**Network Voronoi diagram with weighted links**

**User’s Guide**

(Version 1.1 updated on 10/26/2013)

Table of Contents

[1. Introduction 1](#_Toc353960624)

[2. Setup 2](#_Toc353960625)

[3. N-VD with no weighted links 2](#_Toc353960626)

[3.1 No generator weight 3](#_Toc353960627)

[3.2 Additive weight of the generator points 3](#_Toc353960628)

[3.3 Multiplicative weight of the generator points 4](#_Toc353960629)

[4. N-KDE-VD 4](#_Toc353960630)

[4.1 KDE output 5](#_Toc353960631)

[4.2 N-KDE-VD with linear normalization 6](#_Toc353960632)

[4.3 N-KDE-VD with quantile normalization 6](#_Toc353960633)

[5. N-ILINCS-VD 7](#_Toc353960634)

[5.1 ILINCS output 8](#_Toc353960635)

[5.2 N-ILINCS-VD with linear normalization 8](#_Toc353960636)

[5.3 N-ILINCS-VD with quantile normalization 9](#_Toc353960637)

# Introduction

This tool is a Python library for computing network Voronoi diagram with weighted links (N-WL-VD) based on local-scale clustering analysis. A network Voronoi diagram (N-VD) replaces the Euclidean distance in the planar Voronoi with shortest-path distance defined in the network space. Traditional weighted N-VDs consider weights of the generator points. This library incorporates weights of the links into the diagram construction. The link weights are calculated based on local-scale clustering analysis with a separate event dataset. The clustering level can either be calculated from network-constrained kernel density estimation or local Moran's I, the constructed diagram are thus named N-KDE-VD and N-ILINCS-VD, respectively.

The library is written in a single python module ***“netvoronoi\_cluster.py”***, which relies on PySAL 1.4 or higher[[1]](#footnote-1). Particularly, it uses functions in *network.py, kernal.py* and *lincs.py* in the package *pysal.contrib.network*. These modules cannot be directly be invoked in current version of PySAL (possibly because they’re not matured yet), so the files copied into the same folder of *netvoronoi\_cluster.py*.

# Setup

To invoke the procedures in Section 3, 4 and 5, first working directory needs to be changed to where "netvoronoi.py" resides using *os.chdir* command, then relevant modules should be imported:

**>>>import os**

**>>>os.chdir("F:\program\\python\\network\\src")**

**>>>import netvoronoi\_cluster as nv**

**>>>import network as pynet**

The functions are demonstrated using a dataset in PySAL in the following sections, shown in Figure 1. The stations points are 26 randomly selected points from the original event points. Note that all the input shape files have to be projected first with the same projection system.

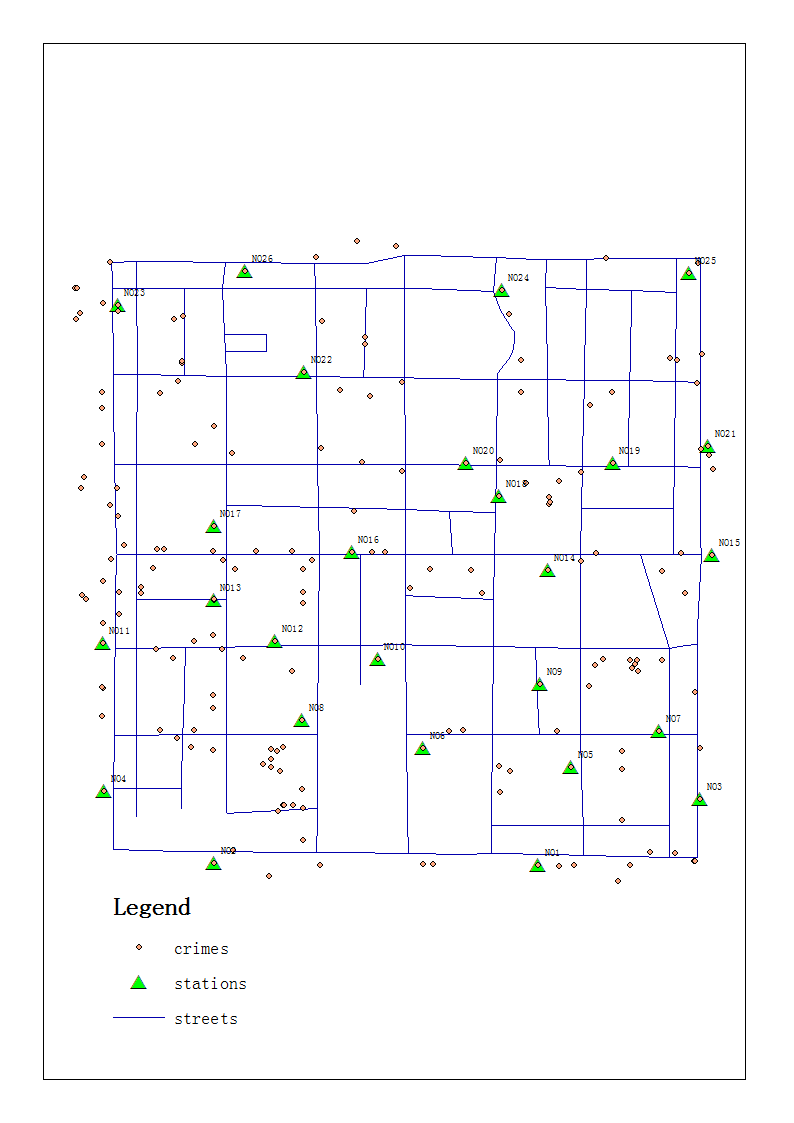


Figure 1. The test dataset

# N-VD with no weighted links

The library can compute normal N-VDs with weights defined for the generator points. The weight can either be additive or multiplicative.

For N-VD with no weighed link, we need to first load the network *G* from a shp file, and mesh the network if needed (partition the links into equal-length segments). The following code snippet reads in the *streets* and parses it using a segment width of 200m.

**>>>G = pynet.read\_network('../data/pysal/streets.shp')**

**>>>G = pynet.mesh\_network(G, 200)**

## No generator weight

This is the simplest case with no weights for the generator points. *ID\_test*is the identifier of the generator points.

**>>>\_=\_=nv.netvoronoi(G, '../data/pysal/stations.shp', "../data/pysal/noevent/netvoronoi\_no\_wgt.shp","ID\_test")**

The terminal will generate the following output during execution, and save the output to a shapefile *"../data/pysal/noevent/netvoronoi\_no\_wgt.shp"*. The time spent varies in different computers.

**>>>**set up

Reading stations ../data/pysal/stations.shp: 0.003 seconds

Collide count: 0

Snapping stations to the network: 0.319 seconds

Finding the nearest event for every node: 0.151 seconds

Finding the nearest event for every edge (or new ones): 0.006 seconds

Number of nodes that are not linked to the main component: 0

Writing the result out ../data/pysal/noevent/netvoronoi\_no\_wgt.shp: 0.075 seconds

Done

The file can be symbolized in ArcMap using the unique value categorization on the "station" field.

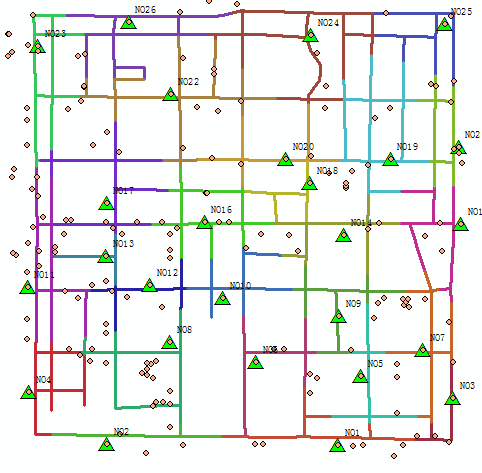


Figure 2. Ordinary N-VD with no weights

## Additive weight of the generator points

The same method call with 3.1, with an extra field *add\_wgt* needed representing the additive weight.

**>>>\_=\_=nv.netvoronoi(G, '../data/pysal/stations.shp', '../data/pysal/noevent/netvoronoi\_add\_wgt.shp', 'ID\_test', 'add\_wgt')**

The terminal will generate similar output as in 3.1. The result diagram is shown in Figure 3.

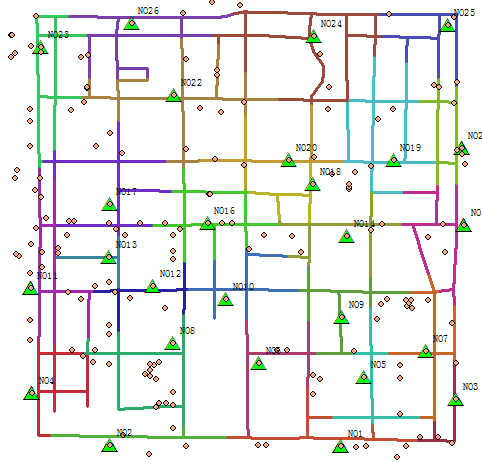
****

Figure 3. Additive weighted ordinary N-VD

## Multiplicative weight of the generator points

The same method call with 3.1, with an extra field *mul\_wgt* needed representing the multiplicative weight.

**>>>\_=nv.netvoronoi(G, '../data/pysal/stations.shp', '../data/pysal/noevent/netvoronoi\_mul\_wgt.shp', 'ID\_test',None,'mul\_wgt')**

The terminal will generate similar output as in 3.1. The result diagram is shown in Figure 4.

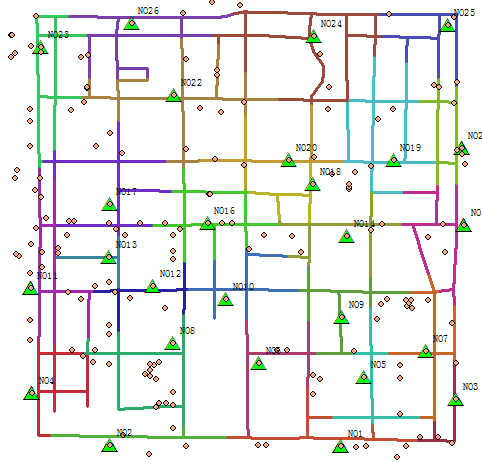
****

Figure 4. Multiplicatively weighted ordinary N-VD

The two types of weights can also be combined using the same method.

# N-KDE-VD

The N-KDE-VD uses the kernel density estimation as an indicator of the clustering level at the link level. The idea is to see the clustering level as a measure of accessibility or load in the network. By transforming the clustering level into weights of the links, we can let the underlying spatial point processes influence the result Voronoi partition process.

The kernel density estimation process needs to be done at first. The *net\_density* method takes the event and the network as input and generates the transformed network. Other parameters include the *segment width, KDE bandwidth, kernel function type, link weighting mode (additive or multiplicative), normalize method and interval*. Two types of normalization method are implemented: *linear* and *quantile*.

## KDE output

We can write the KDE result to a shapefile and visualize it afterwards.

**>>>\_, G\_origin, density\_dic = nv.net\_density('../data/pysal/streets.shp', '../data/pysal/crimes.shp', 200, 300)**

**>>>nv.write\_kde\_to\_shp(G\_origin, density\_dic, '../data/pysal/kde/kernal.shp')**

The result is written into '../data/pysal/kde/kernal.shp', shown in Figure 5. It may seem odd that some links have high density but there seems to be only one point nearby. *It’s because this test dataset has many points that locate in exactly the same position.*

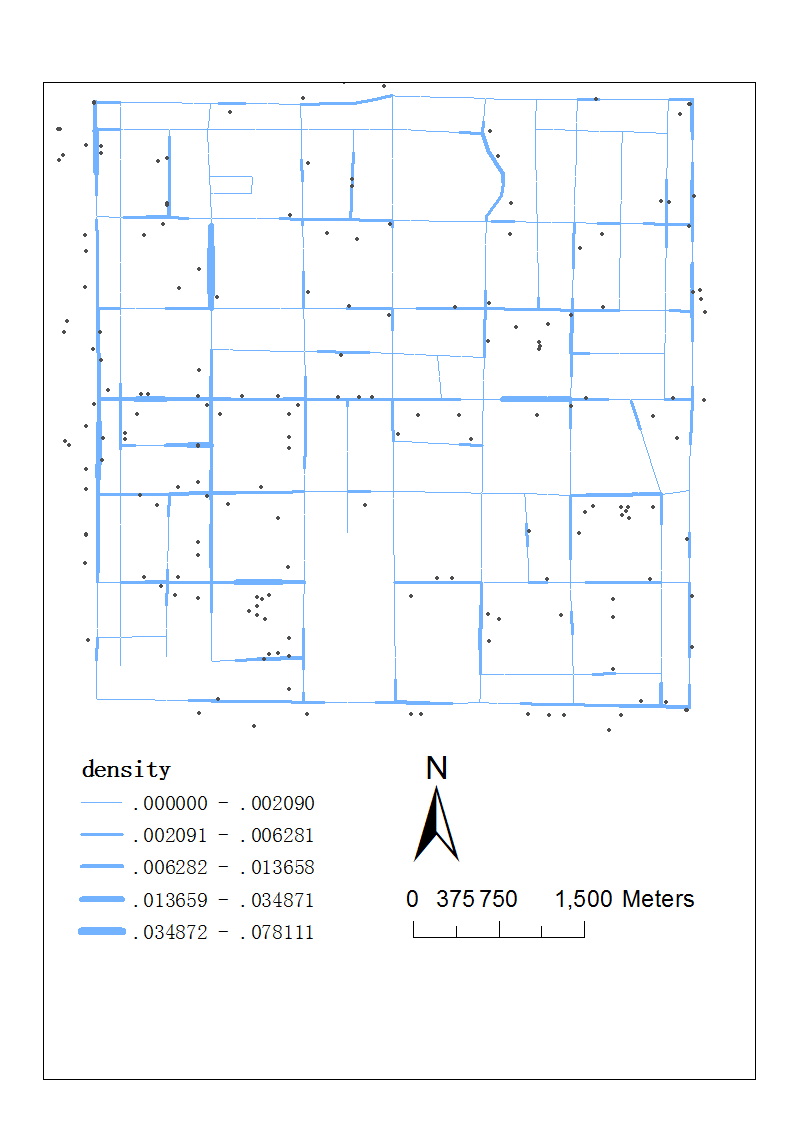


Figure 5. Kernel density estimation

## N-KDE-VD with linear normalization

The linear normalization method normalizes the input linearly to the designated interval. For additive weight:

**>>>G\_weighted, G\_origin, \_ = nv.net\_density('../data/pysal/streets.shp', '../data/pysal/crimes.shp', 200, 300, normalize\_method = "linear", edge\_weight\_mode = "additive", normalize = [-50.0,50.0])**

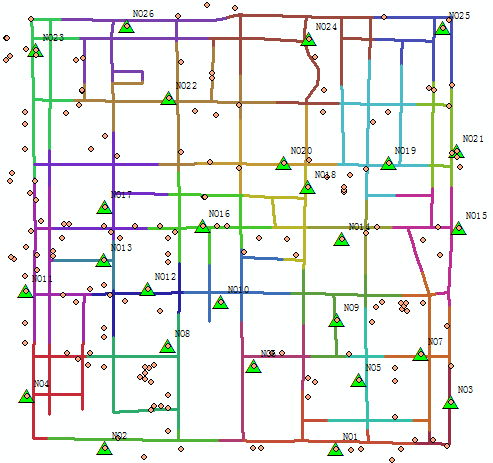
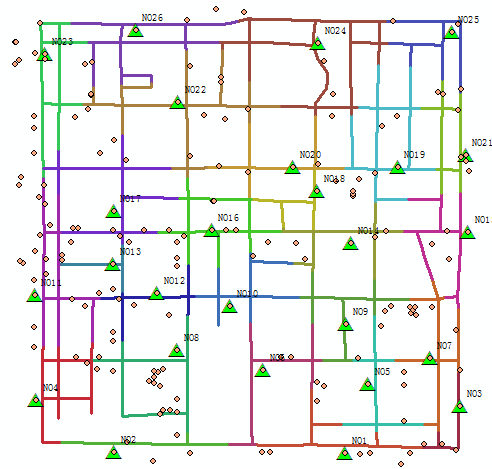
**>>>\_=nv.netvoronoi(G\_weighted, '../data/pysal/stations.shp', '../data/pysal/kde/netvoronoi\_linear\_additive\_link.shp','ID\_test', None, None,G\_origin)**

For multiplicative weight:

**>>>G\_weighted, G\_origin, \_ = nv.net\_density('../data/pysal/streets.shp', '../data/pysal/crimes.shp', 200, 300, normalize\_method = "linear", edge\_weight\_mode = "multiplicative", normalize = [0.5,1.5])**

**>>>\_=nv.netvoronoi(G\_weighted, '../data/pysal/stations.shp', '../data/pysal/kde/netvoronoi\_linear\_multiplicative\_link.shp', 'ID\_test', None, None,G\_origin)**

The results of these two cases are shown in Figure 6.

****

(a) additively weighted (b) multiplicatively weighted

Figure 6. N-KDE-VD with linear normalization

## N-KDE-VD with quantile normalization

The Quantile normalization ignores the absolute value and only looks at the relative relationships of the input values. For additive weight:

**>>>G\_weighted, G\_origin, \_ = nv.net\_density('../data/pysal/streets.shp', '../data/pysal/crimes.shp', 200, 300, normalize\_method = "quantile", edge\_weight\_mode = "additive", normalize = [-50.0,50.0])**

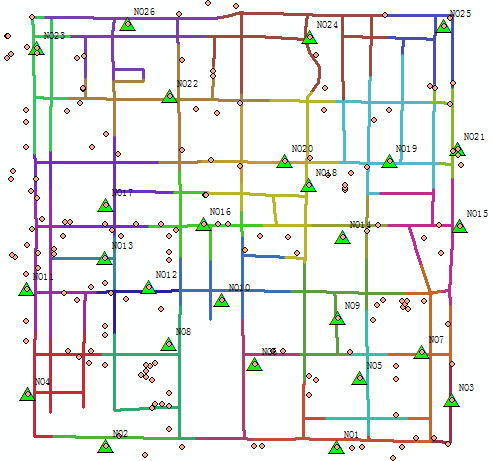
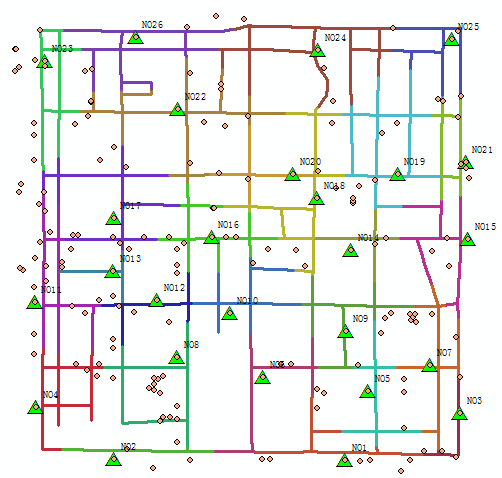
**>>>\_=nv.netvoronoi(G\_weighted, '../data/pysal/stations.shp', '../data/pysal/kde/netvoronoi\_quantile\_additive\_link.shp', 'ID\_test', None, None,G\_origin)**

For multiplicative weight:

**>>>G\_weighted, G\_origin, \_ = nv.net\_density('../data/pysal/streets.shp', '../data/pysal/crimes.shp', 200, 300, normalize\_method = "quantile", edge\_weight\_mode = "multiplicative", normalize = [0.5,1.5])**

**>>>\_=nv.netvoronoi(G\_weighted, '../data/pysal/stations.shp', '../data/pysal/kde/netvoronoi\_quantile\_multiplicative\_link.shp', 'ID\_test', None, None,G\_origin)**

The results of these two cases are shown in Figure 7.

****

(a) additively weighted (b) multiplicatively weighted

Figure 7. N-KDE-VD with quantile normalization

# N-ILINCS-VD

Similar to the idea of N-KDE-VD, the N-ILINCS-VD uses the result from network-constrained local Moran’s I (ILINCS) as an indicator of the clustering level at the link level. The difference is that there is an explicit hypothesis testing phase in local Moran’s I, thus it exclude some random patterns from the weighting process.

The local moran’s I calculation needs to be done at first. The *net\_lincs* method takes the event and the network as input and generates the transformed network. Other parameters include the *segment width, spatial weight type, distance threshold, lisa function, simulation method, number of simulations, significance level, link weighting mode (additive or multiplicative), normalizing method and interval.* Two types of normalization method are implemented: *linear* and *quantile*. Note that the *“lisa\_func”* parameter is preserved, and only “moran” is supported for now.

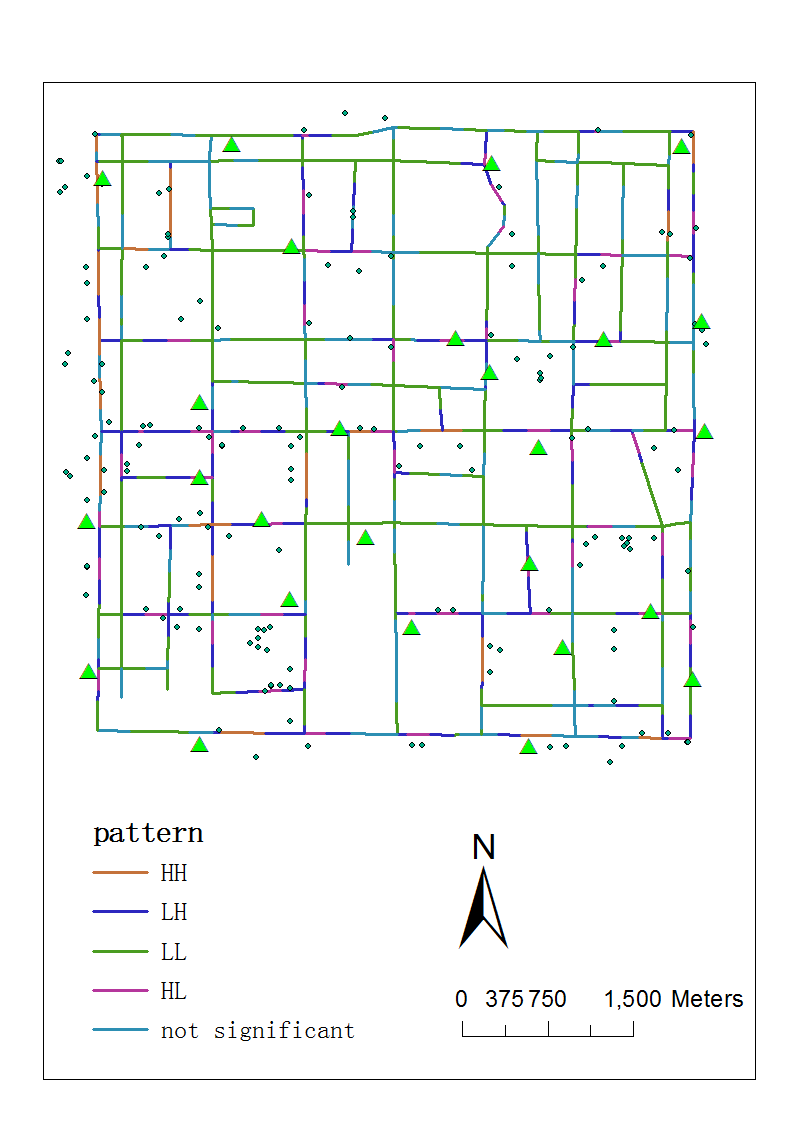
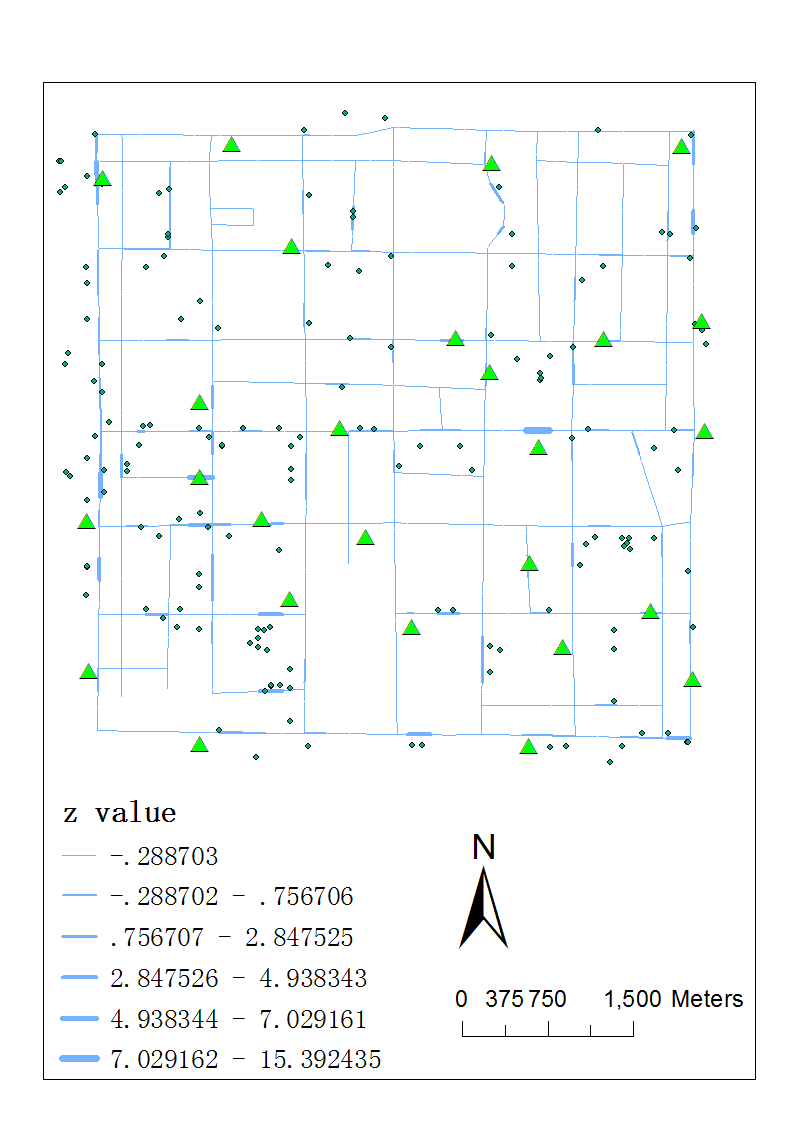
## ILINCS output

We can write the ILINCS result to a shapefile and visualize it afterwards.

**>>>\_, \_, cluster\_levels = nv.net\_lincs('../data/pysal/streets.shp', '../data/pysal/crimes.shp', 200,'Node-based')**

**>>>nv.write\_ilincs\_to\_shp(cluster\_levels, '../data/pysal/ilinc/ilinc2.shp')**

The result is written into '../data/pysal/ilinc/ilinc2.shp', the attributes include the I value, Z value, p value, the pattern (HH, HL, LH, LL). The z value distribution and the pattern classification are shown in Figure 8, we will use Z value combined with the significance level for the link weights.

****

(a) Z value (b) patterns

Figure 8. N-ILINCS-VD with linear normalization

## N-ILINCS-VD with linear normalization

The linear normalization method normalizes the input linearly to the designated interval. For additive weight:

**>>>G\_weighted, G\_origin,\_ = nv.net\_lincs('../data/pysal/streets.shp', '../data/pysal/crimes.shp', 200,'Node-based', normalize\_method = "linear", edge\_weight\_mode = "additive", normalize = [-50.0,50.0])**

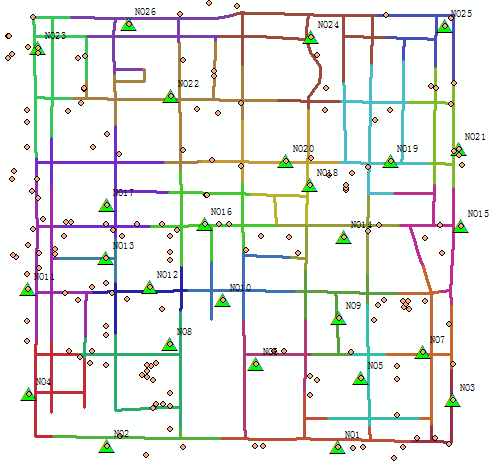
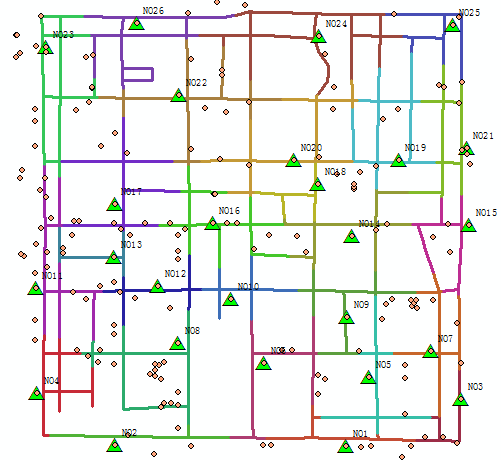
**>>>\_=nv.netvoronoi(G\_weighted, '../data/pysal/stations.shp', '../data/pysal/ilinc/netvoronoi\_linear\_additive\_link.shp', 'ID\_test', None, None, G\_origin)**

For multiplicative weight:

**>>>G\_weighted, G\_origin,\_ = nv.net\_lincs('../data/pysal/streets.shp', '../data/pysal/crimes.shp', 200,'Node-based', normalize\_method = "linear", edge\_weight\_mode = "multiplicative", normalize = [0.5,1.5])**

**>>>\_=nv.netvoronoi(G\_weighted, '../data/pysal/stations.shp', '../data/pysal/ilinc/netvoronoi\_linear\_multiplicative\_link.shp', 'ID\_test', None, None,G\_origin)**

The results of these two cases are shown in Figure 9.

** **

(a) additively weighted (b) multiplicatively weighted

Figure 9. N-ILINCS-VD with linear normalization

## N-ILINCS-VD with quantile normalization

The Quantile normalization ignores the absolute value and only looks at the relative relationships of the input values. For additive weight:

**>>>G\_weighted, G\_origin,\_ = nv.net\_lincs('../data/pysal/streets.shp', '../data/pysal/crimes.shp', 200,'Node-based', normalize\_method = "quantile", edge\_weight\_mode = "additive", normalize = [-50.0,50.0])**

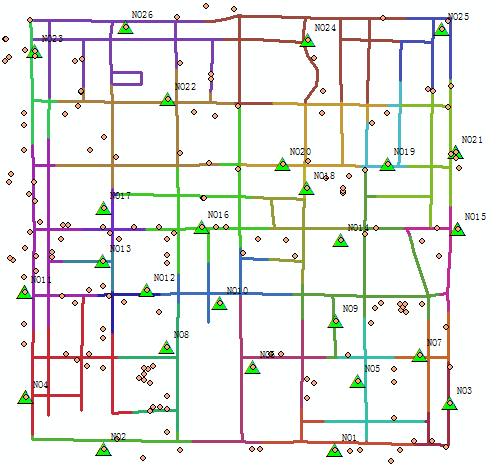
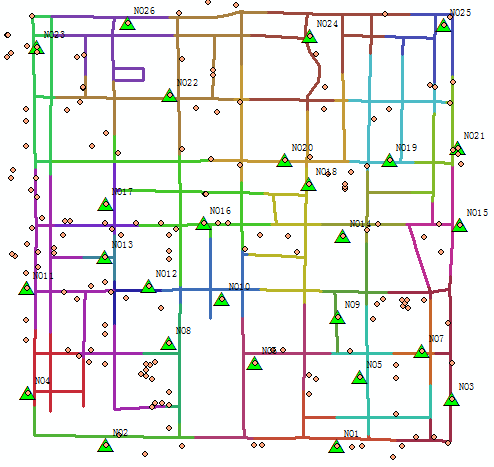
**>>>\_=nv.netvoronoi(G\_weighted, '../data/pysal/stations.shp', '../data/pysal/ilinc/netvoronoi\_quantile\_additive\_link.shp', 'ID\_test', None, None,G\_origin)**

For multiplicative weight:

**>>>G\_weighted, G\_origin,\_ = nv.net\_lincs('../data/pysal/streets.shp', '../data/pysal/crimes.shp', 200,'Node-based', normalize\_method = "quantile", edge\_weight\_mode = "multiplicative", normalize = [0.5,1.5])**

**>>>\_=nv.netvoronoi(G\_weighted, '../data/pysal/stations.shp', '../data/pysal/ilinc/netvoronoi\_quantile\_multiplicative\_link.shp', 'ID\_test', None, None,G\_origin)**

The results of these two cases are shown in Figure 10.

** **

(a) additively weighted (b) multiplicatively weighted

Figure 10. N-ILINCS-VD with quantile normalization

1. https://github.com/pysal/pysal [↑](#footnote-ref-1)