University of Chicago
Department of Sociology
Autumn 2016

SOCI 20253/GEOG 20500, SOCI 30253
Introduction to Spatial Data Science

by
Luc Anselin (anselin@uchicago.edu)

Lab 3 – Spatial Weights (Oct 17, 2016)

[Disclaimer: these notes are lab notes and not a polished manual or textbook]

In this lab, we will explore the spatial weights functionality in GeoDa and also briefly review some options that are available in the R spdep package. We will use two data sets that are available on the course web site, one for 3085 continental U.S. counties (`natregimes`) and one for a house sales in the last quarter of 2014 in the core of Cleveland, Ohio (`clev_sls_154_core`). The former is unprojected (lat-lon), whereas the latter is projected in North Ohio State Plane coordinates (in feet). As we have seen, this matters when computing distances. For the U.S. counties, we will create contiguity-based weights (rook and queen), as well as higher order contiguity. We will examine their properties using the connectivity histogram and explore the connectivity map. We will use the house sales (points) to construct distance-based weights (distance band and k-nearest neighbors), and again investigate their properties. We will also illustrate how in GeoDa one can compute distance-based weights for polygons, and contiguity-based weights for points. Finally, we will take a look at the Project File, which is a way to remember project settings (such as spatial weights) between sessions.

Using spdep, we will highlight the difference between neighbor lists and weights objects in R, assess and visualize their characteristics, and explore different weight standardizations (in GeoDa, spatial weights are always used in row-standardized form). We will also illustrate a procedure to construct inverse distance weights.

By now, you should be pretty familiar with the basic operations of GeoDa and R. Details on weights operations in GeoDa are given in Chapters 3 and 4 of Anselin and Rey (2014), although the interface is for an earlier version. As of GeoDa 1.8, all weights-related functionality is grouped under the Weights Manager (instead of the three separate icons used before).

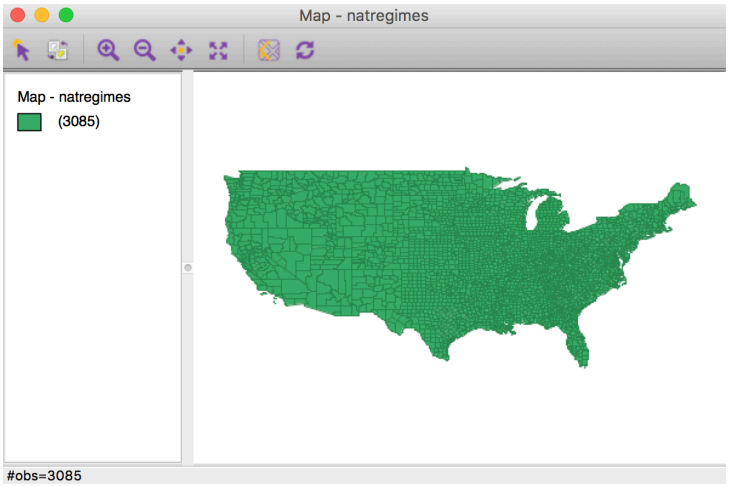
Further materials on the weights operations in spdep can be found in the package manual and in a recent "vignette" by Roger Bivand on "Creating Neighbours" (both available on CRAN as well as on the course web site), as well as in Chapter 9 of Bivand et al (2013).

After completing the lab, you should know how to

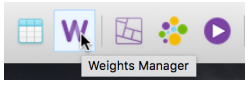
- Construct contiguity-based spatial weights in GeoDa
- Create distance-based spatial weights in GeoDa
- Compute higher order contiguity weights in GeoDa
- Store the weights information in a GeoDa Project File
- Assess the properties of spatial weights in GeoDa
- Read spatial weights into spdep
- Manipulate neighbor lists and weights objects in spdep
- Assess and visualize the characteristics of spatial weights in spdep
- Construct inverse distance weights in spdep

Contiguity-based weights in GeoDa

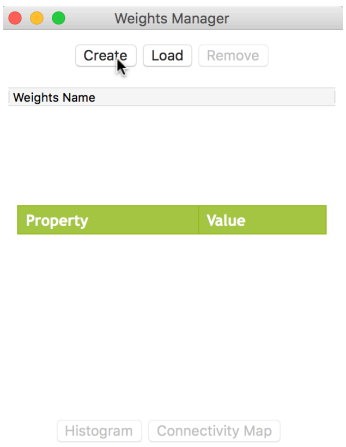
We start by loading the `natregimes.shp` file into GeoDa. This brings up the base map showing 3085 continental counties. The `natregimes` data set is one of the sample data sets used in Anselin and Rey (2014), which contains more specifics on the variables included, sources, etc.



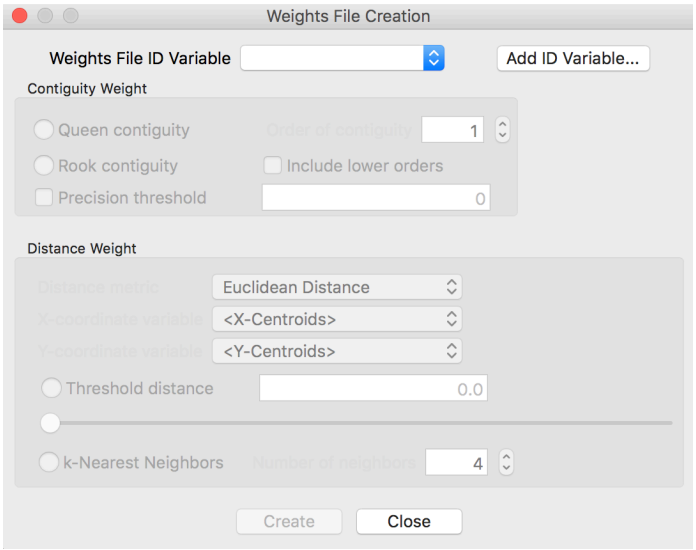
We invoke the weights creation through the `Weights Manager` icon in the toolbar, or by selecting `Tools > Weights Manager` in the menu.



Clicking on the W icon brings up the `Weights Manager` dialog. At this point it is totally empty. Select the `Create` button to start constructing the weights.

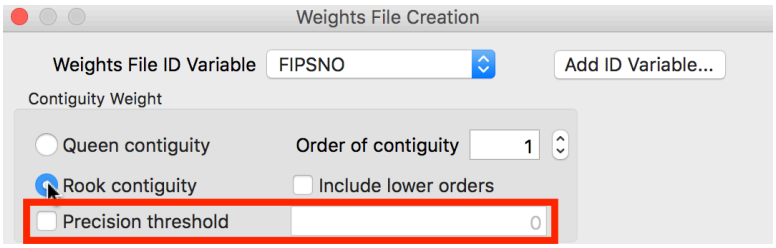


The actual construction of the weights is implemented through the `Weights File Creation` interface. This organizes all the different options in one place. The first item to specify is the `ID Variable`. This is a critical element to make sure that the weights are connected to the correct observations in the data table. In other words, the `ID Variable` is a so-called key that links the data to the weights. In GeoDa, it is best to have the `ID Variable` be integer. In practice, this is often not the case. One way to deal with this is to use the `Variable Edit` functionality in the `Table` to turn a string into an integer, as we have seen earlier. However, sometimes there is no easy way to identify an ID variable. In that case, the `Add ID Variable` button provides the solution: the added ID variable is simply an integer sequence number that is added to the data table (you must `Save` the project to make the addition permanent).



For the `natregimes` data set, we use `FIPSNO` as the ID variable. This is the county FIPS code turned into an integer value. We can then proceed to the `Contiguity Weight` panel in the interface (the top panel). Check the `Rook contiguity` radio

button and click on `Create` to construct the weights file. A File dialog appears in which you specify a file name for the weights (the file extension GAL is added automatically). Next, the weights are computed and written to the file. At the end of this operation, a success message will appear (or an Error message if something went wrong).



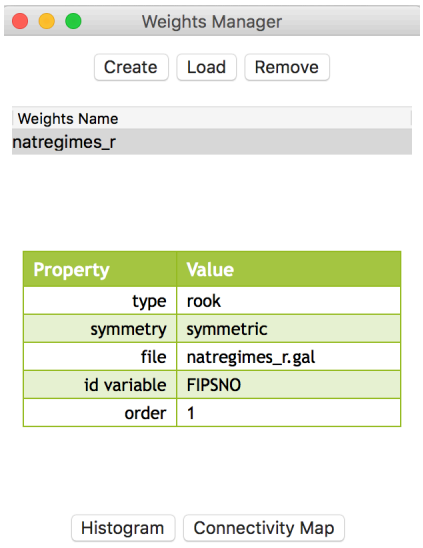
A useful option in the `Weights File Creation` dialog is the specification of a `Precision threshold` (highlighted in the screen shot). In most cases, this is not needed, but in some instances the precision of the underlying shape file in insufficient to allow for an exact match of coordinates (to determine which polygons are neighbors). When this happens, GeoDa suggests a default error band to allow for a fuzzy comparison.

The GAL weights file just created is a simple text file that contains, for each observation, the number of neighbors and their identifiers. The format was suggested in the 1980s by the Geometric Algorithms Lab at Nottingham University and achieved widespread use after its inclusion in SpaceStat (Anselin 1993) and subsequent adoption by spdep. The one innovation SpaceStat added was the inclusion of a header line, with some metadata for the weights, such as the number of observations, the name of the shape file from which the weights were derived, and the name of the ID variable. For each observation, the number of neighbors is listed after its ID (e.g., for county `27077`, there are 3 neighbors), followed by the IDs of the neighbors (`27135 27071 27007`).

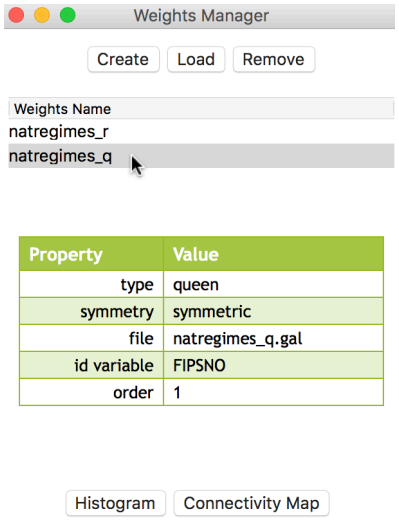
```
0 3085 natregimes FIPSNO
27077 3
27135 27071 27007
53019 3
53065 53047 53043
53065 4
53019 53051 53063 53043
53047 7
53019 53073 53057 53007 53017 53025 53043
53051 4
53065 16021 16017 53063
```

Since the GAL file is a simple text file, it can easily be edited (e.g., to add or remove neighbors), although this is not recommended: it is easy to break the inherent symmetry of the contiguity weights.

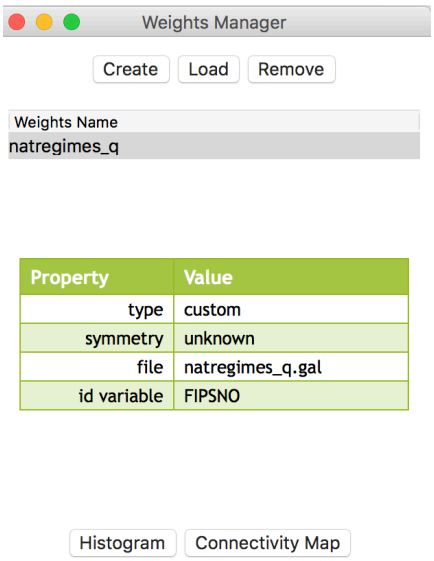
After the rook weights are created, they are listed in the weights manager. In the box below the `Weights Name`, some summary properties are given, including the type (`rook`), whether the weights are inherently symmetric or not (`symmetric`), the file name (`natregimes_r.gal`), the id variable (`FIPSNO`) and the order of contiguity (`1`).



We proceed in the same fashion to construct queen contiguity weights (check the `Queen contiguity` radio button). After the weights are saved to the weights file, they are also listed in the `Weights Manager`. In this dialog, the highlighted weights are the active ones. This determines which weights are used in any analysis, but also determines the properties that are listed in the dialog and what is generated by the connectivity `Histogram` and the `Connectivity Map`. For example, the summary properties for the queen weights are shown below, since that is the highlighted weights name.



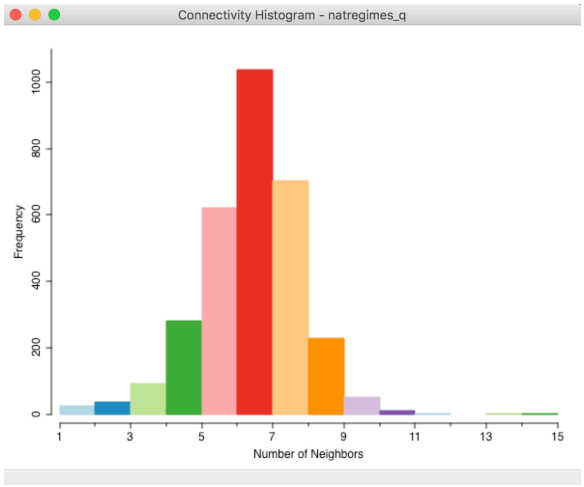
The `Weights Manager` can also be used to `Load` weights files that are already available on disk. Click on the `Load` button (center top) and specify the name of the weights file. However, unlike the weights created on the fly in the current session, only limited items will be contained in the properties list, since there are no metadata for the weights files. For example, as such, GeoDa has no way of knowing whether the loaded file represents queen or rook contiguity (given as `custom` in the properties list), whether it is symmetric (`unknown`), or the order of contiguity.



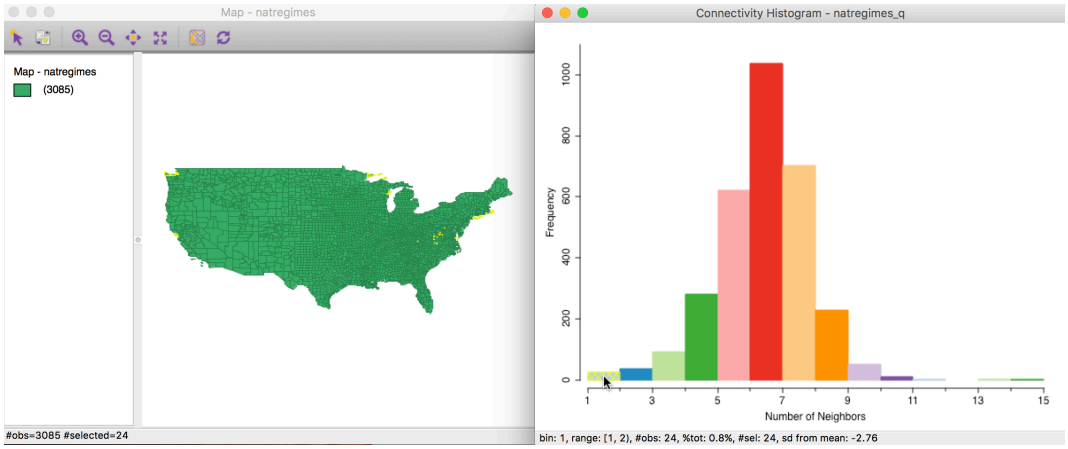
However, this is different when a GeoDa `Project File` is used (with file extension `GDA`). Such a file does contain some metadata about the weights (the same information as contained in the properties list). This is elaborated upon below.

Properties of weights

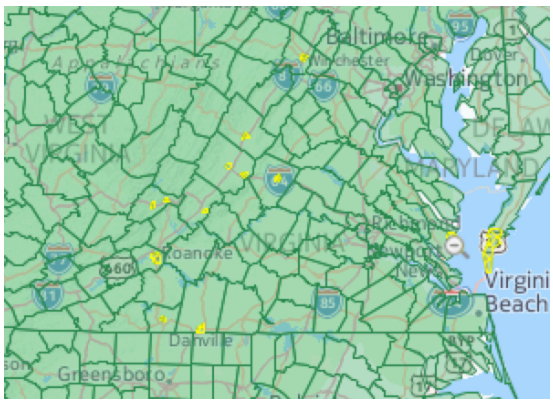
Some useful properties of the currently selected weights are provided in the `Weights Manager` by means of the `Histogram` and the `Connectivity Map`. The `Histogram` button produces a connectivity histogram that shows the cardinality of observations for each number of neighbors (i.e., how many observations have the given number of neighbors). The graph is a standard GeoDa histogram, with a number of options available, some of which are generic, and some specific to the connectivity histogram.



The histogram is connected to all the other views through linking and brushing. For example, by selecting the first bar, all counties with one neighbor are highlighted in the county map.

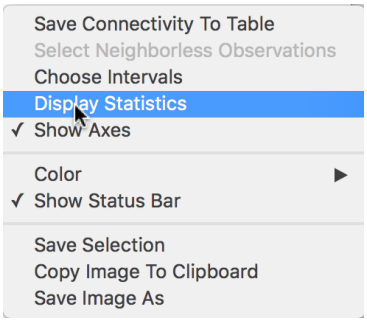


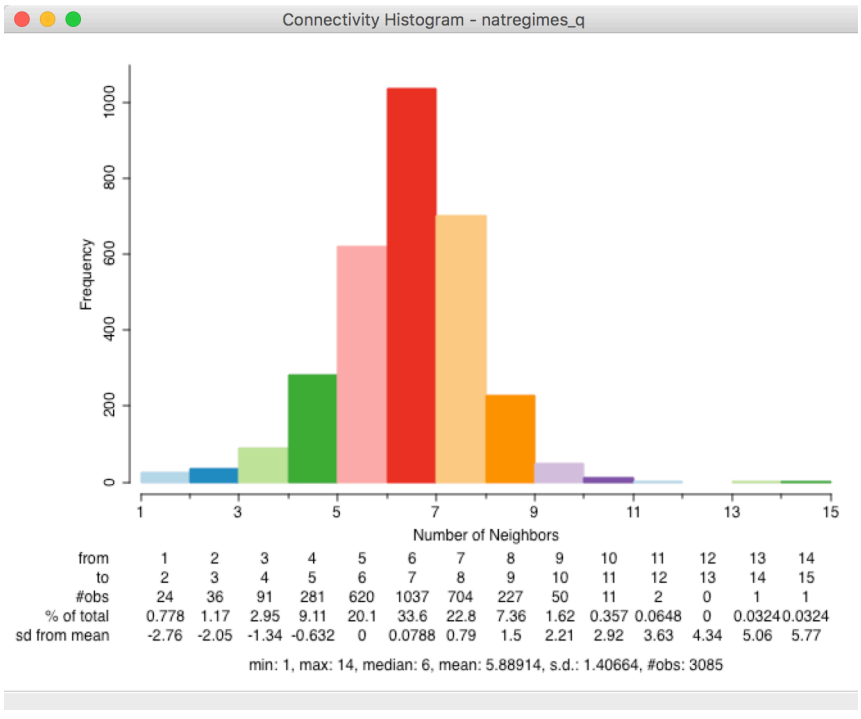
As an interesting aside, note how several of these counties are so-called city counties in the state of Virginia. They are surrounded by a “regular” county, which is their only neighbor.



It is good practice to check the connectivity histogram for any "strange" patterns, such as this one (we cover isolates or neighborless observations in the discussion of distance-based weights). Ideally, we like the distribution of the cardinalities to be nice and symmetric, and are on the lookout for bimodal distributions (some observations have few neighbors and some many) and other deviations from symmetry.

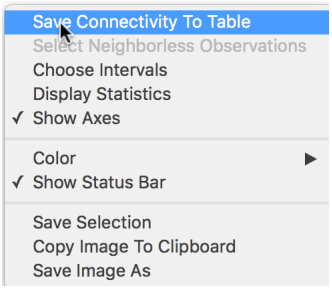
One of the options (right click in the view) of the histogram is to display the statistics (you may need to adjust the window size to see them properly). They appear below the histogram graph.





From the descriptive statistics listed at the bottom of the graph, we can see that the median number of neighbors is 6, the average is 5.89, and the maximum is 14. In addition, the number of observations in each interval is listed as well.

A second useful option to the connectivity histogram is to save the neighbor cardinality to the data table as an additional column/variable to be used in further analysis. The `Save Connectivity to Table` option is invoked in the usual way by right clicking on the view. This brings up a small dialog in which the name for the new variable can be specified (the default is a generic `NUM_NBRS`, which may not be the most insightful when different spatial weights are being compared). Upon clicking `OK`, an additional column is added to the data table with the number of neighbors for that observation (and that particular spatial weights specification).



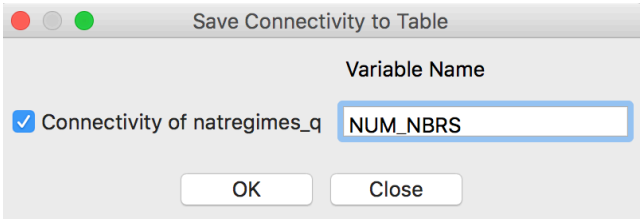
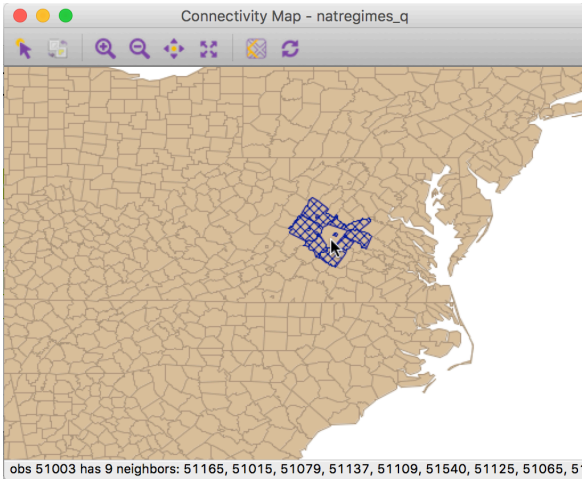


Table - natregimes

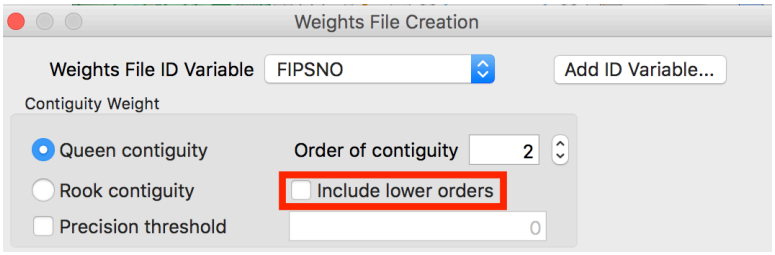
		FH90	West	NUM_NBRS
1	1	9.515860	0	3
2	6	11.397059	1	3
3	7	10.352015	1	4
4	6	12.840340	1	7
5	3	10.313002	1	4

The button on the right at the bottom of the `Weights Manager` interface brings up a `Connectivity Map`. This is a standard GeoDa map view (with all the usual map features, including zooming, panning, base maps, etc.), but with a special functionality that highlights the neighbors of any selected observation. For example, pointing at one of the Virginia counties shows its neighbors on the map (including the city county enclosed by it) and lists their ID values in the status bar. In this instance, the county with FIPS code 51003 (Albemarle, VA) has 9 neighbors using the queen contiguity criterion.

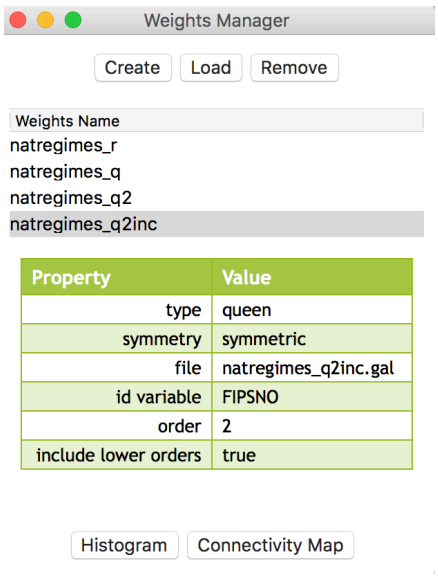


Higher order contiguity

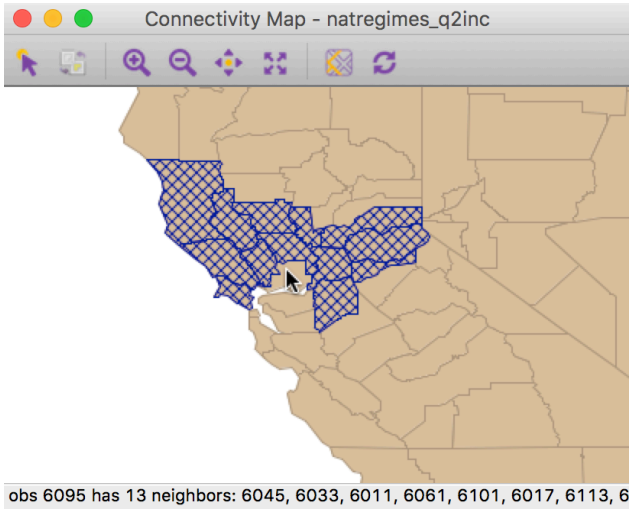
Higher order contiguity weights are constructed in the same manner as the first order weights we just covered, but by specifying a value larger than 1 (which is the default) in the `Order of contiguity` drop down list. The weights are saved to a file after selecting the `Create` button and specifying a file name.

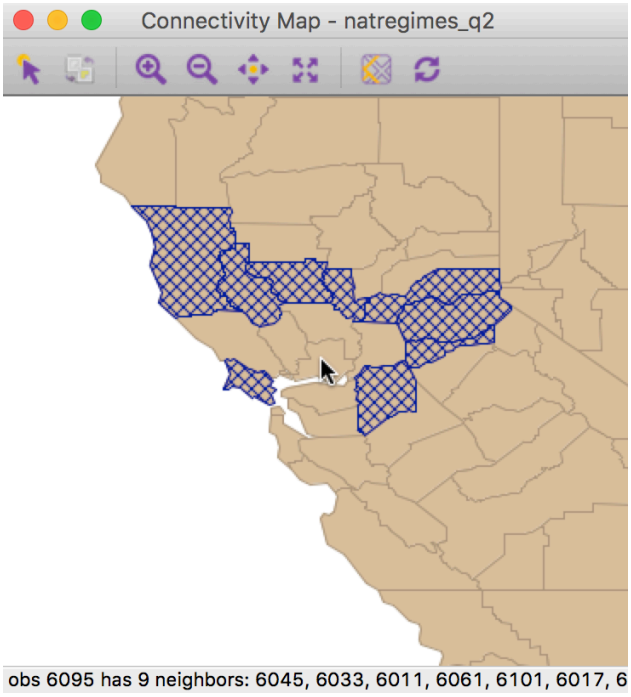


One important aspect of higher order contiguity weights is whether or not the lower order neighbors are included in the weights file. This is determined by a check box (highlighted in the figure above). Conceptually, there is quite a difference between the two concepts. The "pure" higher order contiguity does not include any lower order neighbors (for example, an encompassing notion of second order neighbors would include the first order neighbors as well, since there are two steps – back and forth – connecting each observation to its first order neighbor). This is the notion appropriate for use in a statistical analysis of spatial autocorrelation for different "lag" orders. However, in practice, it is often useful to use increasing orders of contiguity as similar to increasing distance bands, i.e., consisting of both first and second order neighbors. We can easily see the difference by means of the `Connectivity Map` (first, make sure you create the two weights, one without the "include lower orders" box checked, and one with). The `Weights Manager` should include both spatial weights in the weights list.



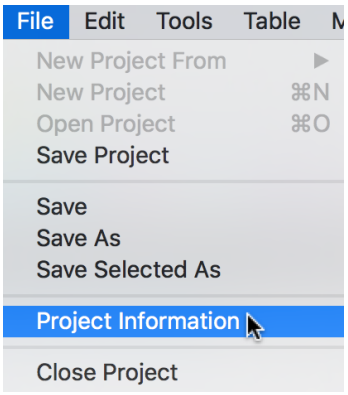
Consider the `Connectivity Map` for `natregimes_q2inc` (second order queen contiguity inclusive of first order neighbors) compared to the one for `natregimes_q2` (pure second order queen contiguity). We focus in on Solano county, CA in the Bay Area (FIPS code 6095): the second map does not include the first order neighbors in the inner ring. The contrast also shows up in the status bar: the inclusive higher order weights have 13 neighbors, the pure second order have 9.

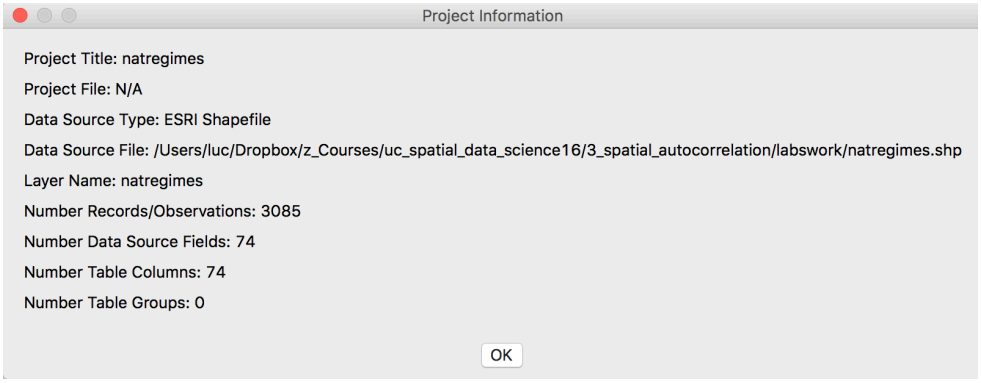




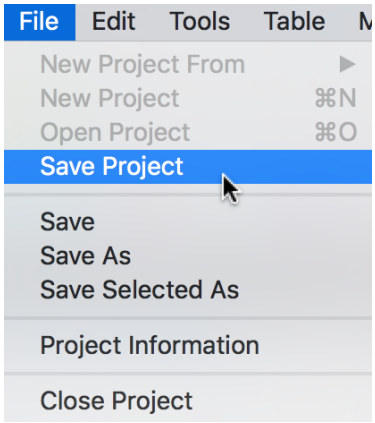
Project file

Upon closing the current project, the information on the characteristics of the spatial weights is lost. When you later reload the same polygon layer (e.g., `natregimes`), you have to reload each weights file using the `Load` button in the weights manager. As we have seen, the summary properties listed for those loaded weights are not very informative. A much superior alternative is to create and save a project file. This file contains information about the project, such as the variables contained in it (especially important when space-time variables are created), as well as the spatial weights. A quick view of the current project is given by selecting the `Project Information` item in the `File` menu. This brings up a summary of the project contents, in terms of data source, number of observations and number of variables. Note how the item `Project File` is listed as N/A (before a project file is created).





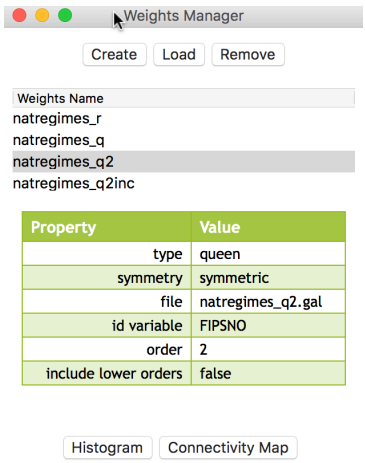
The project information is kept in memory and saved by means of the `Save Project` item in the `File` menu. This will prompt for a file name and save the project file with a file extension `GDA`.



The project file itself is an editable XML text file. It contains all the information on the source file, variable names, any transformations carried out and any space-time (grouped) variables created. It also keeps all the characteristics of the spatial weights. As we see below, the four weights created so far are included, with the properties as they were listed in the Weights Manager.

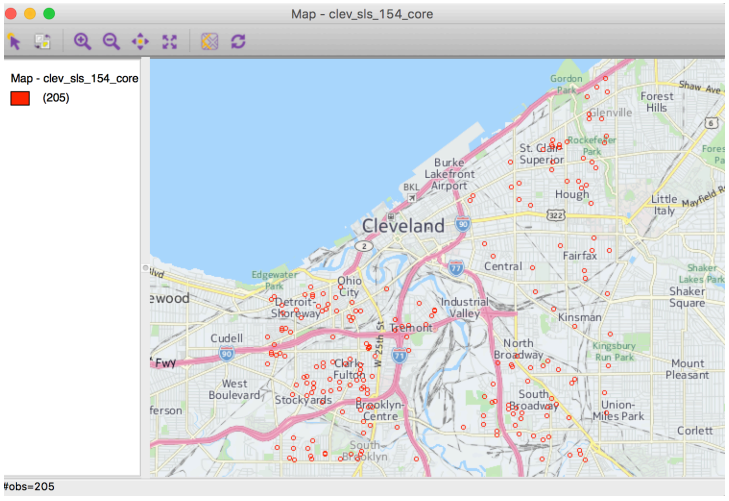
Instead of opening a shape file (or other geographical layer) to start a project, the Project File can (and should) be loaded instead. This will automatically load all the spatial weights contained in the project file and list their properties in the Weights Manager, as shown below. In addition, it will recreate the various transformations and space-time variables that were constructed earlier. You can save different project files for the same data set, each with their own set of weights, space-time variables, etc.

```
<weights_entries>
    <weights>
        <title>natregimes_r</title>
        <meta_info>
            <weights_type>rook</weights_type>
            <order>1</order>
            <inc_lower_orders>true</inc_lower_orders>
            <path>natregimes_r.gal</path>
            <id_variable>FIPSNO</id_variable>
            <symmetry>symmetric</symmetry>
        </meta_info>
    </weights>
    <weights>
        <title>natregimes_q</title>
        <meta_info>
            <weights_type>queen</weights_type>
            <order>1</order>
            <inc_lower_orders>true</inc_lower_orders>
            <path>natregimes_q.gal</path>
            <id_variable>FIPSNO</id_variable>
            <symmetry>symmetric</symmetry>
        </meta_info>
    </weights>
    <weights>
        <title>natregimes_q2</title>
        <default/>
        <meta_info>
            <weights_type>queen</weights_type>
            <order>2</order>
            <inc_lower_orders>false</inc_lower_orders>
            <path>natregimes_q2.gal</path>
            <id_variable>FIPSNO</id_variable>
            <symmetry>symmetric</symmetry>
        </meta_info>
    </weights>
    <weights>
        <title>natregimes_q2inc</title>
        <meta_info>
            <weights_type>queen</weights_type>
            <order>2</order>
            <inc_lower_orders>true</inc_lower_orders>
            <path>natregimes_q2inc.gal</path>
            <id_variable>FIPSNO</id_variable>
            <symmetry>symmetric</symmetry>
        </meta_info>
    </weights>
</weights_entries>
```



Distance-band weights

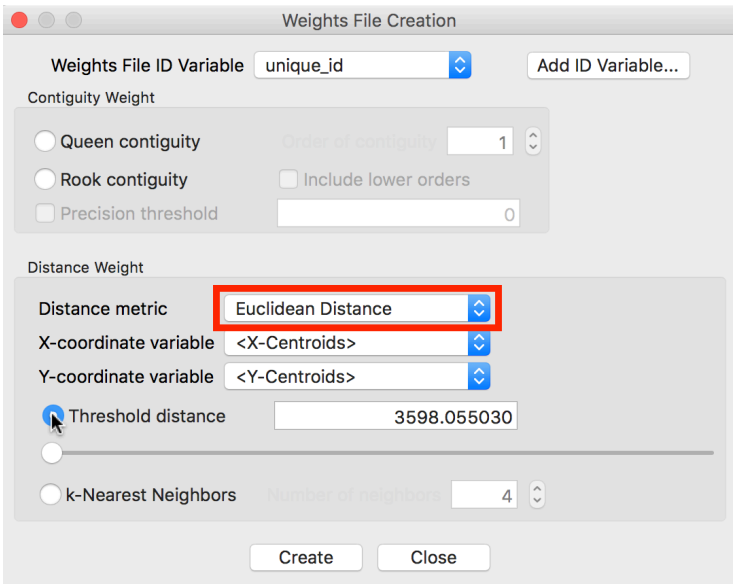
To illustrate the computation of distance-based weights, we use the file `clev_sls_154_core` that contains the location and sales price of home sales in a core area of Cleveland, OH for the fourth quarter of 2015. Make sure to first close the current project to remove the U.S. counties file, then load the Cleveland file. The base map appears in the map view.



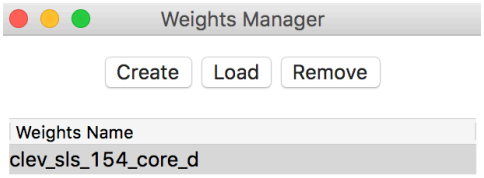
This map is produced by adding a base layer to the points (Nokia Day) and by turning the default green color for the points into red (right click on the legend and change Color for Category to red). The 205 points in question were selected as a rectangular subset of the full data set.

As before, we invoke the `Weights Manager` and click on the `Create` button to get the process started. In the `Weights File Creation` interface, after specifying the ID variable (`unique_id`), we focus on the lower panel (`Distance Weight`) and select the `Threshold distance` radio button. The default max-min distance (largest nearest neighbor distance) is given in units appropriate for the projection

used. Note the importance of the `Distance metric` (highlighted in the screen shot). Since our data is projected, it is appropriate to use Euclidean (straight line) distance. However, many data sets come in simple latitude-longitude, for which a great circle distance (or arc distance) must be used instead. We will revisit this shortly.



The critical distance for our point data is 3598 feet, or roughly 0.7 miles. This is the distance that ensures that each point (house sale) has at least one neighbor. After clicking on the Create button and specifying a file name (the `GWT` file extension is added automatically), the new weights are listed in the `Weights Manager`.



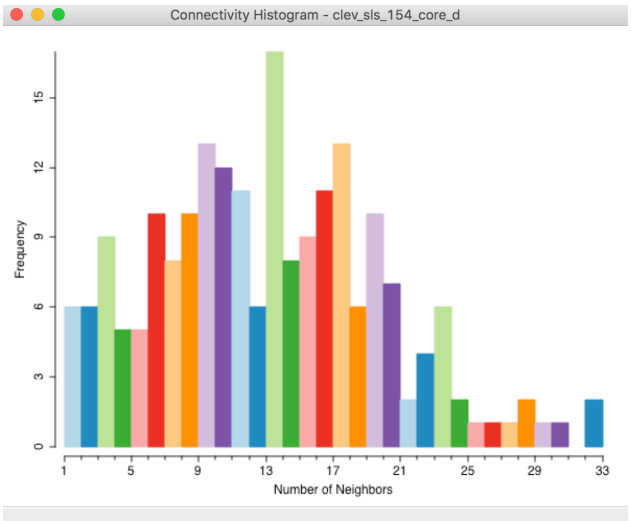
Property	Value
type	threshold
symmetry	symmetric
file	clev_sls_154_core_d.gwt
id variable	unique_id
distance metric	Euclidean
distance vars	centroids
distance unit	mile
threshold value	3598.06

Histogram Connectivity Map

Distance-based weights are saved in files with a GWT file extension. This format is slightly different from the GAL format and was first introduced in SpaceStat in 1995. It was later adopted by spdep and other software packages. The header line is the same as for GAL files, but each pair of neighbors is listed, with the ID of the observation, the ID of its neighbor and the distance that separates them. This distance is currently only included for informational purposes, since GeoDa does not use the distance measure itself in any operations.

```
0 205 clev_sls_154_core unique_id
1183 6842          3253.02459
1183 2024          1858.90398
1183 1624          3013.07086
1183 1198          385.161005
1183 1741          1160.31203
1183 2341          2525.50272
1198 6845          3174.32765
1198 2024          1758.55651
```

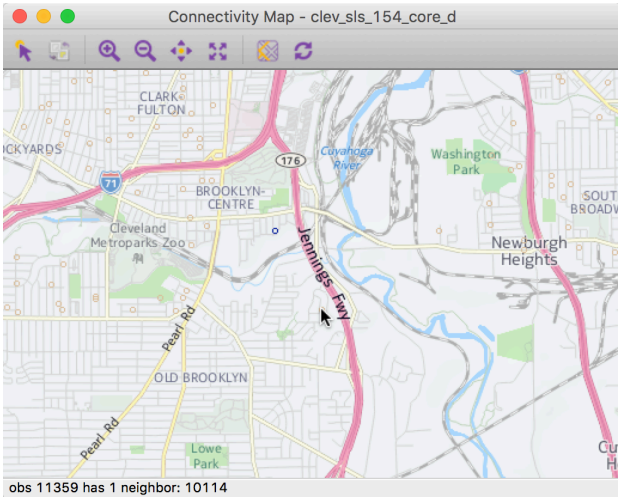
In the same way as we saw before, we can bring up the `Connectivity Histogram` (optionally with the descriptive statistics after a right click).



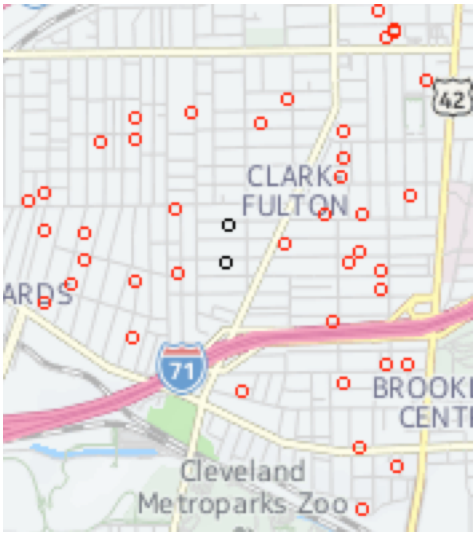
The shape of the connectivity histogram for distance-band weights is typically very different from that of contiguity-based weights. We see a much larger range in the number of neighbors, as well as extremes, with some observations having only one neighbor, and others 32. This is directly related to the spatial distribution of the points. Locations that are somewhat isolated will drive the determination of the largest nearest neighbor cut-off point (their nearest neighbor distance will be large), whereas dense clusters of locations will encompass many neighbors using this large cut-off distance. In the Cleveland example, we can examine the GWT file to find the observation pair separated by the critical distance cut-off (3598). This turns out to be the observation pair #11359 and #10014.

10114	17650	2696.98647
10114	17596	3340.21392
11359	10114	3598.05503
15739	21126	3571.74481
15739	22290	2993.21917

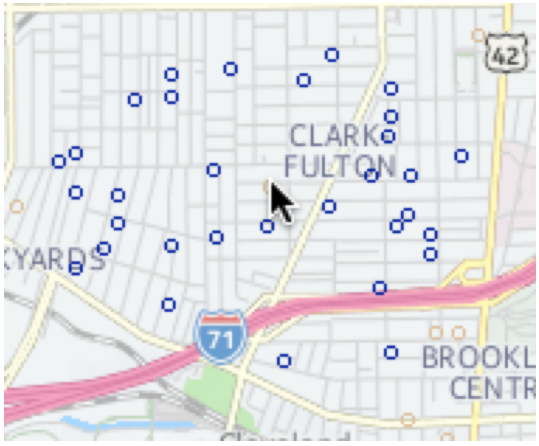
We can also see this in the Connectivity Map. When we hover over observation #11359, we find that it has one neighbor, #10114 (this takes some trial and error – first select this observation in the Table to locate it on the point map, and then find the same point in the Connectivity Map, using the Nokia Day base map to exploit the road network as a reference).



We see the effect of the large distance cut-off on more densely distributed point locations. For example, selecting the right-most bar in the Connectivity Histogram will highlight the two most connected observations in the map (to make things easier to see, the Highlight Color has been changed to black – right-click in the map and choose Color > Highlight Color).

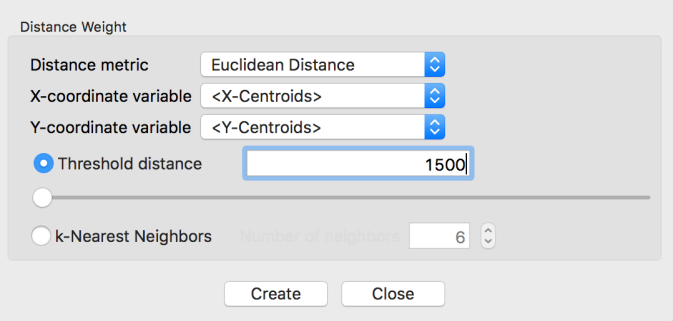


As expected, the two points in question are in the center of a dense cluster of sales transactions. When we inspect the `Connectivity Map`, we see that each of these points has 32 neighbors.



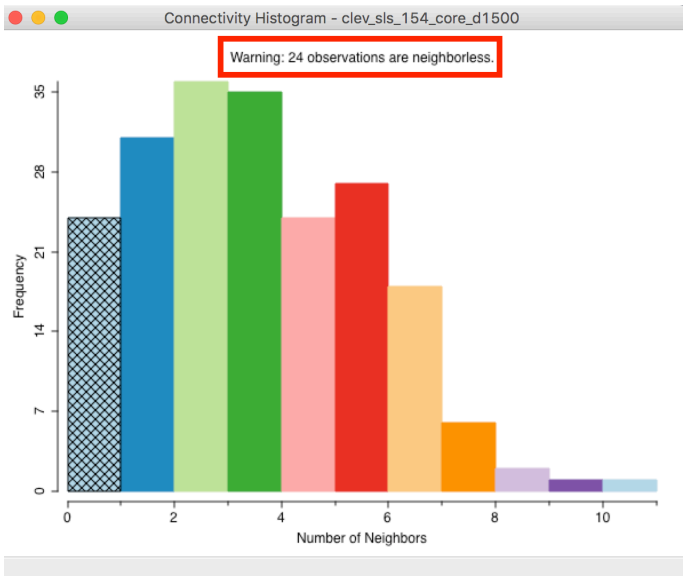
The unequal distribution of the neighbor cardinality in distance-band weights is often an undesirable feature. When the spatial distribution of the points is highly uneven, distance-band weights should be avoided, since they could provide misleading impressions of (local) spatial autocorrelation.

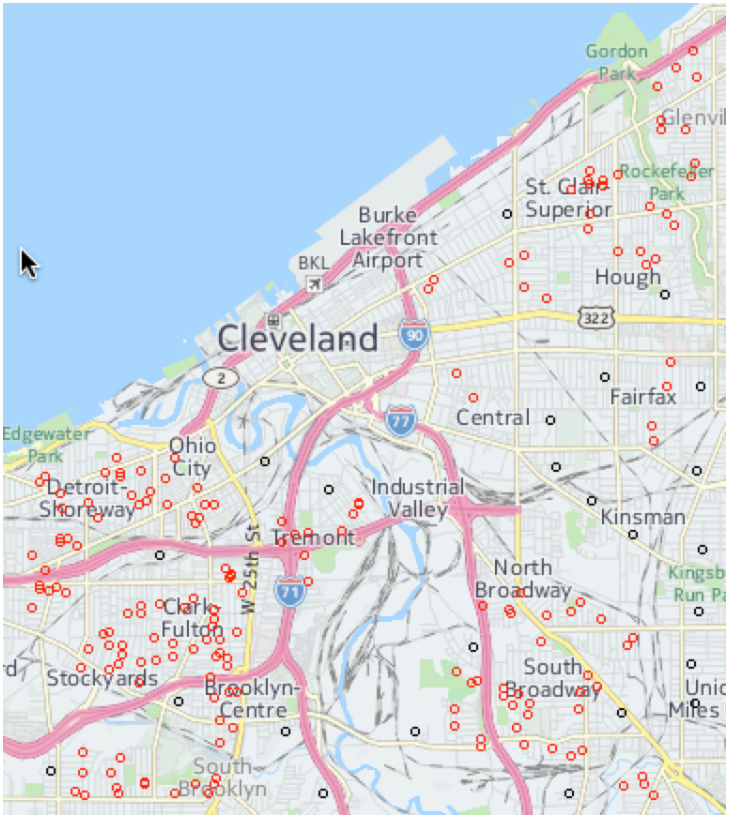
So far, we have used the default cut-off value for the distance band. However, the dialog is flexible enough that we can type in any value for the cut-off, or use the moveable button to drag to any value larger than the minimum. Sometimes, theoretical or policy considerations suggest a specific value for the cut-off that may be smaller than the max-min distance. For example, say we want to use 1500 ft as the distance band. After typing in that value in the dialog, we proceed in the usual way to create the weights.



The connectivity histogram reveals the existence of 24 isolates, i.e., observations without neighbors. When selecting the left-most bar in the histogram, we can locate the points in the map (the black points, after changing the Color for Category to red for the regular points, and the highlight color to black for the selected observations). The selected points are indeed locations that are far away from the other points (more than 1500 feet). Since they are not included in the weights (in effect, the

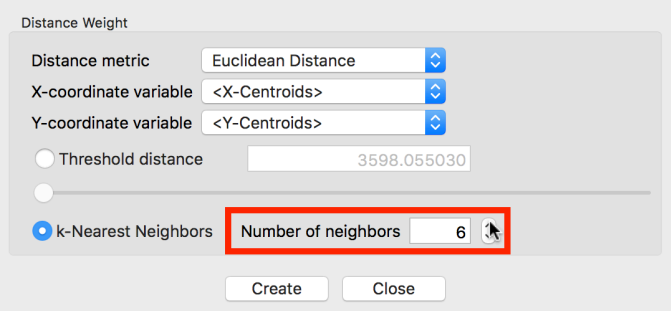
corresponding row in the spatial weights matrix consists of zero), these observations are not accounted for in any spatial analysis, such as tests for spatial autocorrelation, or spatial regression. For all practical purposes, they are removed from the analysis.



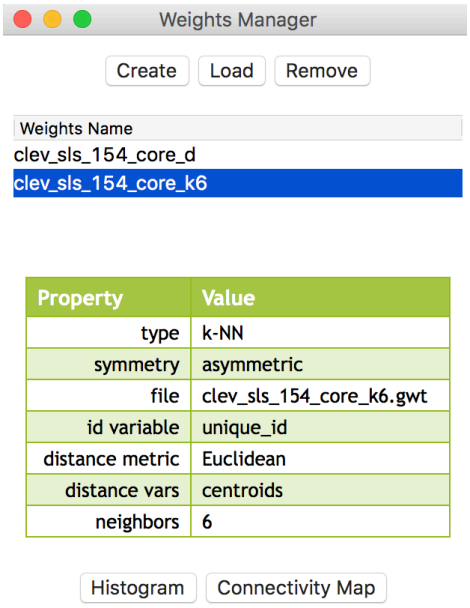


k-Nearest neighbor weights

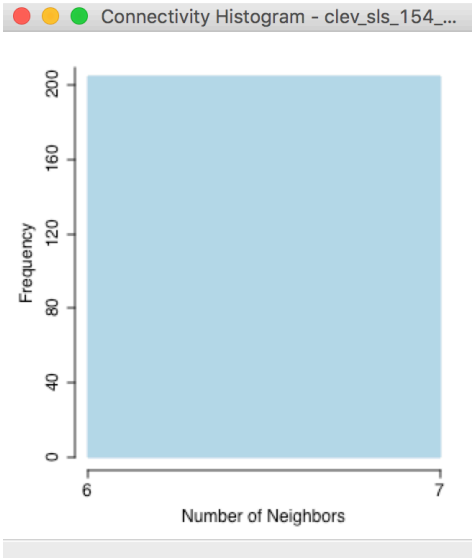
When the density of the locations is very uneven, a partial solution to the problems associated with distance-band weights is provided by k-nearest neighbor weights. In GeoDa, these are computed by selecting the corresponding radio button in the `Weights File Creation` interface. The value for the number of neighbors (k) is specified through the highlighted drop down list shown below. The default is 4, but in this example, we have selected 6.



Clicking on `Create` and specifying a file name will add the k-nearest neighbor weights to the `Weights Manager`.



Again, we can use the `Histogram` and the `Connectivity Map` to inspect the neighbor characteristics of the observations. However, in this case, the `Histogram` doesn't make much sense, since all observations have the same number of neighbors.

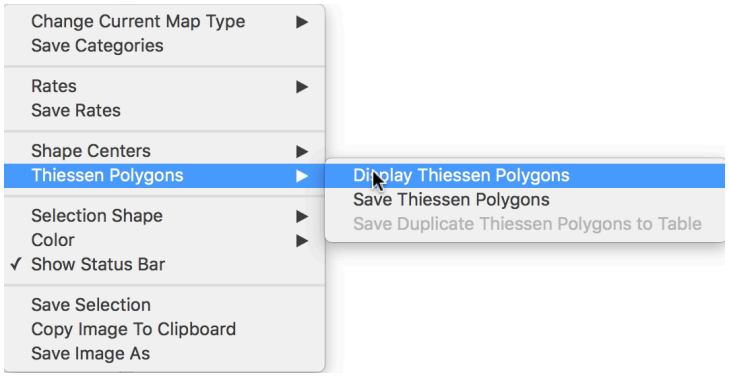


One drawback of the k-nearest neighbor approach is that it ignores the distances involved. The first k neighbors are selected, irrespective of how near or how far they may be. This suggests a notion of distance decay that is not absolute, but relative, in the sense of intervening opportunities (e.g., you consider the two closest grocery stores, irrespective of how far they may be).

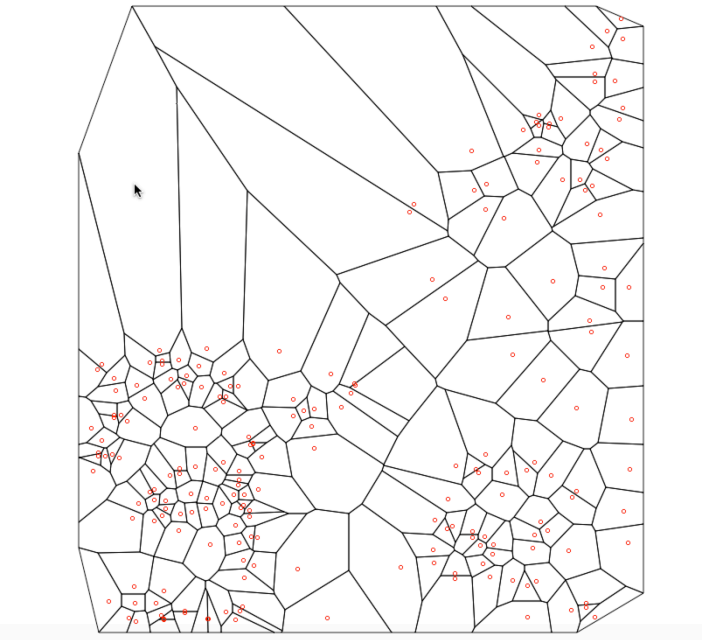
Finally, note that GeoDa deals with ties by randomly picking neighbors such that the list is exactly k (e.g., if there was a tie between neighbor 4 and 5 in a k=4 nearest neighbor case, one of the two would be picked at random).

Contiguity-based weights for points

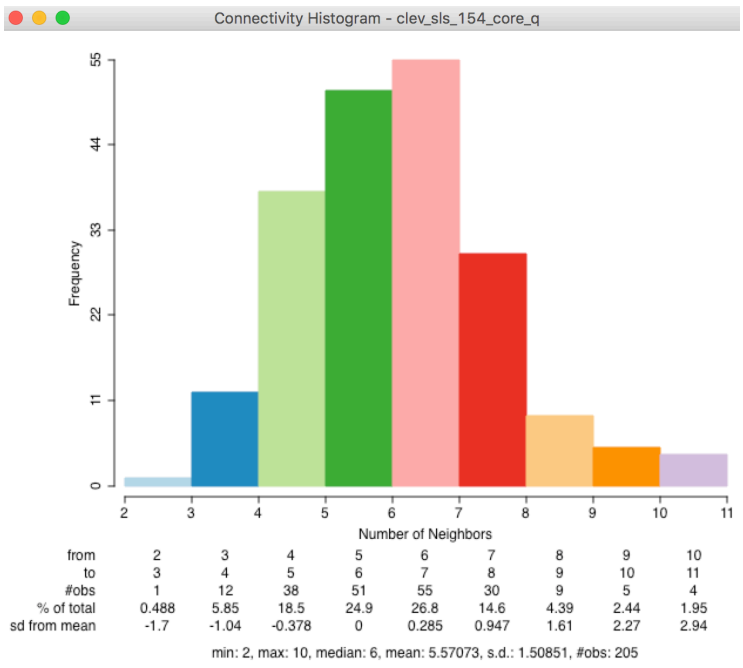
An alternative solution to deal with the problem of the uneven distribution of neighbor cardinality for distance-band weights is to compute a measure of contiguity. This is accomplished by turning the points into Thiessen polygons. This is a standard operation in GeoDa and can be invoked from the map view by selecting `Thiessen Polygons > Display Thiessen Polygons` in the options menu.



The result is a polygon map overlaid on the points.



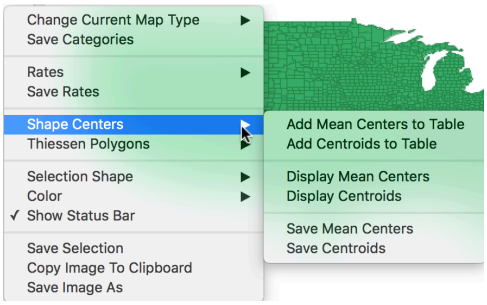
It is actually not necessary to create and display the Thiessen polygons to compute the contiguity weights, but they are shown here to illustrate the concept. In fact, when rook or queen contiguity is selected to create the weights for a point layer, the Thiessen polygons are constructed in the background and never actually shown. The weights creation proceeds in the usual way and the result is displayed in the `Weights Manager`. The associated connectivity histogram illustrates why this approach may be a useful alternative to distance-band or k-nearest neighbor weights (connectivity histogram shown with the statistics displayed).



The histogram represents a much more symmetric and compact distribution of the neighbor cardinalities, very similar to the shape of the histogram for contiguity between polygons. The median number of neighbors is 6 and the average 5.6, with a limited spread around these values. In many instances where the point distribution is highly uneven, this approach provides a useful compromise between the distance-band and the k-nearest neighbors.

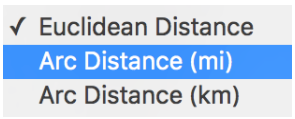
Distance-based weights for polygons

The converse of computing contiguity-based spatial weights for points is to create distance-based weights for polygons. The options in the map view provide several ways to compute, display and save so-called shape centers (either centroids or mean centers).

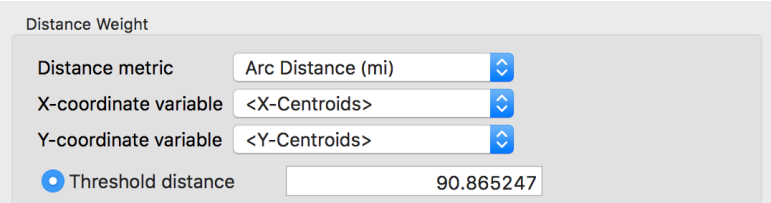


However, in the same way we proceeded for points, the actual calculation (or display) of the shape centers is hidden when creating distance-based weights for polygons. The interface is completely transparent and allows both contiguity and distance-based weights.

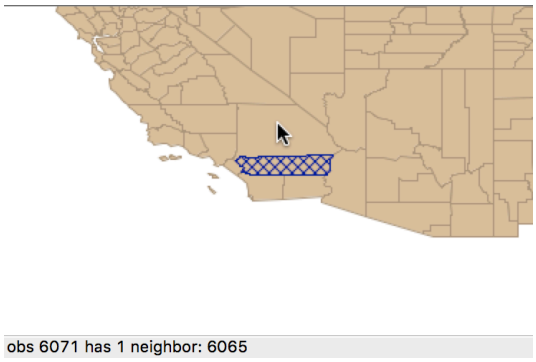
One complication that is often encountered is that the geographic layer is provided in latitude-longitude (i.e., the coordinates are unprojected). Consequently, the use of a straight line Euclidean distance is inappropriate (at least, for larger distances). Instead, the great circle distance or arc distance needs to be computed. So far, we have only considered Euclidean distance (the default), but the drop down list in the weights file creation interface also includes `Arc Distance` (in miles or in kilometers).



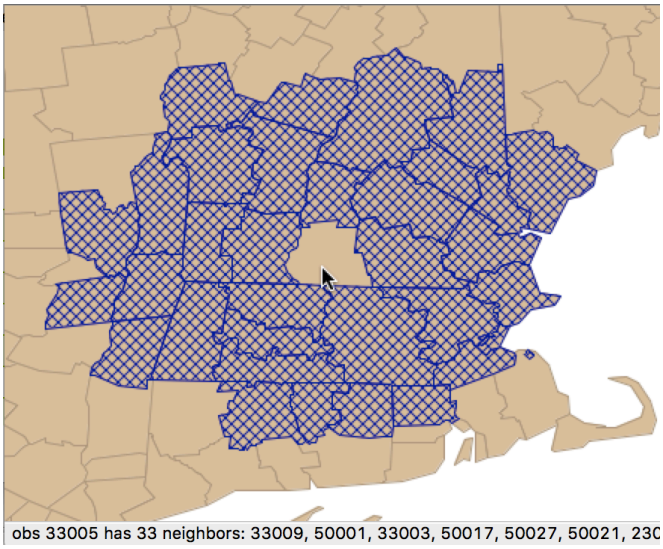
For example, using the `natregimes` data, we find that the max-min cut-off distance for distance-band weights for U.S. counties is about 91 miles. We can create a weights file using this criterion in the standard fashion.



The resulting weights clearly illustrate the pitfalls of using a distance-band when polygons (i.e., counties) are of widely varying sizes. For example, in the Connectivity Map for the U.S. counties, we find for San Bernardino county, CA (FIPS code 6071), the largest county (in area) in the U.S., that there is only one neighbor. In other words, the distance between the centroid of San Bernardino county and its nearest neighbor (Riverside county) drives the designation of neighbors for all the other counties in the U.S.



Clearly, this has major consequences for the smaller counties east of the Mississippi. As an illustration, consider Cheshire county (NH) with FIPS code 33005, which ends up with 33 neighbors using the 91 mile distance cut-off.



In practice, policy or theoretical considerations often dictate a given distance band (e.g., commuting distance). As we have seen, we need to be cautious before we uncritically translate these criteria into distance bands. Especially when the areal units in question are of widely varying sizes, there will be problems with the distribution of the neighbor cardinalities or isolates will be created when the distance is insufficiently large.

Neighbor lists and weights objects in spdep

We will use R to read the contents of GAL and GWT files and turn them into neighbor lists and list weights using the spdep package. We can assess the characteristics of these objects and visualize the connectivity structure by means of a graph. We also construct inverse distance weights.

The detailed commands with comments are contained in the `spatial_weights.ipynb` jupyter notebook (note that this is written with R beginners in mind, more seasoned R users may skip most of the comments). If you use RStudio, the commands are also included (without comments) in the file `spatial_weights.r`. You can load this file into the source editor pane of RStudio and execute each command with Control-Return (or Command-Return). Check the notebook for a more detailed explanation what the commands do.

The contents of the notebook follow below. This was generated from the jupyter notebook by downloading an RMarkdown file and then converting this file to a Word document using knitr in RStudio. Note that the lists look a little different from the way they look in the notebook, due to some strange formatting. Also, the long list outputs were shortened in the materials below.

Spatial weights characteristics

by Luc Anselin (anselin@uchicago.edu) (8/27/2016)

A brief introduction to the treatment of spatial weights in **R spdep**. The discussion is limited to reading weights information from **GAL** and **GWT** weights files created by GeoDa. This is my preferred approach.

While it is perfectly possible to create spatial weights in **R** from a shape file, this involves some fairly detailed insight into the various spatial data objects and file conversion functions in the **gdal** library, which is beyond the current scope. In practice, this will involve several steps, some of which are not straightforward. We refer for details to the discussion in Part I of the Bivand et al (2013) book, especially Chapters 2 and 4 and the "Creating Neighbours" vignette by Roger Bivand (on the **spdep** CRAN site).

For our purposes, we will create the weights in GeoDa, read the relevant files and then analyze the weights characteristics. This includes a plot of the connectedness structure. We will also illustrate how to construct inverse distance weights.

We will be using the **natregimes** and **clev_sls_154_core** sample files and the GAL and GWT weights created for them using GeoDa.

Note: this is written with R beginners in mind, more seasoned R users can probably skip most of the comments pertaining to data structures and basic commands.

For more extensive details about each function, see the spdep manual and vignettes.

Packages used:

- **spdep**
- **foreign**

Preliminaries

First, we need to load the **spdep** library. The required depencies **sp** and **Matrix** will be loaded as well.

Also, make sure the current working directory contains the dbf, GAL and GWT files for the two sample data sets.

```
library(spdep)

Loading required package: sp
Loading required package: Matrix
```

Creating a neighbor list from a GAL weights file

In spdep, there are two classes to deal with weights, one is a neighbor list, or class **nb**, the other a weights list, or class **listw**. Neighbor lists are lists of neighbors, but do not contain the actual weights. In this sense, they are similar to GAL and GWT files, which also only contain the neighbor relations. In GeoDa, this is converted into (row-standardized) weights internally. In most instances in spdep, you must create the weights object explicitly.

We start with the **read.gal** function with the default argument, i.e., the file name of the GAL file. We use **natregimes_q.gal** (queen contiguity for U.S. counties).

```
nb1 <- read.gal("natregimes_q.gal")

Error in read.gal("natregimes_q.gal"): GAL file IDs and region.id diffe
r
Traceback:


1. read.gal("natregimes_q.gal")

2. stop("GAL file IDs and region.id differ")
```

What happened? The default assumption in spdep is that the observation IDs are simple sequence number. However, in our natregimes example, we used **FIPSNO** as

the ID variable. In order to keep these ID values, we need to set **override.id = TRUE** in the arguments to the **read.gal** function.

```
nb1 <- read.gal("natregimes_q.gal",override.id=TRUE)
```

We can now list the contents of the neighbor list object. Note that spdep gives a summary of the characteristics, and not a listing of all the neighbor relations. The **summary** command gives slightly more extensive information.

```
nb1
```

```
Neighbour list object:
Number of regions: 3085
Number of nonzero links: 18168
Percentage nonzero weights: 0.190896
Average number of links: 5.889141
```

```
summary(nb1)
```

```
Neighbour list object:
Number of regions: 3085
Number of nonzero links: 18168
Percentage nonzero weights: 0.190896
Average number of links: 5.889141
Link number distribution:

   1    2    3    4    5    6    7    8    9   10   11   13   14
  24   36   91  281  620 1037  704  227   50   11    2    1    1
24 least connected regions:
53009 53029 25001 44005 36103 51840 51660 6041 51790 51820 51540 51560
6075 51580 51530 51131 51115 51770 51720 51690 51590 27031 26083 55029
with 1 link
1 most connected region:
49037 with 14 links
```

Internals of the neighbor list object

We can check some characteristics of the just created object using **class**, which yields **nb**, and **length**, which gives 3085.

```
class(nb1)
```

'nb'

```
length(nb1)
```

3085

A more comprehensive view of the structure of the neighbor list is given by the **str** command. We see that this object consists of several attributes, the most important of which is a list of 3085 lists that contain the sequence numbers (not the neigbhorhood IDs!) with the neighbors for each observation. In addition, the

neighbor list has several attributes extracted from the header line of the GAL file, such as the attribute **GeoDa**, which contains the name of the shape file (**shpfile**) and the region ID (**ind**). The actual region IDs are stored as characters in the **region.id** attribute.

```
str(nb1)
```

```
List of 3085
 $ : int [1:3] 23 31 41
 $ : int [1:3] 3 4 70
 $ : int [1:4] 2 5 63 70
 $ : int [1:7] 2 28 32 43 56 69 70
 $ : int [1:4] 3 6 29 63


. . .

 $ : int [1:8] 41 73 100 121 129 130 173 174
 $ : int [1:4] 160 161 169 229
 $ : int [1:5] 47 79 82 119 138
  [list output truncated]
 - attr(*, "class")= chr "nb"
 - attr(*, "region.id")= chr [1:3085] "27077" "53019" "53065" "53047" .
..
 - attr(*, "GeoDa")=List of 2
  ..$ shpfile: chr "natregimes"
  ..$ ind    : chr "FIPSNO"
 - attr(*, "gal")= logi TRUE
 - attr(*, "call")= logi TRUE
 - attr(*, "sym")= logi TRUE
```

Since the neighbor list is a list of lists, you can extract any element using the double bracket [[ ]] notation.

For example, the first element yields the sequence numbers 23, 31 and 41 as neigbhors.

```
nb1[[1]]
```

```
<li>23</li>
<li>31</li>
<li>41</li>
```

To extract the actual IDs that correspond to this, we need to keep in mind the special structure of the neighbor list object. For example, the following doesn't work.

```
nb1$region.id
```

```
NULL
```

Instead, we need to extract the **region.id attribute** using the **attr** command. This takes two arguments, the object of interest (w1) and the type of attribute, as a string ("region.id").

We use **head** to list the first few elements.

```
w1att <- attr(nb1,"region.id")
head(w1att)
```

```
<li>'27077'</li>
<li>'53019'</li>
<li>'53065'</li>
<li>'53047'</li>
<li>'53051'</li>
<li>'16021'</li>
```

Now, we can extract the ID values that correspond to the sequence numbers for the neighbors of the first observation.

```
w1att[1]
w1att[c(23,31,41)]
```

'27077'

```
<li>'27135'</li>
<li>'27071'</li>
<li>'27007'</li>
```

This matches the entry in the natregimes_q.gal file.

Keep in mind that these are strings (note the quotes) and not integers. This may matter in later manipulations.

Check for symmetry

Note how one of the attributes of the neighbor list is **sym**. This is a logical variable that shows whether or not the weights are symmetric. Since our example is for queen contiguity, this is indeed the case. We can extract this attribute using the **attr** command, as we did before, but now specifying "sym".

```
attr(nb1,"sym")
```

TRUE

There is also a built-in function **is.symmetric.nb** which accomplishes the same result.

```
is.symmetric.nb(nb1)
```

TRUE

Creating a neighbor list from a GWT weights file

Creating a neighbor list from a **GWT** file is slightly more complex. It requires the function **read.gwt2nb**. Again, this takes as argument the file name. A second argument pertains to the **region.id**. The default is that no **region.id** is specified, and simple sequence numbers are used. However, when a **region.id** is specified, this variable must be available in the workspace, unlike what is the case for **GAL** files. This implies that we first have to read the **dbf** file associated with the shape (and weights), for which we need the **foreign** package.

We first load the library, then pass **clev_sls_154_core.dbf** as the argument to the **read.dbf** function, and finally summarize the resulting data frame.

```
library(foreign)
dat2 <- read.dbf("clev_sls_154_core.dbf")
summary(dat2)
```

```
   unique_id             parcel          x                 y          
 Min.   :  1183   002-02-036:  1   Min.   :2176680   Min.   :647224  
 1st Qu.:  8837   002-02-053:  1   1st Qu.:2182510   1st Qu.:652774  
 Median : 21504   002-14-053:  1   Median :2189250   Median :657074  
 Mean   : 36650   002-15-038:  1   Mean   :2192729   Mean   :659472  
 3rd Qu.: 62055   002-15-043:  1   3rd Qu.:2204140   3rd Qu.:663162  
 Max.   :168806   002-16-003:  1   Max.   :2210280   Max.   :684677  
                  (Other)   :199                                     
   sale_price       tract10int        Quarter      year1        yrquart
er
 Min.   :  1049   Min.   :101200   Min.   :4   Min.   :2015   Min.   :1
54
 1st Qu.:  9000   1st Qu.:104200   1st Qu.:4   1st Qu.:2015   1st Qu.:1
54
 Median : 20000   Median :106100   Median :4   Median :2015   Median :1
54
 Mean   : 41897   Mean   :111505   Mean   :4   Mean   :2015   Mean   :1
54
 3rd Qu.: 48500   3rd Qu.:113801   3rd Qu.:4   3rd Qu.:2015   3rd Qu.:1
54
 Max.   :527409   Max.   :196500   Max.   :4   Max.   :2015   Max.   :1
54
```

The ID variable is **unique_id**. We use the **read.gwt2nb** command with the filename **clev_sls_154_core_d.gwt** (distance-band weights using the default max-min cut-off) and with region.id = dat2$unique_id. If you forget to specify the region.id, the default of NULL will be used, which will bring up a warning message and create an empty neighbor object.

```
nb2 <- read.gwt2nb("clev_sls_154_core_d.gwt",region.id=dat2$unique_id)
nb2
```

```
Warning message:
In read.gwt2nb("clev_sls_154_core_d.gwt", region.id = dat2$unique_id):
region.id not named unique_id


Neighbour list object:
Number of regions: 205
Number of nonzero links: 2592
Percentage nonzero weights: 6.167757
Average number of links: 12.6439
```

Note the strange warning message. Everything works fine, but this seems to suggest that unique_id is not in the workspace. This is likely because we referred to it as dat2$unique_id. If instead we first **attach** the data frame dat2, which makes the variable unique_id available, we don't get the warning.

The use of **attach** and **detach** is discouraged in some R circles, but as long as you only work with one data set at a time, it is not too problematic.

So, after attaching **dat2**, we re-run the **read.gwt2nb** command, and all is well. We summarize the neighbor list object as we did before.

```
attach(dat2)
nb3 <- read.gwt2nb("clev_sls_154_core_d.gwt",region.id=unique_id)
summary(nb3)

Neighbour list object:
Number of regions: 205
Number of nonzero links: 2592
Percentage nonzero weights: 6.167757
Average number of links: 12.6439
Link number distribution:

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26
 6  6  9  5  5 10  8 10 13 12 11  6 17  8  9 11 13  6 10  7  2  4  6  2
  1  1
27 28 29 30 32
 1  2  1  1  2
6 least connected regions:
11359 40925 40961 41083 41161 77675 with 1 link
2 most connected regions:
19195 19785 with 32 links
```

The internal structure of the neighbor list created from a GWT file is the same as for a GAL file, with one exception. The **GeoDa** attribute now also includes the actual distances between neighbors. For each observation, this is a list of distance values.

So, the three components of the GWT file are included in three separate items in the neighbor list object. First, is the list of neighbors as sequence numbers. This can be

extracted by means of the double bracket notation. For example, for the first observation, which has 9 neighbors, this gives:

```
n1 <- nb3[[1]]
n1
```

```
<li>2</li>
<li>6</li>
<li>7</li>
<li>8</li>
<li>9</li>
<li>10</li>
<li>31</li>
<li>32</li>
<li>34</li>
```

Next are the IDs that correspond to this. As before, we create a vector with these values by extracting the "region.id" attribute. Note that, in contrast to what happens for GAL files, the region.id values are integers and not characters. This is because they are extracted explicitly from the data frame, where they are stored as integers.

```
w3att <- attr(nb3,"region.id")
head(w3att)
```

```
<li>1183</li>
<li>1198</li>
<li>1516</li>
<li>1606</li>
<li>1612</li>
<li>1624</li>
```

Now, we can list the ID of the first observation, together with the IDs of its neighbors.

```
w3att[1]
w3att[n1]
```

1183

```
<li>1198</li>
<li>1624</li>
<li>1741</li>
<li>2024</li>
<li>2170</li>
<li>2341</li>
<li>6842</li>
<li>6845</li>
<li>7058</li>
```

Finally, we extract the corresponding distances. These are part of the "GeoDa" attribute. This itself is a list with three components, **dist**, a list of lists of distance

values, **shpfile** and **ind** (same as for GAL files). You can see the details of the structure by means of the **str** command (omitted here because of its lengthy output). The expression may seem somewhat complex, but there is a logic to it. First, we extract the "GeoDa" attribute. From that, we select the $dist component, and finally, we take the first element [1] of that. The result is a list with the distances.

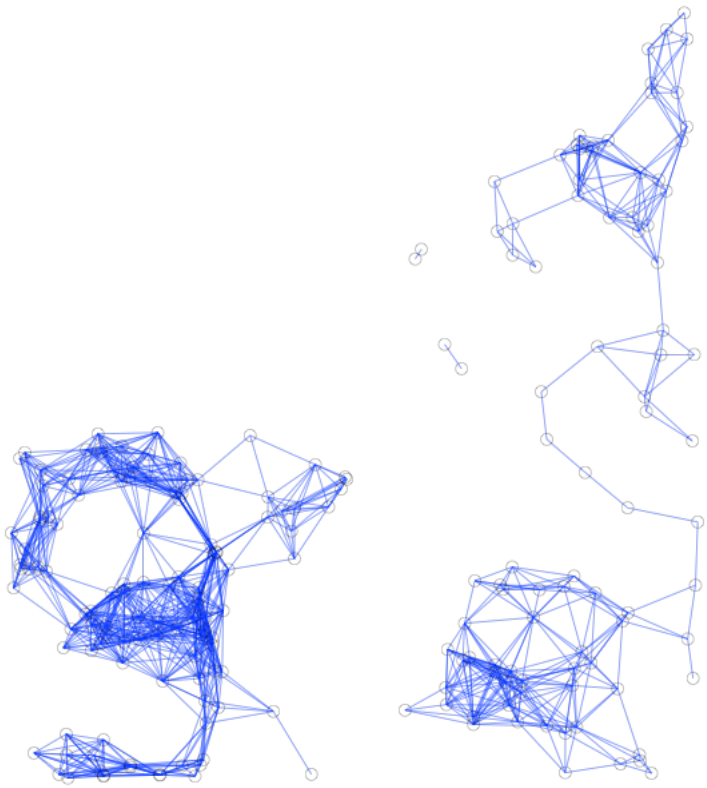
```
d <- attr(nb3,"GeoDa")$dist[1]
d
```

```
<li><ol class=list-inline>
<li>385.161005</li>
<li>3013.07086</li>
<li>1160.31203</li>
<li>1858.90398</li>
<li>3367.15013</li>
<li>2525.50272</li>
<li>3253.02459</li>
<li>3390.73458</li>
<li>3369.6439</li>
```

Plotting the structure of a neighbor list

In addition to the standard print and summary functions, neighbor lists can also be plotted as a graph. This provides a visual summary of the connectedness structure. The **plot** function requires a neighbor list argument and a matching n by 2 matrix of point coordinates that will serve as nodes in the graph. Since we have attached the data set, column-binding (**cbind**) the variables **x** and **y** will provide the coordinates. We can use all the standard plotting arguments to embellish the plot. In our example, the line width (**lwd**) is set to 0.2 (the default crowds the plot too much), and just for fun, the color (**col**) is set to "blue". This can be made much more nicer looking by adding titles, etc., which we don't pursue here.

```
coordn <- cbind(x,y)
plot(nb3,coordn,lwd=0.2,col="blue")
```



The plot reveals a very interesting structure, where the data are essentially split into two large unconnected subgraphs, with two sets of two observations unconnected from either. Since we used the default max-min distance band, every node is connected to another node.

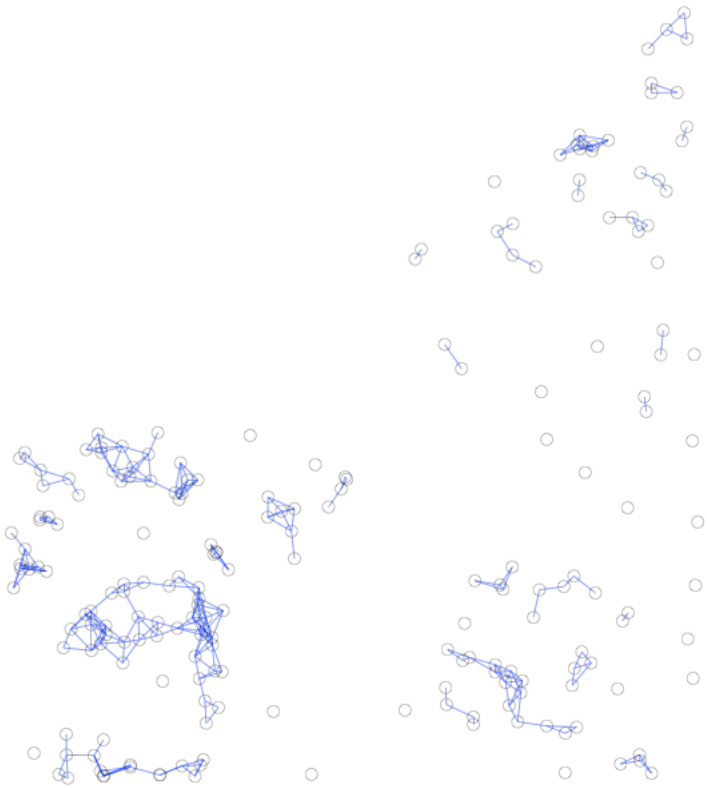
To illustrate the treatment of isolates (unconnected observations), we read in the file **clev_sls_154_core_d1500.gwt**, which used a distance band of 1500ft, clearly less than the max-min distance. Several warning messages are generated and the summary reveals 24 regions with no links.

```
nb4 <- read.gwt2nb("clev_sls_154_core_d1500.gwt",region.id=unique_id)
summary(nb4)

Warning message:
In read.gwt2nb("clev_sls_154_core_d1500.gwt", region.id = unique_id): 3
754, 4072, 7404, 10114, 11359, 20493, 41281, 59428, 59649, 61351, 62046
, 62055, 62122, 63759, 63840, 64585, 65975, 70548, 73001, 73581, 76214,
```

```
 77675, 140008, 168450 are not originsWarning message:
In read.gwt2nb("clev_sls_154_core_d1500.gwt", region.id = unique_id): 3
754, 4072, 7404, 10114, 11359, 20493, 41281, 59428, 59649, 61351, 62046
, 62055, 62122, 63759, 63840, 64585, 65975, 70548, 73001, 73581, 76214,
 77675, 140008, 168450 are not destinations

Neighbour list object:
Number of regions: 205
Number of nonzero links: 624
Percentage nonzero weights: 1.48483
Average number of links: 3.043902
24 regions with no links:
3754 4072 7404 10114 11359 20493 41281 59428 59649 61351 62046 62055 62
122 63759 63840 64585 65975 70548 73001 73581 76214 77675 140008 168450
Link number distribution:

 0  1  2  3  4  5  6  7  8  9 10
24 31 36 35 24 27 18  6  2  1  1
31 least connected regions:
2170 3133 4527 6584 9372 16351 16561 40925 40961 41083 41161 41467 4158
4 42418 42961 43375 44020 44062 44502 46656 46704 59709 60087 63399 636
14 63651 71170 73339 73429 168544 168726 with 1 link
1 most connected region:
19560 with 10 links
```

The plot clearly illustrates the much sparser structure of the network as well as the location of the isolates. These are points that are more than 1500ft from their nearest neighbor. In general, the graph has broken down into several small unconnected subgraphs.

```
plot(nb4,coordn,lwd=0.2,col="blue")
```



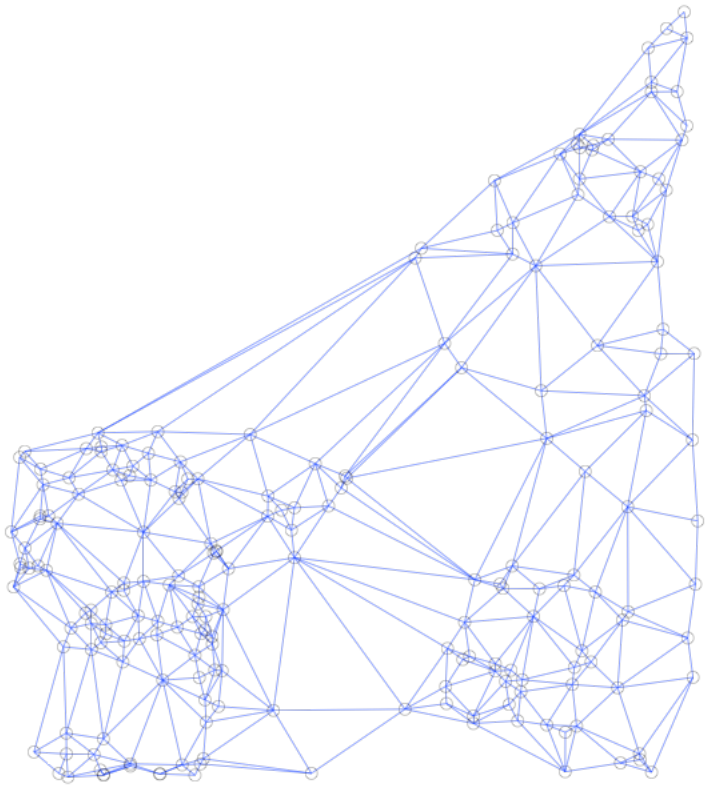
Finally, to illustrate the much more balanced weights structure we obtained by applying contiguity to the Thiessen polygons, we read in the GAL file with those weights: **cleav_sls_154_core_q.gal** and summarize its properties.

```
nb5 <- read.gal("clev_sls_154_core_q.gal",region.id=unique_id)
summary(nb5)

Neighbour list object:
Number of regions: 205
Number of nonzero links: 1142
Percentage nonzero weights: 2.71743
Average number of links: 5.570732
Link number distribution:

 2  3  4  5  6  7  8  9 10
 1 12 38 51 55 30  9  5  4
1 least connected region:
```

```
44802 with 2 links
4 most connected regions:
7404 9372 19758 20493 with 10 links
```

The plot reveals a fully connected graph with most nodes having about average links (from the summary, we see that the links range from 2 to 10).

```
plot(nb5,coordn,lwd=0.2,col="blue")
```



Constructing the weights

Most of the spatial analytical operations we will cover require actual weights objects, and not just neighbor lists. The proper weights object is of the type **listw**. We convert a neighbor list to an actual weights list by means of the **nb2listw** command.

The default settings are usually OK, unless there are isolates in the data set. In that case, you want to make sure to set the **zero.policy** argument to **TRUE** (the default is **FALSE**, which stops the program whenever isolates are encountered). The default value for the **style** argument is **'W'**, which corresponds to row-standardization, so that is usually what we want.

To illustrate the effect of the **zero.policy** argument, we will convert nb4, the neighbor list with many isolates. Since we use the defaults for all arguments, the only item that needs to be specified is the neighbor list object, **nb4**. Next, we set **zero.policy** to **TRUE**. Note that we also have to specify this option for the **summary** command, otherwise it will generate an error message.

```
w4 <- nb2listw(nb4)
summary(w4)

Error in nb2listw(nb4): Empty neighbour sets found
Traceback:


1. nb2listw(nb4)

2. stop("Empty neighbour sets found")

w4a <- nb2listw(nb4,zero.policy=TRUE)
summary(w4a,zero.policy=TRUE)

Characteristics of weights list object:
Neighbour list object:
Number of regions: 205
Number of nonzero links: 624
Percentage nonzero weights: 1.48483
Average number of links: 3.043902
24 regions with no links:
3754 4072 7404 10114 11359 20493 41281 59428 59649 61351 62046 62055 62
122 63759 63840 64585 65975 70548 73001 73581 76214 77675 140008 168450
Link number distribution:

 0  1  2  3  4  5  6  7  8  9 10
24 31 36 35 24 27 18  6  2  1  1
31 least connected regions:
2170 3133 4527 6584 9372 16351 16561 40925 40961 41083 41161 41467 4158
4 42418 42961 43375 44020 44062 44502 46656 46704 59709 60087 63399 636
14 63651 71170 73339 73429 168544 168726 with 1 link
1 most connected region:
19560 with 10 links

Weights style: W
Weights constants summary:
    n    nn  S0       S1       S2
W 181 32761 181 143.5363 749.5522
```

In addition to the same connectedness statistics as for a neighbor list, the summary now also includes several numeric values that are used in the computation of tests for spatial autocorrelation and in spatial regression models. This includes the threesome S0, S1 and S2, which are various summations of the weights. S0 in particular is the sum of all the weights. For row-standardized weights (style="W"), this should add up to the number of observations. However, in this case, since there are 24 isolates, it adds up to 205-24=181. Note that n is listed as 181, which is the number of connected observations, not the size of the data set (205).

Now, lets move away from the isolates case and convert the neighbor list with queen contiguities for the Cleveland locations, nb5. Since there are no isolates, we can use the default setting.

```
w5 <- nb2listw(nb5)
summary(w5)

Characteristics of weights list object:
Neighbour list object:
Number of regions: 205
Number of nonzero links: 1142
Percentage nonzero weights: 2.71743
Average number of links: 5.570732
Link number distribution:

 2  3  4  5  6  7  8  9 10
 1 12 38 51 55 30  9  5  4
1 least connected region:
44802 with 2 links
4 most connected regions:
7404 9372 19758 20493 with 10 links

Weights style: W
Weights constants summary:
    n    nn  S0       S1      S2
W 205 42025 205 76.82244 835.436
```

Note how now both n and S0 equal 205.

Internals of a weights object

The weights object contains all the components of a neighbor list as well as the actual spatial weights values. The latter are stored as a list of lists (one for each observation) in the **weights** attribute. We can look at the detailed structure of the object using the **str** command, but we will skip that since the output is very long (it includes all the neighbors and all the weights).

We can access individual weights lists by means of either the $weights notation, or by using double brackets [[3]], followed by the index of the observation of interest in [ ].

The two equivalent approaches are illustrated for the first obsevation.

```
w5$weights[1]

<li><ol class=list-inline>
<li>0.25</li>
<li>0.25</li>
<li>0.25</li>
<li>0.25</li>

w5[[3]][1]

<li><ol class=list-inline>
<li>0.25</li>
<li>0.25</li>
<li>0.25</li>
<li>0.25</li>
```

So far, we have used the default of row-standardized weights. If we want the unstandardized (binary) weights, we need to specify the **style** as **B** in the **nb2listw** command. We illustrate this with the same **nb5** neighbor list.

```
w5a <- nb2listw(nb5,style="B")
summary(w5a)

Characteristics of weights list object:
Neighbour list object:
Number of regions: 205
Number of nonzero links: 1142
Percentage nonzero weights: 2.71743
Average number of links: 5.570732
Link number distribution:

 2  3  4  5  6  7  8  9 10
 1 12 38 51 55 30  9  5  4
1 least connected region:
44802 with 2 links
4 most connected regions:
7404 9372 19758 20493 with 10 links

Weights style: B
Weights constants summary:
    n    nn   S0   S1    S2
B 205 42025 1142 2284 27304
```

The connectivity characteristics of the weights are the same as before, but the numerical properties are not. Note for example, that S0 is now 1142, the total number of non-zero weights.

We can access the actual weights values in the same way as before. The values of the weights are all 1 (since the zero values are not stored).

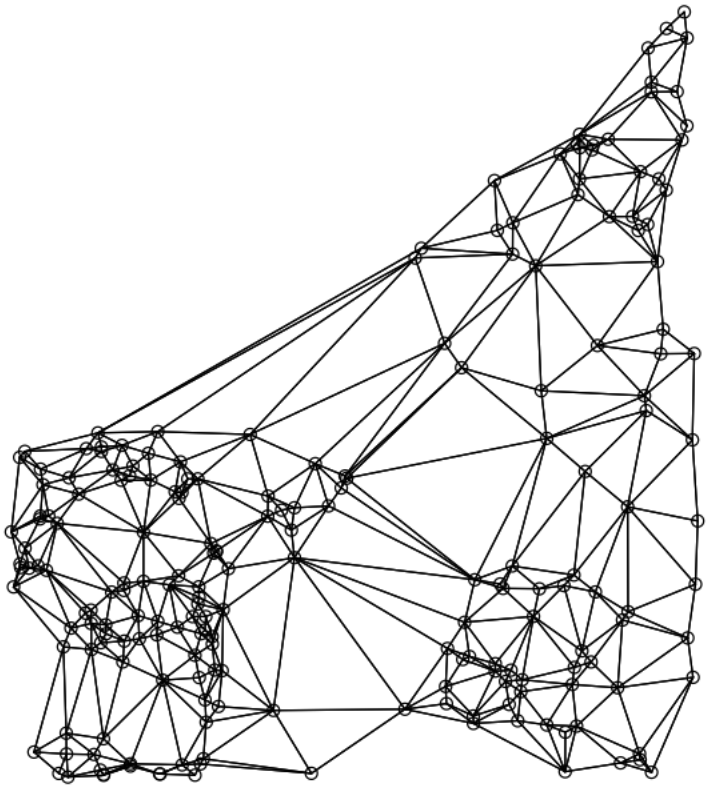
```
w5a$weights[1]
```

```
<li><ol class=list-inline>
<li>1</li>
<li>1</li>
<li>1</li>
<li>1</li>
```

Plotting a weights object

Since a weights object contains a neighbor list, its network structure can be plotted in the same way. We need to pass the list weights object and a matrix of coordinates. The plot is identical to that of the neighbor list.

```
plot(w5,coordn)
```



Inverse distance weights

In spdep, there is no explicit function to create inverse distance weights. However, there is a way to specify the weights directly in the **nb2listw** function, in the attribute **glist**. As long as we can extract the distances from a neighbor list object (e.g., one that was read from a GWT file), we can compute the inverse distances and pass them as weights through the **glist** argument. Note that one could also compute these distances from the coordinates in R itself (using **nbdists**), but that's beyond our current scope. Also, this approach can be readily extended to any other function of the distances, such as inverse square distance, or negative exponential.

We will use the default distance band weights for the Cleveland points, which we turned into the neighbor list **nb3**. We first extract the list of lists of distances (one list for each observation) using the **attr** function, specifying **GeoDa** as the attribute, and selecting the component **dist**. We see that this is indeed a list with the proper length.

```
nd3 <- attr(nb3,"GeoDa")$dist
length(nd3)
```

205

The first element of this list is itself a list with the distances to the neighbors of the first observation.

```
nd3[1]
```

```
<li><ol class=list-inline>
<li>385.161005</li>
<li>3013.07086</li>
<li>1160.31203</li>
<li>1858.90398</li>
<li>3367.15013</li>
<li>2525.50272</li>
<li>3253.02459</li>
<li>3390.73458</li>
<li>3369.6439</li>
```

Now we get fancy and use one of the most powerful features of R, one of the family of **apply** functions. We could of course write this as a loop and go through the list for each observation in turn, but R has very powerful vectorizing functions that speed this process up considerably (this really matters when you have a lot of observations). The principle is that an anonymous function (i.e., a function without a name) will be applied to each element of the list in turn, without having to explicitly write a loop.

In our example, we will use **lapply** (think of it as apply a function to the elements of a list). We specify the list of lists as the first argument, and the anonymous function as the second. Generically, this is specified as **function(x)**, followed by the particular manipulation we want to apply to x. In our case, this is simply the inverse, **1/x**.

The full command is then as shown below.

```
invd3 <- lapply(nd3,function(x) (1/x))

length(invd3)
```

205

We see that **invd3** has the proper length, and we can also check its first elements. Each value is exactly the inverse of the distance values we had earlier.

```
invd3[1]

<li><ol class=list-inline>
<li>0.00259631683119115</li>
<li>0.000331887315786526</li>
<li>0.000861837138756546</li>
<li>0.000537951400803392</li>
<li>0.000296987054747096</li>
<li>0.00039596076934734</li>
<li>0.000307406222219796</li>
<li>0.000294921344153101</li>
<li>0.000296767263745585</li>
```

Note that these values are very small (they would be even smaller if we applied inverse distance squared). In practice, this often constitutes a problem and it is related to the scale in which the distance is expressed. In our case, these are feet, so that the values can be quite large, and as a result, their inverse quite small, even to the point of being hardly distinguishable from zero (for squared inverse distance).

To correct for this scale dependence, we may want to rescale the distances in the process, for example, dividing them all by 100 (you want to avoid ending up with inverse distances that are larger than 1, which may happen if you divide by too large a number, such as 1000).

The new scaled inverse distances are computed in the same way, with only a slight adjustment to the anonymous function.

```
invd3a <- lapply(nd3,function(x) (1/(x/100)))
invd3a[1]

<li><ol class=list-inline>
<li>0.259631683119115</li>
<li>0.0331887315786526</li>
<li>0.0861837138756546</li>
<li>0.0537951400803392</li>
<li>0.0296987054747096</li>
<li>0.039596076934734</li>
<li>0.0307406222219796</li>
<li>0.0294921344153101</li>
<li>0.0296767263745585</li>
```

We now create the corresponding spatial weights object by means of the **nb2listw** function, but passing our inverse distances **invd3a** as the value for the argument **glist**. If we want to keep the actual inverse distances (which we typically do), we need to also specify the **style** as **B**. Otherwise, the default row-standardization will be applied to the inverse distance values.

The new weights object is then created as:

```
w6 <- nb2listw(nb3,glist=invd3a,style="B")

summary(w6)

Characteristics of weights list object:
Neighbour list object:
Number of regions: 205
Number of nonzero links: 2592
Percentage nonzero weights: 6.167757
Average number of links: 12.6439
Link number distribution:

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26
 6  6  9  5  5 10  8 10 13 12 11  6 17  8  9 11 13  6 10  7  2  4  6  2
  1  1
27 28 29 30 32
 1  2  1  1  2
6 least connected regions:
11359 40925 40961 41083 41161 77675 with 1 link
2 most connected regions:
19195 19785 with 32 links

Weights style: B
Weights constants summary:
    n    nn       S0       S1       S2
B 205 42025 180.2882 145.9202 1018.442
```

We see that the connectedness structure is again the same as before, but the quantitative weights properties are different. For example, the value of S0 no longer has a relation to either the number of observations or the number of non-zero links, but it is the sum of all the inverse distance weights.

The weights themselves can be extracted as before. They are exactly the inverse distances we computed.

```
w6$weights[1]

<li><ol class=list-inline>
<li>0.259631683119115</li>
<li>0.0331887315786526</li>
<li>0.0861837138756546</li>
<li>0.0537951400803392</li>
```

```
<li>0.0296987054747096</li>
<li>0.039596076934734</li>
<li>0.0307406222219796</li>
<li>0.0294921344153101</li>
<li>0.0296767263745585</li>
```