmcellsprop, "cell-neighborhood.gpm", "cells", "lotes")
```

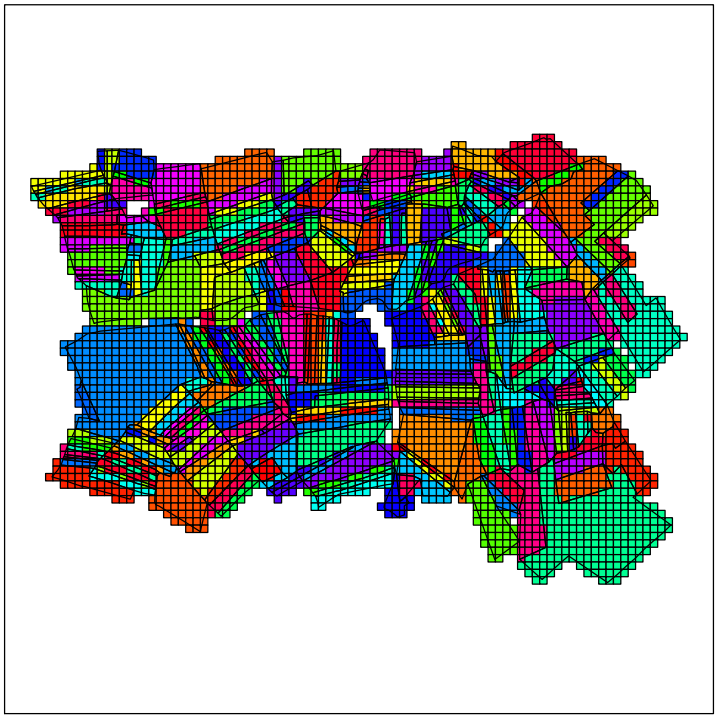

Figure 3: Relations from cells to the polygon with larger intersection area.

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.

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

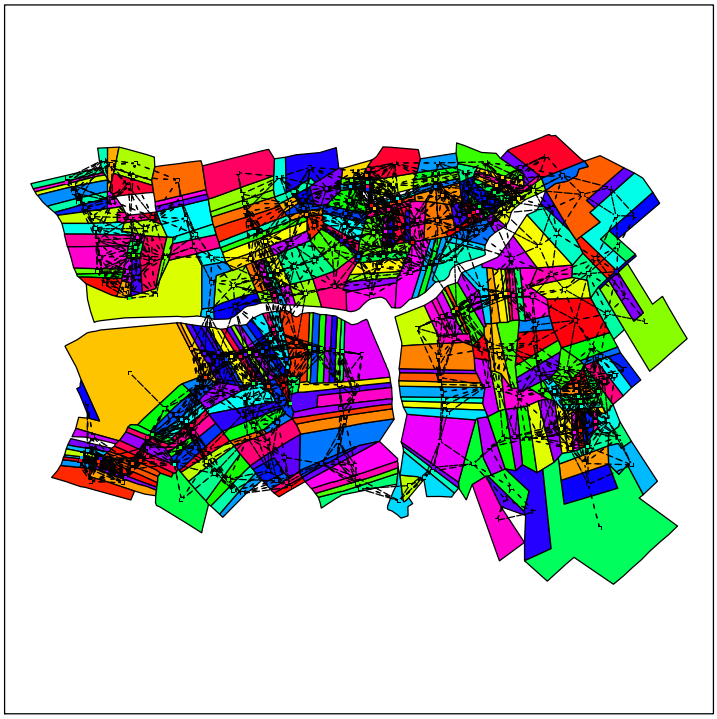

Figure 4: Neighborhoods of centroids within 1000m of radius.

4 Intersection with lines

The strategy presented in this section computes neighborhoods based on the intersection between lines and cells. Each cell is connected with the line segments that intersects it. The function `getNeighborsIntersectionLines()` can be used to generate the function that returns the neighbor lines that intersects a

given cell. It gets a layer of cells and a layer of lines as arguments and returns a function used to effectively create the GPM. Finally, the GPM is effectively created by the `createGPM()`, which receives as arguments the layer of cells passed as argument in the `getNeighborsIntersectionLines()` fuction, and the result returned by it. The code below creates a neighborhood between the layer "cells" and the layer "rodovias".

```
> get_neighbor_lines = getNeighborsIntersectionLines(lcells, lrodovias)
> gpm_intersection_lines = createGPM(lcells, get_neighbor_lines)
```

This GPM can be saved in a file (.gpm, .GAL or .GWT) through the `saveGPM()` method, presented in section 8.

5 Connecting cells with contained points

In this section, we present a function that computes neighborhoods between a layer of cells and a layer of points based on the "contains" spatial relation. Thus, a cell is connected to the points located inside its area. The function `getNeighborsContainedPoints` generates the function that returns the neighbor points located inside the area of a given cell. It gets a layer of cells and a layer of points as arguments and returns a function used to effectively compute the GPM. Finally, the GPM is effectively created by the `createGPM()` method, which receives as arguments the layer of cells passed as argument in the `getNeighborsContainedPoints()` function, and the result returned by it. The code below creates a neighborhood between the layer "cells" and the layer "comunidades".

```
> get_neighbor_points = getNeighborsContainedPoints(lcells, lcomunidades)
> gpm_contained_points = createGPM(lcells, get_neighbor_points)
```

This GPM can be saved in a file (.gpm, .GAL or .GWT) through the `saveGPM()` method, presented in section 8.

6 Connecting lines with intersection polygons

In this section, we present an strategy to compute neighborhoods between a layer of lines and a layer of polygons, in which each line has as neighbors the polygons intersected by it. The function `connectLineToIntersectionPolygons()` generates the function that returns the neighbor polygons intersected by a given line. It gets a layer of lines and a layer of polygons as arguments and returns a function used to effectively compute the GPM. Finally, the GPM is effectively created by the `createGPM()` method, which receives as arguments the layer of lines passed as argument in the `connectLineToIntersectionPolygons()` function, and the result returned by it. The code below creates a neighborhood between the layer "rodovias" and the layer "lotes".

```
> get_neighbor_lines_pols = connectLineToIntersectionPolygons(lrodovias, llotes)
> gpmLinePols = createGPM(lrodovias, get_neighbor_lines_pols)
```

7 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 and their id. The code below creates a network that reduces the distance within the network by one fifth of the Euclidean distance for paved roads and by half on the others. The attribute *pavimentada* of the table connected to the layer of lines indicates whether the road is paved or not.

```
> data = getData(openTable(lrodovias))
> network = createOpenNetwork(comunidades, rodovias, function(d, id) {
+   pos = which(data[,"OBJEID_57"] == id)
+   if(length(pos) == 1 && data[pos, "CD_PAVIMEN"] == "pavimentada")
+         return(d/5)
+   else
+         return(d/2)
+ })
> 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.

8 Saving the GPM in files

Once we have created the GPM through one of the strategies presented above, we can save it in a file, which can be a GPM file (".gpm"), a GAL file (".gal" or ".GAL") or a GWT file (".gal" or ".GWT"), through the function `saveGPM()`. The arguments of this function are:

- **gpm**: the gpm object to be saved;
- **filename**: a string containing the name of the file to be created. The extension is automatically recognized. If you wants to save the file in the current directory, the string would contain just the filename, followed by

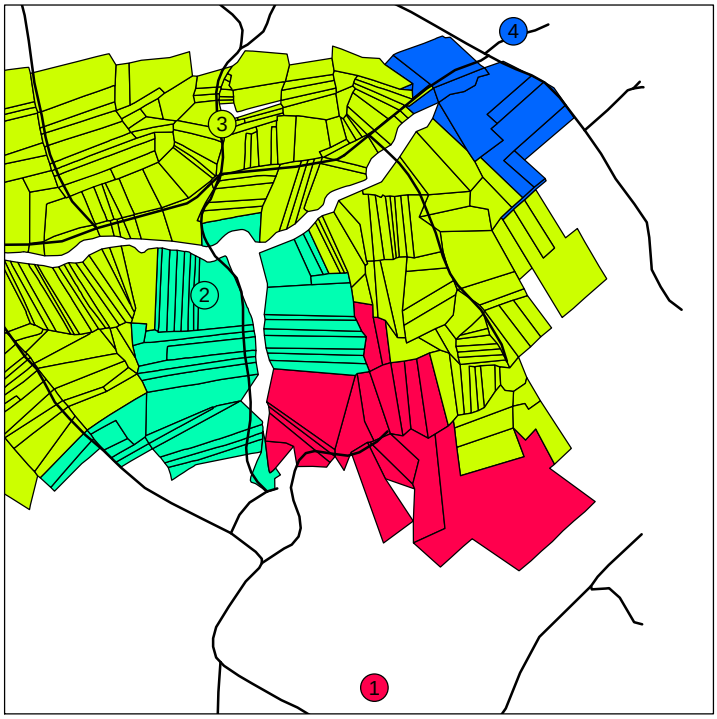

Figure 5: A non-squared cellular space covering the box of the polygonal set.

the extension. If you wants to specify the location where the file will be saved, the string would contain the complete path to the file, followed by the filename and extension;

- **layer1**: a string containing the name of the layer of objects for which the GPM was created;
- **leyer2**: a string containing the name of the layer where the objects of *layer1* has neighbors. This parameter is optional. If it is NULL, it is supposed to be the *layer1*, i. e., the neighborhood is not between two layers, but between objects of the same layer (*layer1*);
- **key**: a string containing the name of the object attribute used as identifier in the file to be saved. This argument is used only for GAL and GWT extrtensions, and its default value is "object_id_", which is the TerraLib unique identifier for the objects of a layer.
- **attrib**: a string containing the name of the gpm attribute used as weight. This argument is used only for GWT extension. It defines what attribute of the gpm will be used as weight when saving the GWT file.

The code below saves the GPM *gpmnetwork*, created in section 7 in the file "gpmnetwork.gpm", presented in Figure 6.

```
> saveGPM(gpmnetwork, "gpmnetwork.gpm", "lotes", "comunidades")
```

```
1       lotes   comunidades     distance
1000    1
22      3091.965
1033    1
16      2752.876
1034    1
19      3908.521
1045    1
19      4444.941
1086    1
22      4011.057
1088    1
22      3561.864
1089    1
22      3741.373
1094    1
22      3621.714
1097    1
22      1371.966
```

Figure 6: Stretch of the file *gpmnetwork.gpm*.

The structure of the GPM file is presented in Table 1. The first line is the header, and the GPM starts in the second line. In the header, we have the following fields:

- **Num_attributes** is the number of attributes of the relations. In the GPM, each relation can have several attributes, which represent its properties.
- **Layer_1** is the name of the layer for which the GPM was created.
- **Layer_2** is the name of the layer where the objects of *Layer_1* has neighbors. If the GPM was created between cells of the same layer, then the name of *Layer_1* is repeated in this field, i. e., *Layer_2* = *Layer_1*.
- **Attribute_1, ..., Attribute_N** are the names of the GPM attributes.

From the second line until the end of the file, the GPM is represented. The neighborhood of each object is represented in two lines. The first contains:

- **ID_Object_N** is the unique identifier of the N-th object;
- **Num_Neighbors** is the number of neighbors of the N-th object;

and the second contais the neighborhood of the object which ID is in the previous line, represented by the fields below, following the structure presented in Table 1:

- **ID_Neighbor_M** is the M-th neighbor of the N-th object;
- **Attrib_K_Neigh_M** is the value of the k-th attribute of the M-th neighbor;

The structure of the GAL file is presented in Table 2. It does not store informations about the attributes of the GPM, but only if two objects are neighbors or not. Furthermore, it does not support neighborhoods between objects of different layers. The first line of the file, as well as in the GPM file, is the header, and the GPM starts in the second line. In the header, we have the following fields:

- **0** is not describe by the creators of the format (Detailed description can be found in the GeoDa User's Guide). Thus, we save it as 0, and it is not used;
- **Num_elements** is the number of objects of the *Layer*;
- **Layer** is the name of the layer for which the GPM was created;
- **Key_Variable** is the name of the object attribute used as identifier of the objects. The default value is "object_id_", which is the unique identifier of the objects in TerraLib.

From the second line until the end of file, the relations are represented. The neighborhood of each object is represented in two lines. The first contains:

- **ID_Object_N** is the unique identifier of the N-th object;
- **Num_Neighbors** is the number of neighbors of the N-th object;

and the second line contains the unique identifier of the neighbors (ID_Neighbor_M) of the N-th object. The code below saves the GPM *gpmdistance*, created in section 3, in the file "gpmdistance.GAL", presented in Figure 7.

```
> saveGPM(gpmdistance, "gpmdistance.GAL", "lotes")
```

The structure of the GWT file is presented in Table 3. It can store one of the attributes of the GPM, which name is passed as parameter to the `saveGPM()` function. This format, even as the GAL file, does not support neighborhoods objects of different layers. The header of the GWT format is the same of the GAL one. However, from the second line until the end of file, it represent one connection by line, where it has the following fields:

- **ID_Object_N** is the unique identifier of the object N-th object;

```
0 343 lotes object_id_
1000 6
1088 1089 2584 593 923 989
1033 9
2022 2572 2573 2574 2927 2937 2945 2946 2947
1034 17
1045 1716 1924 1934 1945 1956 1967 2467 2468 2469 626 637 658 669 680 753 756
1045 9
1034 1716 1727 1924 1934 1945 658 669 680
1086 8
1088 1089 1094 1097 743 814 823 872
1088 13
1000 1086 1089 2584 593 814 835 846 857 923 934 945 989
1089 11
1000 1086 1088 2584 593 814 835 923 934 945 989
1094 2
1086 743
1097 5
1086 1099 743 871 872
```

Figure 7: Stretch of the file *gpmdistance.GAL*.

- **ID_Neighbor_M** is the M-th neighbor of the N-th object;
- **Weight_Neighbor_M** is the weight (attribute value) of the relation between the N-th object and the M-th neighbor.

The code below save the gpm *gpmdistance*, created in section 3 in the file "gpmnetwork.GWT", presented in Figure 8.

```
> saveGPM(gpmdistance, "gpmdistance.GWT", "lotes", attrib="distance")
```

```
0 343 lotes object_id_
1000 1088 556.1925
1000 1089 813.4312
1000 2584 702.8632
1000 593 242.8337
1000 923 963.1457
1000 989 493.0075
1033 2022 418.2776
1033 2572 935.693
1033 2573 663.5345
1033 2574 553.4554
1033 2927 932.6884
1033 2937 823.8539
1033 2945 593.9373
1033 2946 409.1091
1033 2947 569.6868
1034 1045 745.5015
1034 1716 998.0759
1034 1924 856.1945
```

Figure 8: Stretch of the file *gpmdistance.GWT*.

More informations about the GAL and GWT formats can be found in the GeoDa User's Guide and in the SpaceStat documentation.

Num_attributes	Layer_1	Layer_2	Attribute_1	Attribute_2	...	Attribute_N
ID_Object_1	Num_Neighbors					
ID_Neighbor_1	Attrib_1_Neigh_1	Attrib_2_Neigh_1	...	Attrib_N_Neigh_1	ID_Neighbor_2	...
ID_Object_2	...					
...						

Table 1: Structure of the GPM format

0	Num_elements	Layer	Key_Variable
ID_Object_1	Num_Neighbors		
ID_Neighbor_1	ID_Neighbor_2	...	ID_Neighbor_N
...			

Table 2: Structure of the GAL format

0	Num_elements	Layer	Key_Variable
ID_Object_1	ID_Neighbor_1	Weight_Neighbor_1	
ID_Object_1	ID_Neighbor_2	Weight_Neighbor_2	
...			
ID_Object_1	ID_Neighbor_N	Weight_Neighbor_N	
ID_Object_2	ID_Neighbor_1	Weight_Neighbor_1	
...			

Table 3: Structure of the GWT format