Chapter 6 - Generating morphological tessellation and measure morphometric characters

November 10, 2020

This notebook generates morphological tessellation within all tested buffers and measure their morphometric characters.

Note: *Reach* has been calculated using UNA Toolkit in ArcMap 10.6.

[4]:

```
import momepy as mm
import geopandas as gpd
import libpysal
import numpy as np
from tqdm import tqdm
import pandas as pd
```

[5]:

```
mm.__version__, gpd.__version__, libpysal.__version__, np.__version__, pd.
 ↪__version__
```

[5]:

```
('0.1.1', '0.7.0', '4.2.2', '1.18.1', '1.0.3')
```

[ ]:

```
buildings = gpd.read_file('data/zurich.gpkg', layer='buildings')

buildings = mm.preprocess(buildings, size=30, compactness=False, islands=True)

buildings['uID'] = range(len(buildings))

buildings['blg_area'] = buildings.area
buildings.to_file('data/zurich.gpkg', layer='buildings', driver='GPKG')

buildings['geometry'] = buildings.simplify(0.2)
print('simplified')

buffers = [300, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200]
for buf in buffers:
    print('Generating', buf)
    limit = mm.buffered_limit(buildings, buf)
    tessellation = mm.Tessellation(buildings, 'uID', limit).tessellation
    tessellation.to_file('data/tessellation/{0}_tessellation.shp'.format(buf))
```

[ ]:

```
def gini(vals):
    """Calculate the Gini coefficient of a numpy array."""
    # based on bottom eq:
    # http://www.statsdirect.com/help/generatedimages/equations/equation154.svg
    # from:
    # http://www.statsdirect.com/help/default.htm#nonparametric_methods/gini.htm
    # All values are treated equally, arrays must be 1d:
    vals = vals.flatten()
    if np.amin(vals) < 0:
        # Values cannot be negative:
        vals -= np.amin(vals)
    # Values cannot be 0:
    vals += 0.0000001
    # Values must be sorted:
    vals = np.sort(vals)
    # Index per array element:
    index = np.arange(1, vals.shape[0] + 1)
    # Number of array elements:
    n = vals.shape[0]
    # Gini coefficient:
    return ((np.sum((2 * index - n - 1) * vals)) / (n * np.sum(vals)))
```

[ ]:

```
def gini_fn(gdf, values, spatial_weights, unique_id):

    # define empty list for results
    results_list = []
    gdf = gdf.copy()
    print('Calculating gini...')

    for index, row in tqdm(gdf.iterrows(), total=gdf.shape[0]):
        neighbours = spatial_weights.neighbors[row[unique_id]]
        if neighbours:
            neighbours.append(row[unique_id])

            values_list = gdf.loc[gdf[unique_id].isin(neighbours)][values].values

            results_list.append(gini(values_list))
        else:
            results_list.append(0)
    series = pd.Series(results_list, index=gdf.index)

    print('Gini calculated.')
    return series
```

[ ]:

```
for buf in buffers:
    tessellation = gpd.read_file('data/tessellation/{0}_tessellation.shp'.
↪format(buf))
    tessellation['area'] = tessellation.area
    tessellation['lal'] = mm.LongestAxisLength(tessellation).series
    tessellation['circom'] = mm.CircularCompactness(tessellation).series
    tessellation['shapeix'] = mm.ShapeIndex(tessellation, 'lal', 'area').series
    tessellation['rectan'] = mm.Rectangularity(tessellation, 'area').series
    tessellation['fractal'] = mm.FractalDimension(tessellation, 'area').series
    tessellation['orient'] = mm.Orientation(tessellation).series
    distancesw = libpysal.weights.DistanceBand.from_dataframe(tessellation,␣
↪400, ids='uID')
    tessellation['freq'] = mm.Neighbors(tessellation, distancesw, 'uID').series
    tessellation['car'] = mm.AreaRatio(tessellation, buildings, 'area', mm.
↪Area(buildings).series)
    tessellation['gini_area'] = gini_fn(tessellation, 'area', distancesw, 'uID')
    tessellation['gini_car'] = gini_fn(tessellation, 'car', distancesw, 'uID')
    tessellation.to_file('data/tessellation/{0}_tessellation.shp'.format(buf))
```

[ ]:

```
cadastre = gpd.read_file('data/cadastre/Zurich_cadastre.shp')

cadastre['area'] = tessellation.area
cadastre['lal'] = mm.LongestAxisLength(cadastre).series
cadastre['circom'] = mm.CircularCompactness(cadastre).series
cadastre['shapeix'] = mm.ShapeIndex(cadastre, 'lal', 'area').series
cadastre['rectan'] = mm.Rectangularity(cadastre, 'area').series
cadastre['fractal'] = mm.FractalDimension(cadastre, 'area').series
cadastre['orient'] = mm.Orientation(cadastre).series
distancesw = libpysal.weights.DistanceBand.from_dataframe(cadastre, 400,␣
↪ids='uID')
cadastre['freq'] = mm.Neighbors(cadastre, distancesw, 'uID').series
cadastre['car'] = mm.AreaRatio(cadastre, buildings, 'area', mm.Area(buildings).
↪series)
cadastre['gini_area'] = gini_fn(cadastre, 'area', distancesw, 'uID')
cadastre['gini_car'] = gini_fn(cadastre, 'car', distancesw, 'uID')
cadastre.to_file('data/cadastre/cadastre.shp')
```