Create a trajectory Dataframe

In [ ]:

```
# In scikit-mobility, a set of trajectories is described by a TrajDataFrame,
# an extension of the pandas DataFrame that has specific columns names and data types.
# A TrajDataFrame can contain many trajectories, and each row in the TrajDataFrame represents
# a point of a trajectory, described by three mandatory fields (aka columns):

# 1. latitude (type: float);
# 2. longitude (type: float);
# 3. datetime (type: date-time).
```

In [ ]:

```
## Additionally, two optional columns can be specified:

# •uid (type: string) identifies the object associated with the point of the trajectory.
# If uid is not present, scikit-mobility assumes that the TrajDataFrame contains trajectories
# associated with a single moving object;


# •tid specifies the identifier of the trajectory to which the point belongs to.
# If tid is not present, scikit-mobility assumes that all rows in the TrajDataFrame
# associated with a uid belong to the same trajectory;
```

In [2]:
```
import skmob
# create a TrajDataFrame from a list
data_list = [[1, 39.984094, 116.319236, '2008-10-23 13:53:05'], [1, 39
.984198, 116.319322, '2008-10-23 13:53:06'], [1, 39.984224, 116.319402
, '2008-10-23 13:53:11'], [1, 39.984211, 116.319389, '2008-10-23 13:53
:16']]
tdf = skmob.TrajDataFrame(data_list, latitude=1, longitude=2, datetime
=3)
# print a portion of the TrajDataFrame
print(tdf.head())
```

```
   0        lat         lng            datetime
0  1  39.984094  116.319236 2008-10-23 13:53:05
1  1  39.984198  116.319322 2008-10-23 13:53:06
2  1  39.984224  116.319402 2008-10-23 13:53:11
3  1  39.984211  116.319389 2008-10-23 13:53:16
```

In [3]:
```
print(type(tdf))
```

```
<class 'skmob.core.trajectorydataframe.TrajDataFrame'>
```

In [4]:
```
## Now, Creating the TrajDataFrame from a pandas dataframe

import pandas as pd
# create a DataFrame from the previous list
data_df = pd.DataFrame(data_list, columns=['user', 'latitude', 'lng',
'hour'])
tdf = skmob.TrajDataFrame(data_df, latitude='latitude', datetime='hour
', user_id='user')
# print the type of the object
print(type(tdf))
```

```
<class 'skmob.core.trajectorydataframe.TrajDataFrame'>
```

In [6]:
```
>>> # print the TrajDataFrame
>>> print(tdf)
```

```
   uid        lat         lng            datetime
0    1  39.984094  116.319236 2008-10-23 13:53:05
1    1  39.984198  116.319322 2008-10-23 13:53:06
2    1  39.984224  116.319402 2008-10-23 13:53:11
3    1  39.984211  116.319389 2008-10-23 13:53:16
```

In [7]:
```
# We can also create a TrajDataFrame from a file. For example, in the following
# we create a TrajDataFrame from a portion of a GPS trajectory dataset collected in the context of
# the GeoLife project by 178 users in a period of
# over four years from April 2007 to October 2011.
```

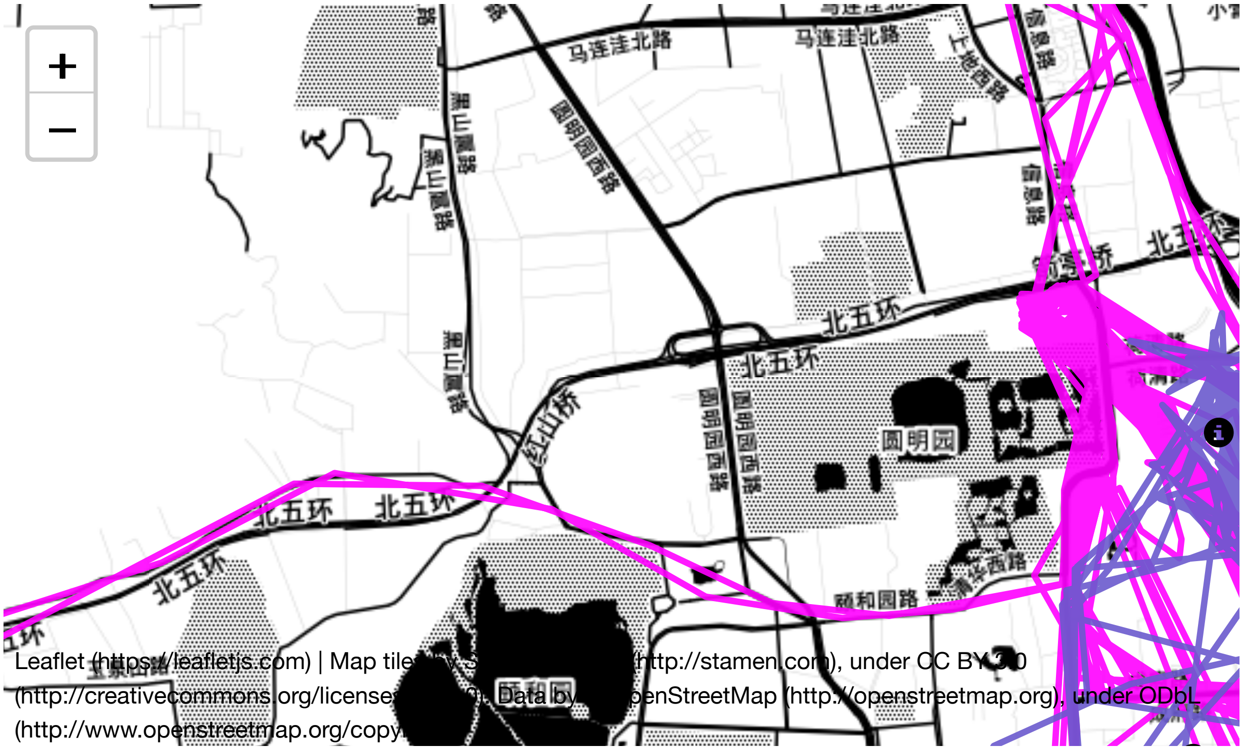
In [11]:
```
# Now, Creating a TrajDataFrame from a file

# download the file from https://raw.githubusercontent.com/scikit-mobility/scikit-mobility/master/tutorial/data/geolife_sample.txt.gz
# read the trajectory data (GeoLife, Beijing, China)
tdf = skmob.TrajDataFrame.from_file('C://Users//Snigdha.Cheekoty//Downloads//geolife_sample.txt.gz', latitude='lat', longitude='lon', user_id='user', datetime='datetime')

# print the TrajDataFrame
print(tdf)
```

```
              lat         lng            datetime  uid
0       39.984094  116.319236 2008-10-23 05:53:05    1
1       39.984198  116.319322 2008-10-23 05:53:06    1
2       39.984224  116.319402 2008-10-23 05:53:11    1
3       39.984211  116.319389 2008-10-23 05:53:16    1
4       39.984217  116.319422 2008-10-23 05:53:21    1
5       39.984710  116.319865 2008-10-23 05:53:23    1
6       39.984674  116.319810 2008-10-23 05:53:28    1
7       39.984623  116.319773 2008-10-23 05:53:33    1
8       39.984606  116.319732 2008-10-23 05:53:38    1
9       39.984555  116.319728 2008-10-23 05:53:43    1
10      39.984579  116.319769 2008-10-23 05:53:48    1
11      39.984579  116.319769 2008-10-23 05:53:51    1
12      39.984577  116.319766 2008-10-23 05:53:53    1
13      39.984611  116.319822 2008-10-23 05:53:58    1
14      39.984959  116.319969 2008-10-23 05:54:03    1
15      39.985036  116.320056 2008-10-23 05:54:04    1
16      39.984741  116.320037 2008-10-23 05:54:05    1
17      39.984620  116.320120 2008-10-23 05:54:07    1
18      39.984530  116.320242 2008-10-23 05:54:11    1
19      39.984508  116.320331 2008-10-23 05:54:16    1
20      39.984537  116.320443 2008-10-23 05:54:21    1
21      39.984529  116.320573 2008-10-23 05:54:26    1
22      39.984466  116.320683 2008-10-23 05:54:30    1
23      39.984409  116.320778 2008-10-23 05:54:34    1
24      39.984320  116.320808 2008-10-23 05:54:38    1
25      39.984252  116.320826 2008-10-23 05:54:43    1
26      39.984238  116.320844 2008-10-23 05:54:48    1
27      39.984232  116.320853 2008-10-23 05:54:53    1
28      39.984246  116.320870 2008-10-23 05:54:58    1
```

```
29      39.984266  116.320876 2008-10-23 05:55:03    1
...           ...         ...                 ...  ...
217623  40.001137  116.326452 2009-03-19 05:44:02    5
217624  40.001103  116.326519 2009-03-19 05:44:07    5
217625  40.001126  116.326607 2009-03-19 05:44:12    5
217626  40.001091  116.326667 2009-03-19 05:44:17    5
217627  40.001073  116.326735 2009-03-19 05:44:22    5
217628  40.001073  116.326821 2009-03-19 05:44:27    5
217629  40.001080  116.326905 2009-03-19 05:44:32    5
217630  40.001073  116.326977 2009-03-19 05:44:37    5
217631  40.001064  116.327037 2009-03-19 05:44:42    5
217632  40.001033  116.327062 2009-03-19 05:44:47    5
217633  40.001022  116.327054 2009-03-19 05:44:52    5
217634  40.001022  116.327054 2009-03-19 05:44:55    5
217635  40.001023  116.327058 2009-03-19 05:44:57    5
217636  40.001019  116.327071 2009-03-19 05:45:02    5
217637  40.000759  116.327088 2009-03-19 05:45:07    5
217638  40.000595  116.327066 2009-03-19 05:45:12    5
217639  40.000514  116.327017 2009-03-19 05:45:17    5
217640  40.000457  116.327019 2009-03-19 05:45:22    5
217641  40.000368  116.327072 2009-03-19 05:45:27    5
217642  40.000291  116.327100 2009-03-19 05:45:32    5
217643  40.000205  116.327173 2009-03-19 05:45:37    5
217644  40.000128  116.327171 2009-03-19 05:45:42    5
217645  40.000069  116.327179 2009-03-19 05:45:47    5
217646  40.000001  116.327219 2009-03-19 05:45:52    5
217647  39.999919  116.327211 2009-03-19 05:45:57    5
217648  39.999896  116.327290 2009-03-19 05:46:02    5
217649  39.999899  116.327352 2009-03-19 05:46:07    5
217650  39.999945  116.327394 2009-03-19 05:46:12    5
217651  40.000015  116.327433 2009-03-19 05:46:17    5
217652  39.999978  116.327460 2009-03-19 05:46:37    5

[217653 rows x 4 columns]
```

In [12]:
```
tdf.plot_trajectory(zoom=12, weight=3, opacity=0.9, tiles='Stamen Toner')
```

Out[12]:



Create Flow DataFrame

In [13]:
```
# In scikit-mobility, an origin-destination matrix is described by the FlowDataFrame structure,
# an extension of the pandas DataFrame that has specific column names and data types. A row in a FlowDataFrame represents a flow of objects between two locations, described by three mandatory columns:

# 1. origin (type: string);
# 2. destination (type: string);
# 3. flow (type: integer).
```

In [14]:
```
#  Each FlowDataFrame is associated with a spatial tessellation,
# a geopandas GeoDataFrame that contains two mandatory columns:

# 1. tile_ID (type: integer) indicates the identifier of a location;

# 2. geometry indicates the polygon (or point) that describes the geom
etric shape of the location
# on a territory (e.g., a square, a voronoi shape, the shape of a neig
hborhood)


# Note that each location identifier in the origin and destination col
umns of a FlowDataFrame
# must be present in the associated spatial tessellation.
```

In [15]:
```
# Create a spatial tessellation from a file describing counties in New
York state:

import skmob
import geopandas as gpd

# load a spatial tessellation
url_tess = 'https://raw.githubusercontent.com/scikit-mobility/scikit-m
obility/master/tutorial/data/NY_counties_2011.geojson'
tessellation = gpd.read_file(url_tess).rename(columns={'tile_id': 'til
e_ID'})

# print a portion of the spatial tessellation
print(tessellation.head())
```

```
  tile_ID  population                                           geom
etry
0   36019       81716  POLYGON ((-74.00667 44.88602, -74.02739 44.99
5...
1   36101       99145  POLYGON ((-77.09975 42.27421, -77.09966 42.27
2...
2   36107       50872  POLYGON ((-76.25015 42.29668, -76.24914 42.30
2...
3   36059     1346176  POLYGON ((-73.70766 40.72783, -73.70027 40.73
9...
4   36011       79693  POLYGON ((-76.27907 42.78587, -76.27535 42.78
0...
```

In [18]:

```
# CREATE A FLOW DATAFRAME
# .......from a spatial tesselation and a file of real flows between c
ounties in NY State

# load real flows into a FlowDataFrame
# download the file with the real fluxes from: https://raw.githubuserc
ontent.com/scikit-mobility/scikit-mobility/master/tutorial/data/NY_com
muting_flows_2011.csv
fdf = skmob.FlowDataFrame.from_file("C://Users//Snigdha.Cheekoty//Down
loads//NY_commuting_flows_2011.csv", tessellation=tessellation, tile_i
d='tile_ID',
sep=",")

# print a portion of the flows
print(fdf.head(10))
```

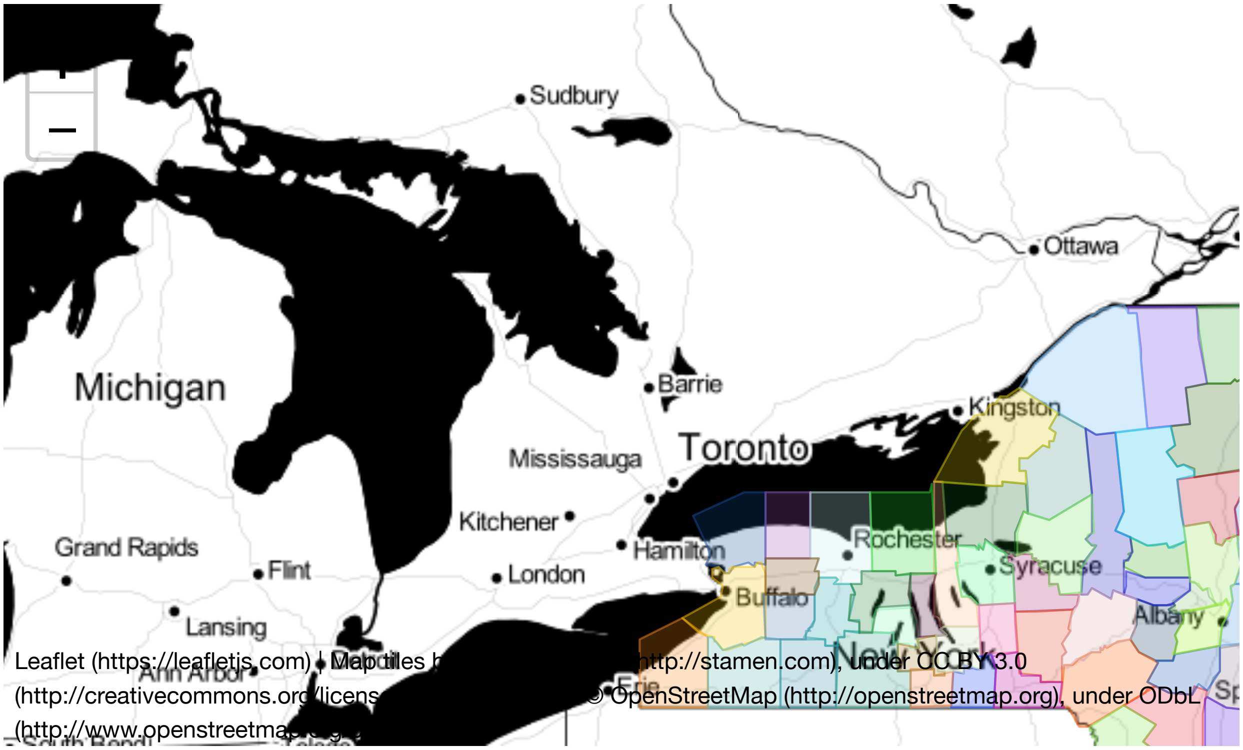
```
     flow origin destination
0  121606  36001       36001
1       5  36001       36005
2      29  36001       36007
3      11  36001       36017
4      30  36001       36019
5     728  36001       36021
6      38  36001       36023
7       6  36001       36025
8     183  36001       36027
9      31  36001       36029
```

In [19]:
```
# A FlowDataFrame can be visualized on a folium interactive map using the plot_flows function,
# which plots the flows on a geographic map as lines between the centroids of the tiles
# in the FlowDataFrame's spatial tessellation:

fdf.plot_flows(flow_color='red')
```

Out[19]:

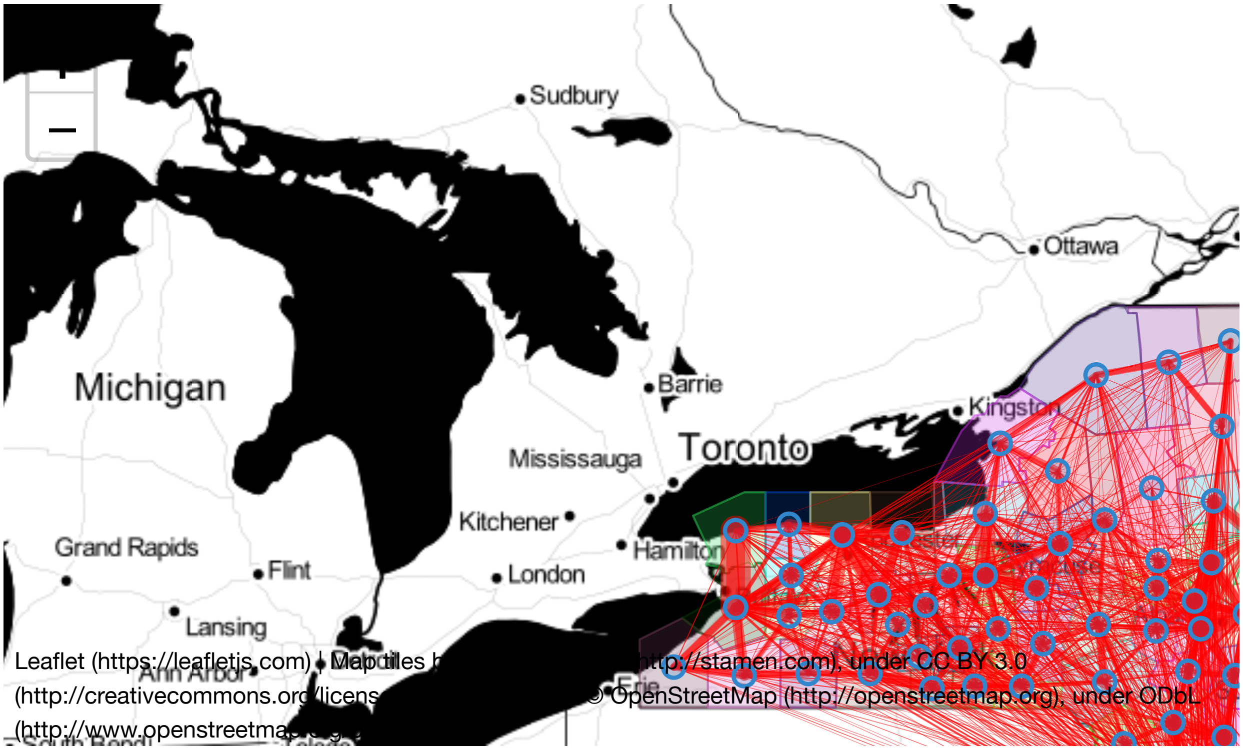
In [20]:
```
# Spacial Tessellation of FlowDataFrame can be visualized using plot_t
esselation function

fdf.plot_tessellation(popup_features=['tile_ID', 'population'])
```

Out[20]:



In [21]:
```
# Visualizing spatial tessellation and the flows together

m = fdf.plot_tessellation() # plot the tessellation
fdf.plot_flows(flow_color='red', map_f=m) # plot the flows
```

Out[21]:



In [ ]:

In [ ]:

Trajectory Preprocessing

In [22]:
```
## The preprocessing needed for mobility data analysis:

#1. Noise filtering
#2. Stop detection
#3. Stop Clustering
#4. Trajectory Compression
```

In [23]:
```
# Note that, if a TrajDataFrame contains multiple trajectories from mu
ltiple users,
# the preprocessing methods automatically apply to the single trajecto
ry and,
# when necessary, to the single moving object.

# ******************** Noise filtering ******************************
***********************
# Filter ot a point if:
#  if the speed from the previous point is higher than the parameter m
ax_speed,
# which is by default set to 500km/h

from skmob.preprocessing import filtering

# filter out all points with a speed (in km/h) from the previous point
higher than 500 km/h
ftdf = filtering.filter(tdf, max_speed_kmh=500.)
print(ftdf.parameters)
```

```
{'from_file': 'C://Users//Snigdha.Cheekoty//Downloads//geolife_sampl
e.txt.gz', 'filter': {'function': 'filter', 'max_speed_kmh': 500.0,
'include_loops': False, 'speed_kmh': 5.0, 'max_loop': 6, 'ratio_max'
: 0.25}}
```

In [24]:
```
n_deleted_points = len(tdf) - len(ftdf) # number of deleted points
print(n_deleted_points)
```

```
54
```

In [26]:
```
# ********************************* Stop detection ******************
****************************

# Some points in the trajectory can represent POINTS OF INTEREST (POI)
such as schools, bars, restuarants,
# and also user-specific loactions like home and work locations
# These POIs are also called STOPS and can be detected in different wa
ys
# Common approach is to apply spatial clustering algorithms to cluster
trajectory points by looking at theor spatial proximity


from skmob.preprocessing import detection

# compute the stops for each individual in the TrajDataFrame
# Identifying the stops where the object spent atleast certain minutes
within a certain distance
stdf = detection.stops(tdf, stop_radius_factor=0.5, minutes_for_a_stop
=20.0, spatial_radius_km=0.2, leaving_time=True)

# print the detected stops
print(stdf)
```

```
          lat         lng            datetime  uid    leaving_datet
ime
0    39.978030  116.327481 2008-10-23 06:01:37    1 2008-10-23 10:32
:53
1    40.013820  116.306532 2008-10-23 11:10:19    1 2008-10-23 23:45
:27
2    39.978419  116.326870 2008-10-24 00:21:52    1 2008-10-24 01:47
:30
3    39.981166  116.308475 2008-10-24 02:02:31    1 2008-10-24 02:30
:29
4    39.981431  116.309902 2008-10-24 02:30:29    1 2008-10-24 03:16
:35
5    39.979583  116.313643 2008-10-24 03:26:30    1 2008-10-24 03:50
:05
6    39.977984  116.326675 2008-10-24 04:08:59    1 2008-10-24 05:29
:22
7    39.982380  116.311099 2008-10-24 05:39:05    1 2008-10-24 06:09
:35
8    39.977902  116.327064 2008-10-24 06:35:30    1 2008-10-24 23:44
:05
9    39.993218  116.145509 2008-10-25 05:55:19    1 2008-10-25 06:42
:16
10   39.991001  116.194009 2008-10-25 09:32:02    1 2008-10-25 10:07
:57
11   40.013846  116.306162 2008-10-25 11:21:42    1 2008-10-25 23:40
:05
```

```
12   40.074877  116.341682 2008-10-26 00:20:37    1 2008-10-26 02:38
:57
13   39.974988  116.333675 2008-10-26 03:24:39    1 2008-10-26 03:45
:46
14   39.977658  116.384156 2008-10-26 03:53:30    1 2008-10-26 06:30
:55
15   39.978088  116.326790 2008-10-26 07:06:53    1 2008-10-26 08:12
:29
16   40.013805  116.306498 2008-10-26 08:57:51    1 2008-10-26 23:47
:00
17   39.978463  116.327248 2008-10-27 00:32:28    1 2008-10-27 04:07
:29
18   39.977639  116.326155 2008-10-27 04:51:25    1 2008-10-27 11:17
:45
19   40.013557  116.306455 2008-10-27 12:27:55    1 2008-10-27 23:30
:29
20   39.977751  116.326087 2008-10-28 00:06:29    1 2008-10-28 10:28
:05
21   39.979795  116.307687 2008-10-28 11:05:33    1 2008-10-28 13:21
:25
22   40.013846  116.306394 2008-10-28 13:40:54    1 2008-10-28 23:33
:30
23   39.977058  116.325337 2008-10-29 00:15:05    1 2008-10-29 11:05
:29
24   40.013842  116.306462 2008-10-29 11:33:57    1 2008-10-29 23:43
:45
25   39.978417  116.325712 2008-10-30 00:13:59    1 2008-10-30 04:51
:39
26   39.978134  116.326852 2008-10-30 04:51:39    1 2008-10-30 05:19
:39
27   39.975648  116.313343 2008-10-30 06:55:37    1 2008-10-30 08:08
:18
28   39.978665  116.326366 2008-10-30 08:16:05    1 2008-10-30 13:14
:29
29   40.013975  116.306071 2008-10-30 13:42:32    1 2008-10-30 23:40
:04
..         ...         ...                 ...  ...
...
361  41.107107  121.155537 2009-02-08 07:01:02    5 2009-02-22 05:07
:35
362  40.010501  116.322069 2009-02-22 05:53:40    5 2009-02-22 06:38
:00
363  40.004741  116.322371 2009-02-22 06:41:45    5 2009-02-22 07:13
:35
364  39.989976  116.333492 2009-02-22 07:40:00    5 2009-02-22 09:32
:30
365  39.991957  116.327184 2009-02-22 10:36:00    5 2009-02-22 11:27
:58
366  40.010760  116.321834 2009-02-22 11:44:33    5 2009-02-23 10:02
:13
```

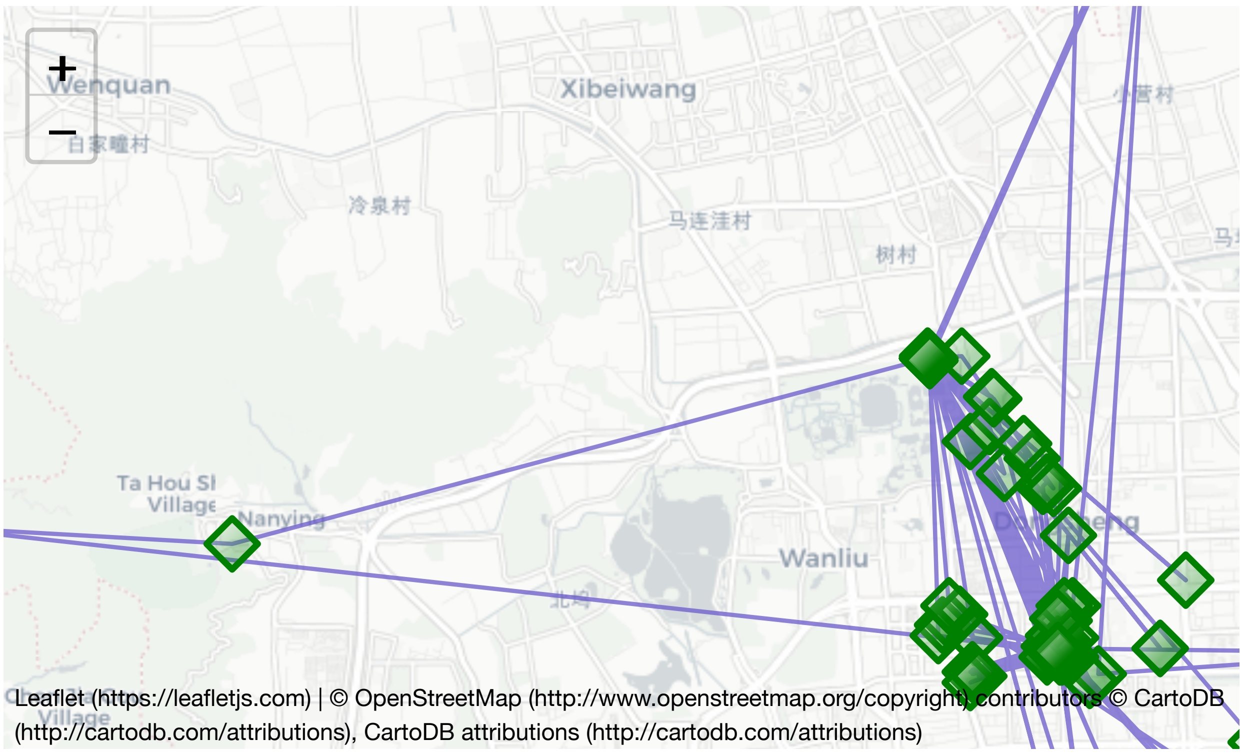
```
367  40.006708  116.320260 2009-02-23 10:02:13    5 2009-02-23 12:21
:04
368  40.010717  116.321911 2009-02-23 12:28:24    5 2009-02-24 01:03
:35
369  40.000014  116.326870 2009-02-24 01:09:30    5 2009-02-24 04:05
:05
370  40.010374  116.322523 2009-02-24 04:13:10    5 2009-02-24 04:33
:55
371  40.010991  116.321619 2009-02-24 04:35:40    5 2009-02-24 16:56
:37
372  40.010477  116.321913 2009-02-24 17:02:22    5 2009-02-25 10:06
:43
373  40.009866  116.322294 2009-02-25 10:11:13    5 2009-02-25 10:35
:48
374  40.010550  116.321783 2009-02-25 10:35:48    5 2009-02-27 15:19
:04
375  40.008988  116.316367 2009-02-27 15:42:14    5 2009-02-28 03:25
:05
376  39.999060  116.325999 2009-02-28 03:57:55    5 2009-03-04 20:27
:01
377  40.011840  116.321571 2009-03-04 20:36:11    5 2009-03-07 09:51
:00
378  39.991330  116.327065 2009-03-07 09:52:45    5 2009-03-07 13:52
:30
379  40.000753  116.326956 2009-03-07 13:52:30    5 2009-03-07 14:15
:32
380  40.011094  116.321424 2009-03-07 14:18:17    5 2009-03-11 10:10
:59
381  40.010189  116.322596 2009-03-11 10:12:34    5 2009-03-11 10:33
:19
382  40.010752  116.321777 2009-03-11 10:33:19    5 2009-03-12 09:31
:37
383  40.009584  116.320356 2009-03-12 09:50:47    5 2009-03-12 10:18
:27
384  40.010922  116.321797 2009-03-12 10:19:22    5 2009-03-12 15:47
:32
385  39.999232  116.326992 2009-03-12 16:02:52    5 2009-03-13 05:02
:34
386  40.000265  116.327024 2009-03-13 05:05:54    5 2009-03-13 13:29
:06
387  40.011017  116.322609 2009-03-13 13:32:06    5 2009-03-14 05:31
:07
388  39.990798  116.327509 2009-03-14 05:48:22    5 2009-03-14 06:36
:12
389  39.990123  116.333491 2009-03-14 06:43:27    5 2009-03-19 04:35
:37
390  40.003332  116.318045 2009-03-19 05:02:47    5 2009-03-19 05:33
:52

[391 rows x 5 columns]
```

In [27]:
```
>>> print('Points of the original trajectory:\t%s'%len(tdf))
>>> print('Points of stops:\t\t\t%s'%len(stdf))
```

```
Points of the original trajectory:      217653
Points of stops:                        391
```

In [28]:
```
## A new column leaving_datetime is added to the TrajDataFrame
## in order to indicate the time when the user left the stop location.
## We can then visualize the detected stops using the plot_stops funct
ion:

m = stdf.plot_trajectory(max_users=1, start_end_markers=False)
stdf.plot_stops(max_users=1, map_f=m)
```

Out[28]:



In [29]:
```
tdf.head()
```

Out[29]:

	lat	lng	datetime	uid
0	39.984094	116.319236	2008-10-23 05:53:05	1
1	39.984198	116.319322	2008-10-23 05:53:06	1
2	39.984224	116.319402	2008-10-23 05:53:11	1
3	39.984211	116.319389	2008-10-23 05:53:16	1
4	39.984217	116.319422	2008-10-23 05:53:21	1

In [30]:
```
# ******************************** Trajectory Compression ************
**************************

# The goal of trajectory compression is to reduce the number of trajec
tory points while
# preserving the structure of the trajectory.
# For instance, to merge all the points that are closer than 0.2km fro
m each other,
# we can use the following code:

from skmob.preprocessing import compression

# compress the trajectory using a spatial radius of 0.2 km
ctdf = compression.compress(tdf, spatial_radius_km=0.2)

# print the difference in points between original and filtered TrajDat
aFrame
print('Points of the original trajectory:\t%s'%len(tdf))
print('Points of the compressed trajectory:\t%s'%len(ctdf))
```

```
Points of the original trajectory:      217653
Points of the compressed trajectory:    6281
```

Mobility measures

In [32]:
```
# Patterns of human mobility can be captured at individual and collect
ive levels
# We can capture mobilit patterns of individual object or a group as a
whole
# SCIKIT-MOBILITY:  provides a wide set of mobility measures, each imp
lemented as a function that takes in
# input a TrajDataFrame and outputs a pandas DataFrame

# Let's compute the radius of gyration, the jump lengths and the home
locations of a TrajDataFrame

from skmob.measures.individual import jump_lengths, radius_of_gyration
, home_location

# load a TrajDataFrame from an URL
url = "https://snap.stanford.edu/data/loc-brightkite_totalCheckins.txt
.gz"
df = pd.read_csv(url, sep='\t', header=0, nrows=100000,
     names=['user', 'check-in_time', 'latitude', 'longitude', 'locatio
n id'])
tdf = skmob.TrajDataFrame(df, latitude='latitude', longitude='longitud
e', datetime='check-in_time', user_id='user')
df.head(10)
```

Out[32]:

	user	check-in_time	latitude	longitude	location id
0	0	2010-10-16T06:02:04Z	39.891383	-105.070814	7a0f88982aa015062b95e3b4843f9ca2
1	0	2010-10-16T03:48:54Z	39.891077	-105.068532	dd7cd3d264c2d063832db506fba8bf79
2	0	2010-10-14T18:25:51Z	39.750469	-104.999073	9848afcc62e500a01cf6fbf24b797732f8963683
3	0	2010-10-14T00:21:47Z	39.752713	-104.996337	2ef143e12038c870038df53e0478cefc
4	0	2010-10-13T23:31:51Z	39.752508	-104.996637	424eb3dd143292f9e013efa00486c907
5	0	2010-10-13T20:05:43Z	39.751300	-105.000121	d268093afe06bd7d37d91c4d436e0c40d217b20a
6	0	2010-10-13T16:41:35Z	39.758974	-105.010853	6f5b96170b7744af3c7577fa35ed0b8f
7	0	2010-10-13T03:57:23Z	39.827022	-105.143191	f6f52a75fd80e27e3770cd3a87054f27
8	0	2010-10-12T19:56:49Z	39.749934	-105.000017	b3d356765cc8a4aa7ac5cd18caafd393
9	0	2010-10-11T02:51:09Z	39.891077	-105.068532	6f3a2db56d4fa788f72def616f79b7a4

In [33]:
```
df.tail(10)
```

Out[33]:

	user	check-in_time	latitude	longitude	location id
99990	163	2009-02-21T00:38:33Z	37.268832	-121.975513	ee822ba4a22411dd8d55c3af1d87b00b
99991	163	2009-02-20T23:40:29Z	37.368830	-122.036350	ee8503d8a22411ddbf1c8b0502a6e649
99992	163	2009-02-20T22:41:55Z	37.441883	-122.143019	ee7e0ceaa22411dda99e93c64fce5520
99993	163	2009-02-20T08:51:28Z	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99994	163	2009-02-20T04:45:11Z	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99995	163	2009-02-19T08:10:06Z	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99996	163	2009-02-19T01:30:03Z	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99997	163	2009-02-18T07:00:47Z	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99998	163	2009-02-18T01:55:06Z	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99999	163	2009-02-16T05:03:03Z	37.368830	-122.036350	ee8503d8a22411ddbf1c8b0502a6e649

In [34]:
```
tdf.head(10)
```

Out[34]:

	uid	datetime	lat	lng	location id
0	0	2010-10-16 06:02:04+00:00	39.891383	-105.070814	7a0f88982aa015062b95e3b4843f9ca2
1	0	2010-10-16 03:48:54+00:00	39.891077	-105.068532	dd7cd3d264c2d063832db506fba8bf79
2	0	2010-10-14 18:25:51+00:00	39.750469	-104.999073	9848afcc62e500a01cf6fbf24b797732f8963683
3	0	2010-10-14 00:21:47+00:00	39.752713	-104.996337	2ef143e12038c870038df53e0478cefc
4	0	2010-10-13 23:31:51+00:00	39.752508	-104.996637	424eb3dd143292f9e013efa00486c907
5	0	2010-10-13 20:05:43+00:00	39.751300	-105.000121	d268093afe06bd7d37d91c4d436e0c40d217b20a
6	0	2010-10-13 16:41:35+00:00	39.758974	-105.010853	6f5b96170b7744af3c7577fa35ed0b8f
7	0	2010-10-13 03:57:23+00:00	39.827022	-105.143191	f6f52a75fd80e27e3770cd3a87054f27
8	0	2010-10-12 19:56:49+00:00	39.749934	-105.000017	b3d356765cc8a4aa7ac5cd18caafd393
9	0	2010-10-11 02:51:09+00:00	39.891077	-105.068532	6f3a2db56d4fa788f72def616f79b7a4

In [35]:
```
tdf.tail(10)
```

Out[35]:

	uid	datetime	lat	lng	location id
99990	163	2009-02-21 00:38:33+00:00	37.268832	-121.975513	ee822ba4a22411dd8d55c3af1d87b00b
99991	163	2009-02-20 23:40:29+00:00	37.368830	-122.036350	ee8503d8a22411ddbf1c8b0502a6e649
99992	163	2009-02-20 22:41:55+00:00	37.441883	-122.143019	ee7e0ceaa22411dda99e93c64fce5520
99993	163	2009-02-20 08:51:28+00:00	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99994	163	2009-02-20 04:45:11+00:00	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99995	163	2009-02-19 08:10:06+00:00	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99996	163	2009-02-19 01:30:03+00:00	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99997	163	2009-02-18 07:00:47+00:00	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99998	163	2009-02-18 01:55:06+00:00	37.339386	-121.894955	ee81fddca22411dd93aeb7dc12ab591c
99999	163	2009-02-16 05:03:03+00:00	37.368830	-122.036350	ee8503d8a22411ddbf1c8b0502a6e649

In [36]:
```
# compute the radius of gyration for each individual
rg_df = radius_of_gyration(tdf)
print(rg_df)
```

```
100%|██████████████████████████████████████████████████
██████████| 162/162 [00:01<00:00, 141.25it/s]

     uid  radius_of_gyration
0      0         1564.436792
1      1         2467.773523
2      2         1439.649774
3      3         1752.604191
4      4         5380.503250
5      5         2168.447820
6      6          954.818786
7      7         1896.439626
8      8         1979.975534
9      9         1206.270685
10    10         2170.567972
11    11         1080.221899
12    12         2140.776375
```

```
13    13         483.569574
14    14        3690.204679
15    15        1336.129544
16    16        1410.953458
17    17         563.332901
18    18        1355.387869
19    19        1417.638379
20    20        2521.260289
21    21        3798.013764
22    22        6318.594070
23    23        1727.109422
24    24        1351.700076
25    25         461.374109
26    26        3578.238123
27    27         858.089717
28    28        3232.511849
29    29         687.748882
..   ...               ...
132  133         324.319477
133  134         979.816518
134  135        1391.873563
135  136           0.000000
136  137        1203.118669
137  138         257.723886
138  139          27.143599
139  140        1138.570232
140  142           0.113260
141  143        1639.569893
142  144        1481.182800
143  145         287.762378
144  146        2381.516323
145  147        2398.486629
146  148        1940.848918
147  149        1530.046526
148  150         855.245886
149  151          17.579672
150  152        1657.495313
151  153        1775.450512
152  154         613.192852
153  155          39.515486
154  156        1389.746802
155  157          28.216690
156  158        1533.171635
157  159        4539.000785
158  160         530.445870
159  161        8117.341229
160  162        3012.589268
161  163        1103.993327

[162 rows x 2 columns]
```

In [37]:
```
# compute the jump lengths for each individual
jl_df = jump_lengths(tdf.sort_values(by='datetime'))
print(jl_df.head())
```

```
100%|██████████████████████████████████████████████████
██████████| 162/162 [00:01<00:00, 120.20it/s]

   uid                                       jump_lengths
0    0  [19.640467328877936, 0.0, 0.0, 1.7434311010381...
1    1  [6.505330424378251, 46.75436600375988, 53.9284...
2    2  [0.0, 0.0, 0.0, 0.0, 3.6410097195943507, 0.0, ...
3    3  [3861.2706300798827, 4.061631313492122, 5.9163...
4    4  [15511.92758595804, 0.0, 15511.92758595804, 1....
```

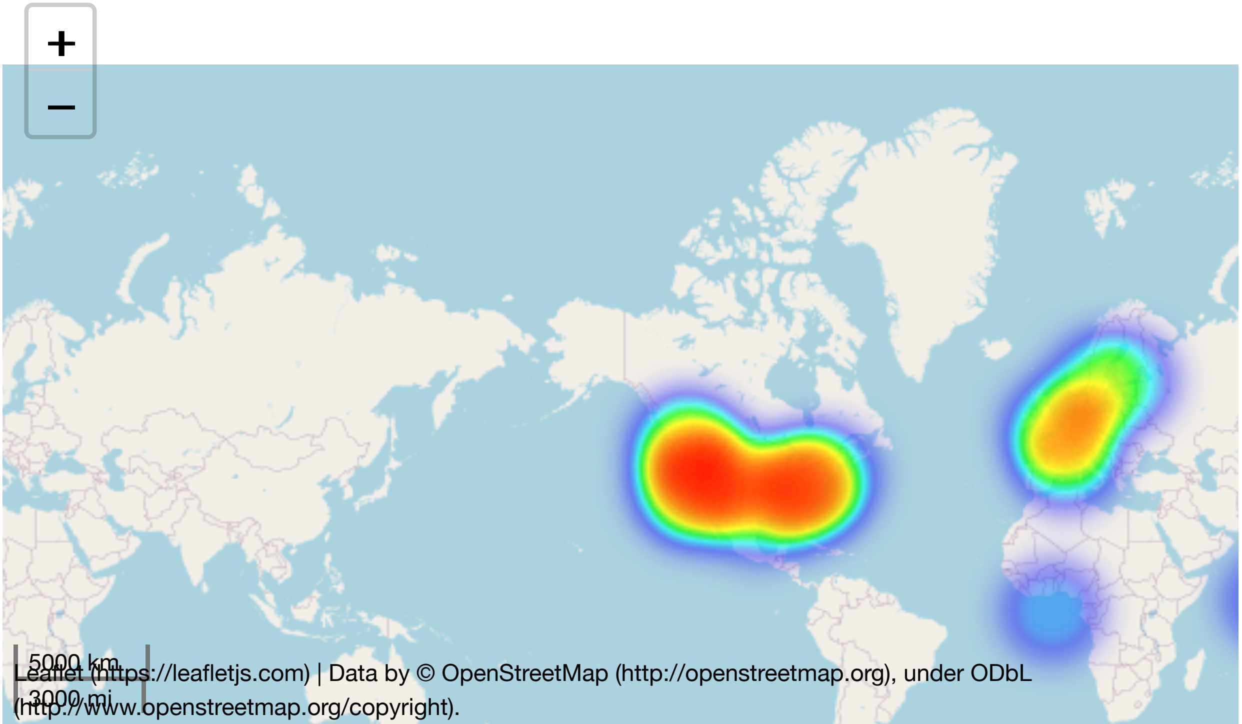
In [38]:
```
# compute the home location for each individual
hl_df = home_location(tdf)
print(hl_df.head())
```

```
100%|██████████████████████████████████████████████████
██████████| 162/162 [00:01<00:00, 158.31it/s]

   uid        lat         lng
0    0  39.891077 -105.068532
1    1  37.630490 -122.411084
2    2  39.739154 -104.984703
3    3  37.748170 -122.459192
4    4  60.180171   24.949728
```

In [39]:

```
# now let's visualize a cloropleth map of the home locations
import folium
from folium.plugins import HeatMap
m = folium.Map(tiles = 'openstreetmap', zoom_start=12, control_scale=True)
HeatMap(hl_df[['lat', 'lng']].values).add_to(m)
m
```

Out[39]:



In [40]:

```
## Collective generative models

# Collective generative models estimate spatial flows between a set of discrete locations.
# Examples of spatial flows estimated with collective generative models include
# 1. commuting trips between neighborhoods,
# 2. migration flows between municipalities,
# 3. freight shipments between states,
# 4. and phone calls between regions.
```

In [41]:
```
# Collective generative model takes in input a spatial tesselation and
geopandas dataframe
# This spatial tesselation file should contain two columns: geometry a
nd relevance

# These columns are used to compute two variables:
#    1. the distance between the tiles
#    2. the importance (aka, "attractiveness of each tile")

# A collective dataframe produces a Flow dataframe that contains gener
ated flows and spatial tesselation
```

In [42]:
```
# The collective generative algorithms that we are going to use:
# 1. Gravity model
# 2. Radiation model
```

Gravity Model

In [43]:
```
# The gravity model has two main methods
# 1. fit method: caliberates model parameters using a flow dataframe
# 2. generate method: which generates flows on given spatial tessellat
ion
```

In [46]:
```
from skmob.utils import utils, constants
import geopandas as gpd
from skmob.models import Gravity
import numpy as np

# load a spatial tessellation
url_tess = 'https://raw.githubusercontent.com/scikit-mobility/scikit-mobility/master/tutorial/data/NY_counties_2011.geojson'
tessellation = gpd.read_file(url_tess).rename(columns={'tile_id': 'tile_ID'})
# download the file with the real fluxes from: https://raw.githubusercontent.com/scikit-mobility/scikit-mobility/master/tutorial/data/NY_commuting_flows_2011.csv
fdf = skmob.FlowDataFrame.from_file("C://Users//Snigdha.Cheekoty//Downloads//NY_commuting_flows_2011.csv", tessellation=tessellation, tile_id='tile_ID', sep=",")

# compute the total outflows from each location of the tessellation (excluding self loops)
tot_outflows = fdf[fdf['origin'] != fdf['destination']].groupby(by='origin', axis=0)['flow'].sum().fillna(0).values
tessellation[constants.TOT_OUTFLOW] = tot_outflows
```

In [47]:
```
# Instantiate a gravity model object and generate synthetic flows

# instantiate a singly constrained Gravity model
gravity_singly = Gravity(gravity_type='singly constrained')
print(gravity_singly)
```

```
Gravity(name="Gravity model", deterrence_func_type="power_law", deterrence_func_args=[-2.0], origin_exp=1.0, destination_exp=1.0, gravity_type="singly constrained")
```

In [48]:
```python
# start the generation of the synthetic flows
np.random.seed(0)
synth_fdf = gravity_singly.generate(tessellation, tile_id_column='tile_ID', tot_outflows_column='tot_outflow', relevance_column= 'population', out_format='flows')

# print a portion of the synthetic flows
print(synth_fdf.head())
```

```
100%|██████████████████████████████████████████████████| 62/62 [00:00<00:00, 5651.23it/s]
C:\scikit_mobility\scikit_mobility\skmob\models\gravity.py:43: RuntimeWarning: divide by zero encountered in power
  return np.power(x, exponent)

  origin destination  flow
0  36019       36101   109
1  36019       36107    52
2  36019       36059  1105
3  36019       36011   152
4  36019       36123    34
```

In [49]:
```python
# instantiate a Gravity object (with default parameters)
gravity_singly_fitted = Gravity(gravity_type='singly constrained')
print(gravity_singly_fitted)
```

```
Gravity(name="Gravity model", deterrence_func_type="power_law", deterrence_func_args=[-2.0], origin_exp=1.0, destination_exp=1.0, gravity_type="singly constrained")
```

In [52]:
```python
# fit the parameters of the Gravity from the FlowDataFrame
gravity_singly_fitted.fit(fdf, relevance_column='population')
print(gravity_singly_fitted)
```

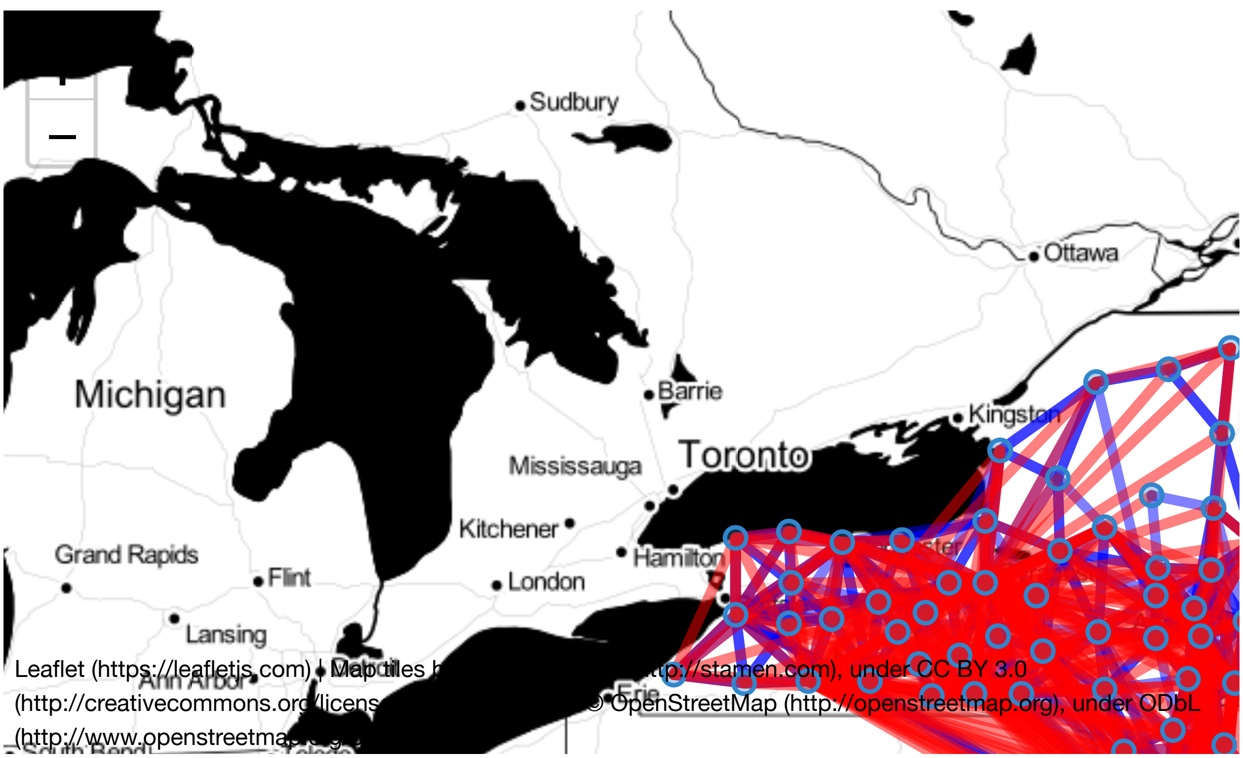
```
Gravity(name="Gravity model", deterrence_func_type="power_law", deterrence_func_args=[-1.9947152031914204], origin_exp=1.0, destination_exp=0.6471759552223136, gravity_type="singly constrained")
```

In [53]:
```
# generate the synthetics flows
np.random.seed(0)
synth_fdf_fitted = gravity_singly_fitted.generate(tessellation, tile_i
d_column='tile_ID', tot_outflows_column='tot_outflow',
                            relevance_column= 'population', out_format
='flows')

# print a portion of the synthetic flows
print(synth_fdf_fitted.head())
```

```
100%|██████████████████████████████████████████████████████████████
██████████████| 62/62 [00:00<00:00, 3657.07it/s]
C:\scikit_mobility\scikit_mobility\skmob\models\gravity.py:43: Runti
meWarning: divide by zero encountered in power
  return np.power(x, exponent)

  origin destination  flow
0  36019       36101   142
1  36019       36107   101
2  36019       36059   578
3  36019       36011   213
4  36019       36123    97
```

In [54]:
```
m = fdf.plot_flows(min_flow=100, flow_exp=0.01, flow_color='blue')
synth_fdf_fitted.plot_flows(min_flow=1000, flow_exp=0.01, map_f=m)
```

Out[54]:



Radiation Model

In [55]:

```python
# The Radiation model is parameter-free and has only one method: generate.
# Given a spatial tessellation,
# the synthetic flows can be generated using the Radiation class as follows:

from skmob.models import Radiation
# instantiate a Radiation object
radiation = Radiation()
# start the simulation
np.random.seed(0)
rad_flows = radiation.generate(tessellation,tile_id_column='tile_ID',tot_outflows_column='tot_outflow',
                        relevance_column='population', out_format='flows_sample')

# print a portion of the synthetic flows
print(rad_flows.head())
```

```
100%|██████████| 62/62 [00:00<00:00, 807.00it/s]

  origin destination   flow
0  36019       36033  11648
1  36019       36031   4232
2  36019       36089   5598
3  36019       36113   1596
4  36019       36041    117
```

In [ ]:

In [ ]: