Consiglio Nazionale
delle Ricerche

UNIVERSITÀ DI PISA

Projects

Geospatial Analytics

2022/2023

General remarks

- The project should be delivered two weeks before the date of the "appello" chosen by the student.

- The project should be delivered as a .zip folder through **this form**. For python code, both .py (scripts) and .ipynb (notebooks) files are allowed.
  - for notebooks, remember to clear all outputs before uploading it on the form (this will save memory space in the zip folder).

- Unless it is strictly needed, do not upload big datasets but provide the link where to download them or, even better, make the code download the dataset directly from a public URL. You can eventually upload small files (e.g., small json, geojson, or shapefile files).
  - For experiments on SUMO, send also the road networks, the traffic demands, and the configuration files.

- The code in the main notebook should run correctly without any modification from our side.

The projects will be evaluated based on their correctness, level of detail and depth, the elegance of coding, and creativity.

In case of questions about the projects, please write an email addressed to:

- luca.pappalardo@isti.cnr.it
- mirco.nanni@isti.cnr.it
- giuliano.cornacchia@phd.unipi.it
- giovanni.mauro@phd.unipi.it

Project 1: On the error between GPS traces and mobile phone records

In this project, the student investigates the difference between GPS traces and mobile phone records (MPR). This should be done by developing Python code that develops the following steps:

1. select at least four different datasets of GPS trajectories (e.g., the Geolife dataset and/or the Milano dataset);

2. construct a Voronoi tessellation based on the position of the phone towers on the geographic area the selected dataset(s) refer to. To get the position of mobile phone towers, you can use for example the OpenCellID dataset, but other, more creative, choices are welcome;

3. perform a mapping (spatial join) of each GPS trajectory T to the constructed Voronoi tessellation and create a modified trajectory T' where each point $p \in T$ is substituted by the coordinates of the centroid of the tile p falls within.

4. “Sparsify” T' by removing records at random so that the inter-time between consecutive records is on average x minutes (e.g., $x = 5$, try different x);

5. Implement some trajectory similarity metrics (see, e.g., this paper) and compute the similarity between each pair T, T' based on these metrics. Study the distribution of similarities (i.e., shape and aggregated statistics such as average, standard deviation, etc.).

6. Make the most appropriate visualizations to show the difference between the two types of trajectories in all the datasets. Study also the impact of parameter x (how the similarities and their distribution change with x).

7. Discuss and comment on the top-k and bottom-k users (e.g., k=5) users based on the (average) similarity between T and T’.

NOTE: for statistical consistency, step 4 should be repeated a number k of times ($k \geq 10$) every time with a different seed, e.g., with different random removal of records. Then, for each pair T, T', take the average similarity and study and proceed with the points 5, 6, 7.

Project 2: Implementation and testing of the Voronoi tessellation

The student should modify the **`VoronoiTessellationTiler`** class of the scikit-mobility library in order to make a function that creates a Voronoi tessellation together with its geometric shapes.

In particular, the `skmob.tilers.tiler.get` function should be able to create a Voronoi tessellation as a `GeoDataFrame` through the following signature:

```
skmob.tilers.tiler.get(“voronoi”, points, base_shape)
```

where:

- `“voronoi”` is a string;
- `points` is a numpy array or a list of tuples containing latitude and longitude pairs;
- `base_shape` is a Shapely polygon indicating the area the points fall within.

The student should also test the provided code using the `assert` statement and `pytest` (see the examples of testing of the skmob.tessellation module).

The developed code should be delivered as a `test_tilers.py` file, which is a modification of the corresponding file on the scikit-mobility repository. Provide also a subfolder named `examples` in which you show (e.g. using a notebook) how the developed function works on some real-world datasets and compare the Voronoi tessellation with the squared and the h3 tessellation already provided in scikit-mobility.

Example: create a Voronoi tessellation based on the OpenCellID dataset (or other datasets that you find on the web) and show some possible use of it (unleash your creativity!).

Project 3: Enrichment of spatial tessellations with Points Of Interest

The student should develop Python code to enrich a scikit-mobility spatial tessellation (either squared, voronoi, h3, or based on official division in areas such as neighborhoods), i.e., a `GeoDataFrame`, with information about Points Of Interest (POIs) downloaded from OSMnx.
In particular, the student will develop a function `enrich(gdf, tags, *)`, where:

- `gdf` (`GeoDataFrame`) represents the tessellation;
- `tags` (dict) – Dict of tags used for finding objects in the selected area. The tags should be the same as those used in the OSMnx functions (see OSMNx documentation).
  Examples,
  - `tags = {‘building’: True}` returns all building footprints.
  - `tags = {‘amenity’:True, ‘landuse’:[‘retail’,’commercial’], ‘highway’:’bus_stop’}` would return all amenities, `landuse=retail`, `landuse=commercial`, and `highway=bus_stop`.
- `*` indicates all the other arguments `plot_gdf` already has in scikit-mobility.

The function should return a `GeoDataFrame` with a `tile_ID` column indicating the tessellation’s tile where the POIs falls within, a `geometry` column describing the geometric shape of the POI (i.e., a Shapely object), and other columns describing the POI based on the output from OSMnx.

Then, the student should modify the scikit-mobility’s `plot_gdf` function so that it can take in input this new `GeoDataFrame` as well, and plot the tessellations together with the POIs’s geometries within them. Add to `plot_gdf` a parameter `max_pois` with the max number of POIs to show for each tile (you choose whether these POIs are selected randomly or based on other criteria).

Show how the `enrich` and the new `plot_gdf` functions work, developing some practical cases in at least four different cities, each for squared, voronoi, h3, and based on official division in areas (e.g., neighborhoods or census cells).

Example: show that these two functions (separately or in conjunction) may help understand the different POIs composition of each neighborhood or zone in the city.

Project 4: Implementation and testing of trajectory split based on stop detection

The student should modify scikit-mobility's stay_locations function, which takes in input a TrajDataFrame, tdf, and returns a new TrajDataFrame that splits each individual's global trajectory into sub-trajectories based on stop-detection.

An individual's global trajectory $T^{(u)}$ consists of all the points of that individual u in the TrajDataFrame (i.e., all rows in tdf with uid=u). An individual's sub-trajectory $T_i^{(u)}$ indicates all points between two consecutive stops detected by stay_locations.

In practice, the new stay_locations function should return a TrajDataFrame containing all the rows of the original tdf but with an additional column tid, which contains the identifier of each sub-trajectory of an individual. tid must have an integer value between 1 to k, where 1 is the first sub-trajectory in chronological order and k is the number of stops detected.

The signature of the new function should be as follows:

```
stay_locations(tdf, sub_traj=True, stop_radius_factor=0.5,
minutes_for_a_stop=20.0, spatial_radius_km=0.2,
leaving_time=True, no_data_for_minutes=1e12,
min_speed_kmh=None, inplace=False)
```

where:

- sub_traj=True indicates that the output should be the TrajDataFrame as defined above, while sub_traj=False indicates that the output should be the same as the current version of stay_locations.
- inplace=True indicates that the original tdf should be overwritten with the new one (i.e., the one containing the new tid column); inplace=False indicates that a new TrajDataFrame should be returned. inplace is used when sub_traj=True only, otherwise it is discarded.
- all the other arguments (those not in bold) are the same as the current stay_locations function.

NOTE: The current version of `stay_locations` returns a stop `TrajDataFrame`, which contains only the stops detected by the function based on the specified arguments.

The student should also test the provided code using the `assert` statement and `pytest` (see the examples of testing of the skmob.preprocessing module).

The developed code should be delivered as a `detection.py` file, which is a modification of the corresponding file on the scikit-mobility repository. Provide also a subfolder named `examples` in which you show how the developed function works on at least four real-world datasets.

Project 5: Systematic comparison of EPR models

The student should develop a systematic comparison of Exploration and Preferential Return (EPR) models using scikit-mobility. In particular, the student should compare the following models provided in the library:

- SpatialEPR
  - see also Human Mobility Modelling: Exploration and Preferential Return Meet the Gravity Model
- DensityEPR
  - see also Human Mobility Modelling: Exploration and Preferential Return Meet the Gravity Model
- DITRAS
  - see also Data-driven generation of spatio-temporal routines in human mobility

The student should run each model for 5,000 agents for a period of two weeks, on at least four different geographic areas (e.g., cities, regions), and using squared, hexagonal (h3), voronoi, and an official division in areas such as neighborhoods or census cells. For the Voronoi tessellation, the student can select the dataset they prefer (e.g., OpenCellID).

For example, for the SpatialEPR model, the student should perform the following list of experiments (similarly for the other models):

Model	Geographic Area	Tessellation
SpatialEPR	area1	squared
SpatialEPR	area1	h3
SpatialEPR	area1	voronoi
SpatialEPR	area1	official division
…	…	…
SpatialEPR	area2	squared
SpatialEPR	area2	h3
…	…	…
SpatialEPR	area3	squared

SpatialEPR	area3	h3

The models' realism (i.e., the realism of the generated trajectories) should be compared in terms of:

1. distribution of jump length (M_1);
2. distribution of the radius of gyration (M_2);
3. distribution of uncorrelated entropy (M_3);
4. number of distinct visited locations (M_4);
5. number of visits per location (M_5);
6. location frequency (M_6).

Use the appropriate scikit-mobility measures to compute the above metrics.

For given an area (e.g., city, region), compute the distributions $M_1, \ldots, M_6$ across the agents for each of the three models (SpatialEPR, DensityEPR, DITRAS). Given a distribution M_i ($i \in \{1, \ldots, 6\}$), compute a proper binning for the relative distribution and compare all models' distributions pair by pair using the Root Mean Squared Error (RMSE). See this paper (Section Results, Model comparison and validation).

NOTE: some of the models (e.g., DensityEPR, DITRAS) need information about location relevance, so define a proper relevance measure and find a proper dataset on which to estimate it. Moreover, DITRAS requires the training of a Markov chain on some trajectory dataset.

Project 6: Systematic comparison of collective mobility models

The student should develop a systematic comparison of Gravity and Radiation models based on at least four different datasets. The student may be inspired by Lenormand et al.'s paper on how to conduct a systematic comparison of collective mobility models.

In particular, the student should consider the following models (provided in scikit-mobility):

- Singly-constrained gravity model with power-law deterrence function
- Singly-constrained gravity model with exponential deterrence function
- Globally-constrained gravity model with power-law deterrence function
- Globally-constrained gravity model with exponential deterrence function
- Radiation model

As per the evaluation metrics, the student should use the following ones (refer to Lenormand et al.'s paper, Cha's paper, and scikit-mobility docs for the formulas):

- Common Part of Commuters (CPC)
- Normalized Root Mean Squared Error (NRMSE)
- Information Gain Statistics (I)
- Common Part of Links (CPL)
- Common part of commuters according to the distance (CPCd)

The dataset on which the comparison should be done can be chosen by the student, possibly of different types and geographic areas.

Examples of datasets:

1. same data source, but different geographic areas
2. same geographic area (e.g., city), but four different data sources
3. a mix of 1. and 2., justify your choice

The comparison should be done providing appropriate tables and/or plots in a notebook and adequately commenting the steps and the results obtained.

Project 7: Vehicles’ GPS traces and shortest paths

Do vehicles follow the shortest path on a road network? The student should answer this question using the Milano dataset, which describes the GPS traces of several private vehicles circulating in the city of Milan.

First, the student should split each vehicle’s global trajectory in sub-trajectories (you may exploit the `stay_locations` function provided in scikit-mobility). Then, the student should develop code to implement the following steps.

For each vehicle:

- For each sub trajectory:
    1. take the initial and the final position and associate them with the corresponding road using OSMnx.
    2. compute the shortest path and the fastest path on the road network provided by OSMnx given the starting and ending roads.
    3. associate each GPS point with the corresponding road (point snapping);
    4. use OSMnx to compute the path which passes through the obtained roads (either shortest or fastest, depending on what you are testing).

The student should develop appropriate measures to quantify to what extent the vehicles follow the shortest path (or the fastest path) calculated in point 2. This comparison should be done in terms of:

- difference in paths length (in meters)
- Jaccard coefficient between the two sets of roads (the real one and the shortest/fastest paths obtained from OSMnx)

Show and discuss the distributions of these two quantities, in terms of their shape, average, standard deviation and provide a thorough interpretation of the obtained results. Could you conclude, on the basis of these distributions, that vehicles tend to follow the shortest path or the fastest path? Visualize some single cases (i.e., cases in which the vehicle does or does not follow the shortest or fastest path) to provide evidence of your interpretation.

Repeat the entire pipeline for the Rome taxi dataset and the San Francisco taxi dataset. Do taxis tend to follow the shortest/fastest path?

Project 8: Spatial networks and small worlds

The small-world effect is a well-known phenomenon characterizing real-world social networks. Around a decade ago, the availability of data allowed scientists to discover the existence of this effect in several contexts, such as collaboration networks (see, e.g., the oracle of Bacon). Does a small world effect exist in human mobility as well?

Use the Brightkite, Gowalla and Foursquare datasets to construct an undirected network $M = (V, E)$ in which nodes in V are individuals and a link in E indicates that two individuals visited at least once the same location.

Analyze the structure of this network in terms of (use library networkx):

1. distribution of degree $P(k)$;
2. clustering coefficient CC;
3. average path length $< d >$;
4. betweenness centrality BC.

Comment on the results you find for points 1-4.

By small in the "small world phenomenon" we mean that the average path length $< d >$ depends logarithmically on the number of nodes (see here for details). Hence, “small” means that $< d >$ is proportional to $ln\, N$, rather than N or some power of N. In other words, a network has the small-world effect if d is around the natural logarithm of the number of nodes in the network. Is M a small world? Why?

Compare the shape of $P(k)$ and the values of CC, $< d >$, and BC of M with those of a social network $G = (U, I)$ where U are the users in the dataset used to construct M and a link in I indicates that two users are friends in the social network platform. Is G a small world? What’s the smallest world between M and G?

Project 9: Error in the reconstruction of OD matrices from different data sources

The student should select at least four geographic areas (e.g., cities, regions, countries), for each geographic area should select different mobility data sources (e.g., taxis traces, bike rides, checkins, vehicle traces, individual GPS traces), and should reconstruct the Origin-Destination (OD) matrix for each area/source.

In particular, the student should reconstruct the OD matrix given:

- a squared tessellation (try at least four different tile sizes);
- an h3 tessellation (try at least four different tile sizes);
- a Voronoi tessellation (e.g., based on OpenCellID);
- an official division of the territory (e.g., neighborhoods, regions).

The student should compute the OD matrices in two ways:

- using the built-in method of scikit-mobility that converts a `TrajDataFrame` into a `FlowDataFrame`;
- applying the `stay_locations` function (try different parameters). Two consecutive stops determine a movement by an individual. Clearly, stops should be spatially joined with the relative tessellation.

Systematically assess the error in the OD between the data sources and the various OD matrices computed in the different ways. Define a proper measure of error between flows, taking it from the literature or defining a new one on purpose. Find a succinct way (using plots, metrics, or both) to summarize to what extent OD matrices constructed with different tessellation/source differ among themselves.

Among the datasets, at least one area should be chosen to use flows estimated by official statistics (e,.g., the Milano dataset can be compared with official flows in Milan). For such areas, besides the analysis above, use flows from the official statistics as ground truth and rank each OD matrix by its similarity with the official flows.

Project 10: Nocturnal vs Day mobility patterns

Do nocturnal mobility patterns differ from day mobility patterns? In this project, the student should select at least four different public trajectory datasets (e.g., Geolife, Milano dataset, etc.) and investigate mobility patterns during night and day. In particular, the student should:

- define reasonable times to delimit night and day, taking into account the peculiarities of the datasets and the associated geographic areas;

- for each dataset, compare in a dual map night and day trajectories, commenting on whether some differences are evident from the visualization;
  - Note: trips may be detected using the `stay_locations` function or the `cluster` function, using the parameters the student believes are the most appropriate. The student should also decide (motivating the decision) what to do with trips that are at the border of night and day.

- compute, plot, and comment on the distribution of the main individual human mobility patterns using scikit-mobility, both for daytime trips and nighttime trips;

- compute, visualize, and comment on the Origin-Destination (OD) matrices (i.e., `FlowDataFrame`s) based on night and day and compare them both quantitatively and quantitatively.

- Develop a coherent discussion about what you find, i.e., write a sort of blog post in the form of a Jupyter notebook. Try to answer fundamental questions, such as: Are nocturnal trips more predictable than day ones? Are they typically longer? Etc.

.

Project 11: Data-driven estimation of urban gentrification

Yelp is a platform for sharing, reading, and collecting reviews about activities like restaurants, pubs, hotels, etc. AirBnB is a famous platform that operates an online marketplace focused on short-term homestays and experiences.

Gentrification is defined as "The process by which a place, especially part of a city, changes from being a poor area to a richer one, where people from a higher social class live" (Cambridge Dictionary). Scientific studies on Gentrification rely on insideairbnb.com data, a platform containing data about the airbnb presence in several cities in the world. The quantity of Airbnb facilities is one of the signals of the gentrification of a neighborhood.

The student should:

- Download the Yelp Dataset:
    - Among the JSON files, focus on the `yelp_academic_dataset_review.json` containing informations about the reviews of the platform;
    - Select the reviews for Toronto, Canada;
- Download the listing of the insideairbnb facilities of Toronto:
    - Here you find the list of all insideairbnb datasets; in the Toronto section, you find other data you may need, like the Toronto's neighborhood geojson object.

Both for the Yelp and for the insideairbnb data, consider the date of the first review as the date of creation/foundation of the facility.

Start from the insideairbnb data (here you find the description of the columns of each dataset type). With the help of scikit-mobility and Geopandas, split the city into neighborhoods and study the evolution of the **2** time series of:

- The number of facilities (insertion date, col. `first_review`) per neighborhood (per each facility, perform the spatial join between the lat/lng columns and the geojson of the neighborhood, or rely on the `neighbourhood_cleansed` column);
- The average price of facilities per neighborhood;
- For both the time series, study:

a. The “cumulative” one, i.e., the time series that aggregate the listing progressively, therefore keeping into account, at a given timestep, all of the previously inserted ones;
b. The “iterative” one, i.e., the time series that keep into account only the listing inserted in the current timestep;
c. A “k-sliding window” one, i.e., a time series that accounts for the insertion of the previous k-steps (find some meaningful values for the k parameter).

Do you notice any trend, pattern, seasonality? Do you find any breakout date?

Regarding the Yelp Dataset:

- Analyze the `yelp_academic_dataset_business.json`
- Pick only the first review per business activity
- Create the same time series as before.

Compare the Yelp and InsideAirBnB time series: Can you spot some neighborhoods with abnormal, consistent, values? Can you spot neighborhoods with an abnormal increment of both AirBnB houses AND Yelp activities? Would you define such neighborhoods gentrified?

Project 12: Euro bills and distance patterns

Eurobilltracker (EBT) is a website that tracks the position of euro bills, similarly to what wheresgeorge does. In this project, the student should:

- write python code to download euro bill traces using EBT APIs and download at least 1M bank notes from the website.
- make a bar chart (in matplotlib) and a choropleth map (in folium) to show the frequency of bank notes for each European country.
- visualize the trajectories of the top-10 bills in the downloaded dataset
- compute the percentage of bank notes that are next reported in the vicinity of the initial entry location (≤ 10 km) and the percentage of bank notes reported beyond a distance of 500 km.
- compute and plot the distribution of the following quantities:
    1. the distances traveled by each bank note, $P(r)$
        - compute $P(r)$ for three classes of initial entry locations: highly populated metropolitan areas (population > 120,000), cities of intermediate size (120,000 > population > 22,000) and small towns (population < 22,000)
    2. the radius of gyration of each bank note
    3. the number of bank notes per each user
    4. the radius of gyration of each user
    5. the 2-radius of gyration of each bank note
    6. the 2-radius of gyration of each user
    7. the country-uncorrelated entropy (CUE) of each trajectory in the dataset, where each country is a possible location.
- For each EU capital, compute and plot the distribution of:
    a. the distances of all bank notes firstly reported in that city;
    b. the maximum distance and the radius of gyration of the city, i.e., considering the movements of all the euro bills firstly reported in that city.

Write a blog post in the form of a Jupyter notebook to tell a coherent story regarding the analysis above. Submit both the notebook and the script to download data from the EBT website.

Project 13: Segregation in Networks

In this work, the authors propose a variation of the Schelling model in which they modify the structure of the city: agents, instead of moving over a lattice (grid), move over six different kinds of non-directed graphs (NDGs):

1. two-dimensional lattice with Von-Neumann neighborhoods (2D-VN);
2. two-dimensional lattice with Moore neighborhoods (2D-M);
3. regular NDGs (REG);
4. random network (RAND);
5. small-world network (SW);
6. scale-free network (SF).

The authors propose different parameters and simulations to characterize the dynamics of the model (see the paper).

Students should replicate the analysis in the paper (until Section 5.1, ignoring SSI and Mixed Deviation Index) by:

- Creating all the classes of graphs using Python library NetworkX;
  - e.g. `G = nx.scale_free_graph(100);`
  - Create a scale-free network of 100 nodes;
- Exploiting the possibility given by the MESA library of designing a NetworkGrid (see an example here), so as to test all the network structures in the paper;
- Compute the first segregation index proposed by the authors (Freeman Segregation Index, FSI), see below for details.
- Verify whether the results are consistent with the ones proposed by the authors.

Submit well-commented Python files for the MESA part and a well commented Jupyter notebook for the analytical part.

Freeman Segregation Index (FSI)

FSI is just one of the metrics of segregation. You can follow this procedure to compute it:

- Compute the number of “cross-tie” edges, i.e., undirected edges connecting two node of different category (say +1 and -1) is $\bar{e} = 2 \sum_{k=1}^{N} x_{k,+1} \cdot n_{k,-1}$,where
  - N is the number of nodes
  - $x_{k,+1}$ is a variable that indicates whether the k-th node has label -1(1 if the label is -1, 0 otherwise);
  - $n_{k,-1}$ is a variable that indicates the number of node with -1 label that are neighbors of the k-th node.

- The expected value of $\bar{e}$ in a graph with edges distributed at random (you can think about it as the average value of $\bar{e}$ in several experiment in which the edge configuration change randomly) is
  - $Exp(\bar{e}) = \frac{2NLP(1-P)}{N-1}$, where
  - L is the number of edges of the graph
  - P is the proportion of black nodes ($(1 - P)$ is the proportion of white nodes)

- FSI is therefore the deviation of the number of “cross-tie” edges from the expected value of them in a graph with edges distributed at random:
  - $FSI = \frac{Exp(\bar{e})-\bar{e}}{Exp(\bar{e})}$

Project 14: Can relocation patterns mitigate Schelling effects?

Schelling model postulates a simple relocation choice: when agents are unhappy they relocate choosing a cell, among the one they would be happy with, at random. The happiness of an agent is dependent on one parameter, the one controlling the threshold of intolerance (Memo: Schelling proved that even if everybody is willing to tolerate up to ⅔ of its neighbors different than him, the city ends up segregated). This is quite a simplistic, yet reasonable assumption.

The goal of this project is to test different policies of relocation. Students should test and implement four different versions of the Schelling model, according to different relocation policies (corresponding to different degrees of freedom).

When an agent is unhappy it relocates choosing:

1. *Pure random policy*:
   - Uniformly at random, a new, free, cell.
2. *Mild random policy*:
   - Uniformly at random, a new, free, cell, among the ones in which it would be happy
3. *Minimum-gain policy*:
   - Among the free cells it would be happy into, the one with the minimum level of happiness, i.e., a cell in which it *minimizes* its happiness above the tolerance threshold. In case of ties (cells with the same level of minimum satisfaction) agents must choose uniformly at random among them.
4. *Best policy:*
   - Among the free cells it would be happy into, the one that maximizes its happiness. Again, in case of ties, agents must choose uniformly at random among the cells that maximize its happiness.

All of the four variants of the Schelling movements must be coded and embedded into a Mesa Script (like the one seen in class). The student should compare times of convergence and segregation levels (both final and during the process) of the different approaches, providing meaningful analysis, possible explanations, visualizations and tables in a Python Notebook.
The comparison must show the different behaviors of the policies (in terms of time of convergence and segregation levels) by varying several parameters like the size of the grid or the tolerance of the agents: is it observable some trends? E.g. “large cities amplify final segregation levels in case of best policies” or “a growing trend is observed when…” etc.

Project 15: How Routing strategies impact vehicle-related measures

In this project, the student will study the impact of different routing strategies on some vehicle-related quantities (e.g., travel time and emissions) using SUMO.

In detail, the student should:

1) Download the road network of La Spezia (Italy) from OSMWebWizard.

2) Create an hexagonal tessellation of the city with a H3 resolution of 8;

3) Create an OD-matrix M, based on the tessellation, where the weight of a $m_{i,j} \in M$ is defined as $\frac{R_i R_j}{d_{ij}^{2}}$, where:

- R_i is the number of edges with the "from" or "end" node associated that falls in tile i;
- $d_{i,j}$ is the distance between the centroids of tile i and tile j.

Normalize the matrix so that the sum of all elements is 1 to make cell values used as transition probabilities.

4) Create a Traffic Demand TD, based on M, describing the movements of 1,000 vehicles. Assign the starting and ending edge associated with each vehicle's movement, selecting the edges randomly within the starting and ending tile (ensure that there exists a path, i.e., they are connected). Assign the departure time uniformly at random in the interval $[0, 600]$.

5) Starting from TD, compute the following traffic demands using different routing strategies:

- TD_{short}: assign the path from origin to destination using the **shortest** path;
- TD_{fast}: assign the path from origin to destination using the **fastest** path;
- TD_{w15}: assign the path from origin to destination using the tool **duarouter** with a random factor w = 15.

6) For each Traffic Demand computed in point 5, compute the following quantities for each vehicle.

- Total distance traveled;
- Travel time;
- CO2 emissions;
- NOx emissions;
- Fuel consumption.

see https://sumo.dlr.de/docs/TraCI/Vehicle_Value_Retrieval.html

7) Compare the vehicles' distribution of the measures computed in point 6) for each Traffic Demand and explain the results. For each vehicle, compare the difference in the measures computed for all the travel demands.

8) According to the result obtained:

- Which routing strategies save more fuel (on average)?
- Which routing strategies minimize the travel time (on average)?
- Which routing strategies pollute less (on average)?

Project 16: Route Diversity and Urban Emissions

In this project, the student should design two measures of route diversity and investigate whether there is a correlation between route diversity, CO2 and NOx emissions using SUMO.

More in detail, the student should:

1) Download the road network of Pisa (Italy) from OSMWebWizard.

2) Create an hexagonal tessellation of the city with a H3 resolution of 8;

3) Create an OD-matrix M, based on the tessellation, where the weight of a $m_{i,j} \in M$ is defined as $\frac{R_i R_j}{d_{ij}^2}$, where:

- R_i is the number of edges with the "from" or "end" node associated that falls in tile i;
- $d_{i,j}$ is the distance between the centroids of tile i and tile j.

Normalize the matrix so that the sum of all elements is 1 to make cell values used as transition probabilities.

4) Create a Traffic Demand TD, based on M, describing the movements of 500 vehicles. Assign the starting and ending edge associated with each vehicle's movement selecting the edges randomly within the starting and ending tile. Assign the departure time uniformly at random in the interval $[0, 300]$.

5) Starting from TD, compute the following traffic demands using OpenStreetMap routing service and duarouter:

- TD_{OSM_short}: assign the path from origin to destination using OpenStreetMap with the parameter preference=shortest;
- TD_{OSM_fast}: assign the path from origin to destination using OpenStreetMap with the parameter preference=fastest;
- TD_{OSM_rect}: assign the path from origin to destination using OpenStreetMap with the parameter preference=recommended;
- TD_{dr15}: assign the path from origin to destination using the tool duarouter with a random factor w = 15.

6) Propose and implement two measures (d_1 and d_2) to quantify the "diversity" of road usage. Motivate your choices and provide a formal definition of the measures. For each Traffic Demand computed in point 5):

- compute the total CO2 and NOx emitted by the vehicles;
- measure the road usage (e.g., for each edge, the number of vehicles that traveled that edge) and apply d_1 and d_2.

7) According to the results obtained:

- Which routing strategies maximize the diversity? why?
- Which routing strategies minimize the diversity? why?
- Is there a correlation between diversity and CO2 emissions?
- Is there a correlation between diversity and NOx emissions?
- Are CO2 emissions and NOx emissions correlated?

Project 17: Understanding international migration

The student should use this international migration flows dataset to study international migration between countries and world zones. First of all, visualize the information in the two datasets:

- plotting two circular plots (like that in Figure 4);
- creating two `FlowDataFrame`s where the tessellation is the division of the world in countries and world zones. Plot the two datasets using the method `plot_flows`. Make the visualization as nice and informative as possible exploiting the method's arguments.

Then, plot the distribution of migration flows and flow-distances. A flow-distance between two locations consists of the migration flow between them multiplied by the distance between the two locations. For a country, consider the position of its capital as its "centroid". A world zone's centroid is the centroid of the country's capitals.

Split the dataset into a training and a test set and use the former to fit:

- *singly-constrained* gravity models
  - use the total number of out-migrants to estimate O_i
  - use the population as measure of location relevance
  - variant 1: with power-law deterrence function (model G_1)
  - variant 2: with exponential deterrence function (model G_2)
- *doubly-constrained* gravity models
  - use the total number of out-migrants to estimate O_i
  - use the population as measure of location relevance
  - use the total number of in-migrants to estimate D_i
  - variant 1: with power-law deterrence function (model G_3)
  - variant 2: with exponential deterrence function (model G_4)

Evaluate on the test set, both qualitatively and quantitatively, the goodness of the gravity models, as well as of a radiation model (R) where the population is used as an estimate of the opportunities. What's the model that best approximates the real migration flows? Do the previous task for both countries and world zones. The population of a world zone is the sum of the population of all its countries.

Repeat the previous task using the GDP (Gross Domestic Product) of each location instead of the population (for both the gravity and the radiation models). The GDP of a world zone is the sum of the GDP of all its countries. Do it for both countries and world zones. What's now the best model?

Project 18: Urban desegregation model

Can we define a mechanism to desegregate a segregated city?

The student should use MESA to develop a simulation model that starts from an already segregated space (i.e., output of a Schelling model's execution) and generates a desegregated space.

The simulation should be *location-based* rather than agent-based: at each simulation step, it is not the individuals who choose where to go, but a *district* decides which individuals to attract or reject. The district decides to attract or repel individuals on the basis of a measure of *racial diversity*: a district should be racially mixed as much as possible.

With respect to the original Schelling model, where each square is a single location, in this model we introduce a *multigrid*: each of the m cells that compose the space (districts) can host up to $n > 1$ individuals. Therefore, at most n individuals can reside in a district, one for each location. The racial diversity D of a district is calculated as the Shannon's entropy on the races of the individuals in it:

$$D = -\sum_{r \in R} p(r) \log p(r)$$

where r is a race and R is the set of all possible races. For simplicity, we assume that there are only two races, namely $R = \{r_1, r_2\}$.

The desired diversity D^* is the minimum racial diversity that each district must respect. Note that when $D^* = 0$, we want only one race in a district (i.e., a fully segregated district). Since our model starts from a segregated city (output of Schelling's model), we expect that in the beginning a district's racial diversity x is far from the desired one D^* ($D_x << D^*$).

At each step of the desegregation model, the following steps take place:

- each district x evaluates its racial diversity D_x;
- if $D_x < D^*$, i.e. if the racial diversity of the district is lower than the expected diversity, then the district is unhappy;

- unhappy districts are divided into two equivalence classes (R_1 and R_2) based on race, creating a network initially without arches;
- districts in R_1 connect with districts R_2 to exchange individuals in order to increase racial diversity while respecting the racial tolerance of individuals;
- The contact between the districts (i.e., the creation of the arches in the bipartite network) can be done based on a random principle or a “best” principle (so that the diversity is maximized).

The simulation ends when racial tolerance is respected AND there are no unhappy districts. To facilitate the convergence of the model, we can relax this last condition and terminate the model when at least k districts are happy.

The result of the simulation should be essentially equivalent to the input of the Schelling model in which the neighborhoods have maximum racial diversity. In other words, our location-based model generates a de-segregated city from a segregated city.

Study how the behavior of the model changes with different parameter values (number of agents, locations, and districts, tolerance, homophily, etc.).

Project 19: Data-driven Geo Schelling model

Mesa-Geo is an extension of MESA that offers the possibility of handling geospatial data when defining the space of the model. Here you can find an example of a simple GeoSchelling model, i.e., a model in which agents move inside some geographical areas. There are two types of `GeoAgents` (extension of the `Agent` class of Mesa): people (movable) and regions (fixed GeoJSON geometries). Each person resides in a randomly assigned region, and checks the color ratio of its region against a predefined happiness threshold at every time step. If the ratio falls below a certain threshold (e.g., 40%), the agent is unhappy and randomly moves to another region. People are represented as points, with locations randomly chosen within their regions. A region's color depends on the color of the majority population it contains (i.e., point in polygon calculations, not so important for our scope). The main difference from classical Schelling is that more agents can occupy a cell and consider as neighbors all the agents that live in their same cell.

- Download from OpportunityInsights a csv file containing information about the racial composition of each census tract in the US (here a description of the columns of the dataset). As you can see in the column description, each census tract is uniquely identified by three identifiers (2010 FIPS) – state, county, and tract, i.e. the first three columns of the dataset. Create a column named, for example *geoid* being the concatenation of these three ids.
- Visit the US Census Bureau website at this link:
  - Select, FOR YEAR **2010,** the state of New York and the County of New York;
  - Download the associated shapefile, and read it with GeoPandas
  - Save it into a GeoJSON file
  - Now you are able to use it as space in Mesa-Geo
- Choose two categories among the variables, calculate them aggregating the existing variables (e.g., whites/non-whites, black/non-blacks), or deduce them (poor/non-poor, note that you have the poor share column, but not the rich one: it can be calculated, for example as *1-poor share*).
- As you can see, almost each variable is calculated for three timestamps: 1990, 2000, 2010. These are the dates corresponding to the US census years. Consider only the information about 2010.
- As you can see, the GeoSchelling example contains a small number of agents: the idea of this project is to move a step towards the definition of the so called *data-augmented ABM:*

- Set up the model so that it can use as *space* the pre-calculated GeoJSON as grid
- Suppose each NYC census tract was populated, in 2010, by 1200 agents
    - If you want to refine it, you can search the web for the exact population
    - Or you can multiply 1200 by the factor of the column *popdensity2010*
- Using the value of the population, calculate the exact number of agents of your two categories per each census tract
- Assign the number of agents to the GeoSchelling model
- Let the simulation run for some steps… Do you find something interesting? Be creative, propose a simple, but complete, analysis:
    - For the tolerance parameter, some hints:
        - Leave the ⅔ of classical Schelling
        - Increase it to 50%
        - …
    - For the steps:
        - Is there a number of steps that make the simulation converge?
        - Do you observe any difference in the evolution of the process by changing some parameters?
    - Can you provide a visualization/interpretation of the process, static or dynamic?

Project 20: Football matches as mobility networks

During a football match, players move on the field to attack and defend. This generates a series of movements that can be analyzed to understand the players behavior.

The student should use the Wyscout open dataset, describing the “events” in all matches of seven competitions, to analyze the mobility of players. A player’s movement is defined by consecutive events made by that player in the match.

First, compute and plot the distribution of (Euclidean) distances in the entire dataset, as well as the distribution of the average distance per match and per player. What is the shape of these distributions? Are they similar to the distributions of geographic distances observed in mobility datasets?

Second, split the field into x equally-sized tiles (try different values of x) and construct the mobility network of each match. A node in this network identifies a zone and a weighted edge indicates how many times any player moved between two zones during the match. Create these networks for each match and the two teams separately. Then:

- compute and plot the distribution of the node degree for 1) the entire dataset, 2) for each competition separately, and 3) for each team separately. Can you find differences?
- compute the football-uncorrelated entropy, defined as the predictability of the ball to be in a particular zone of the field. The higher this entropy, the harder it is to predict where the ball is during a game. Do it for 1) the entire dataset, 2) each competition separately, and 3) each match separately. Can you find differences?

Third, define the H indicator of a team in a match as the harmonic mean of the average degree and the standard deviation of the degree of the mobility network. Compare the H indicator of two teams and evaluate how frequently a team with a higher H indicator wins over the opponent. Can the H indicator be a valuable descriptor of a team’s strength?

Develop a simulation in which you take all matches in at least one of the competitions available in the dataset, compute for each team the H indicator, and make a team win (+3 points) if its H indicator is higher than the opponent. Compare the final ranking you obtain with the real final ranking for that league/year. Are they correlated?

Write a blog post in the form of a Jupyter notebook to tell a coherent story about the analysis above.

Project 21: Trajectory compression

The student should modify the scikit-mobility function `compress` to provide three different compression algorithms, taking in input a `TrajDataFrame`, `tdf`, and returning a new `TrajDataFrame` with a reduced number of points. The new compression methods should be the following:

- Ramer–Douglas–Peucker, 1973
- Driemel–HarPeled–Wenk, 2010 (Section “Curve Simplification”)
- Imai–Iri, 1988 (see slides)

The new `compress` function should return a `TrajDataFrame` containing only the rows selected by the algorithm.

The signature of the new function should be as follows:

```
compress(tdf,  spatial_radius_km=0.2,
               algorithm=”default”,
               <other_parameters>)
```

where:

- `tdf` is the input `TrajDataFrame`.
- `spatial_radius_km=0.2` is the parameter required by the current implementation of compress, that we keep for compatibility.
- `algorithm=”default”` is the selected compression algorithm. “default” represents the current implementation. Other options are ‘ramer’, ‘driemel’, ‘imai’.
- `<other_parameters>` are the parameters required by the new algorithms – each algorithm will consider only the relevant parameters and ignore the others.

The student should test and compare the results of the three methods against a real-world dataset of trajectories, both through global statistics and a selection of examples to show on a map.

The developed code should be delivered as a `compression.py` file, which is a modification of the existing one on the scikit-mobility repository. Provide also a subfolder named `examples` in which you show how the developed function works on the test data.

Project 22: Implementing Trajectory Segmentation

The student should add to scikit-mobility a function `traj_segmentation` that, similarly to the existing `stay_locations`, takes in input a `TrajDataFrame`, `tdf`, and returns a new `TrajDataFrame` that splits each individual's global trajectory into sub-trajectories based on the Trajectory Segmentation procedure described in Siła-Nowicka et al. IJGIS 2015, 30:5.

An individual's global trajectory $T^{(u)}$ consists of all the points of that individual u in the `TrajDataFrame` (i.e., all rows in `tdf` with `uid=u`). An individual's sub-trajectory $T_i^{(u)}$ indicates all points belonging to the same segment identified by the function..

In practice, the new `traj_segmentation` function should return a `TrajDataFrame` containing all the rows of the original `tdf` but with an additional column `tid`, which contains the identifier of each sub-trajectory of an individual. `tid` must have an integer value between 1 to k, where 1 is the first sub-trajectory in chronological order and `k` is the number of segments detected.

The signature of the new function should be as follows:

```
traj_segmentation(tdf, window=10, gap=2000, inplace=False)
```

where:

- `tdf` is the input `TrajDataFrame`.
- `inplace=True` indicates that the original `tdf` should be overwritten with the new one (i.e., the one containing the new `tid` column); `inplace=False` indicates that a new `TrajDataFrame` should be returned.
- all the other arguments are as described in the reference paper.

NOTE: The student is invited to include additional parameters if needed, following the indications in the reference paper and their own creativity.

The student should also test the provided code on a small set of trajectories of their own choice (for instance, traces of taxis), comparing the number and average length of the segments obtained with those yielded by the standard `stay_locations`, and visually showing examples on a map..

The developed code should be delivered as a `segmentation.py` file, which is a modification of the detection.py file on the scikit-mobility repository. Provide also a subfolder named `examples` in which you show how the developed function works on the test data.

Project 23: Deviation Patterns

It is well known that actual mobility of vehicles often does not follow shortest/fastest paths (ref. Project 7), yet it is not clear what are the reasons. This project aims to dig deeper in the phenomenon. The basic approach required involves to test two possible (non-exclusive) hypotheses: (1) there are some roads that are almost systematically avoided by real vehicles; (2) there are special events (road works, accidents, exceptional traffic conditions) that make the road unappealing for a limited time – either in a specific day, or regularly every day (e.g. systematic traffic jams). The student is welcome to integrate them with additional ideas to explore.

First, follow the same process described in Project 7 to obtain, for each trajectory *T* of the input dataset, the sequence $T_R = \{s_1, \ldots, s_n\}$ of segment IDs that maps *T* to the road network, and the sequence $T_S = \{s^*_1, \ldots, s^*_m\}$ of the **fastest path** between start and end of *T* (by construction, you should obtain $s^*_1 = s_1$ and $s^*_n = s_m$, though in general m != n – a good test to put in an `assert` test…).

Second, for each input trajectory *T* we compute the set T_M of segments that the real vehicle "missed" w.r.t. the shortest path T_S, namely $T_M = T_S - T_R$ (set difference).

Testing theory 1. For each road segment *r*, compute its "missing" absolute frequency $f_{ABS}(r)$ as size of {T in dataset | r in T_M }, and its relative frequency $f_{REL}(r)$ as $f_{ABS}(r)$/|{T | r in TS}|. Identify the highest frequency ones (both for relative and absolute definition) and show them on a map. Are there extreme peaks, namely hugely avoided roads?

Testing theory 2. Follow a similar approach as above, yet this time we associate to each road segment *r* in T_S, T_R and T_M the time they were traversed, i.e. these sets contain now pairs *(r, t)* where t is the traversal time. Time can be discretized into hours. We consider two cases: (i) t is described as "date + hour_of_the_day"; (ii) t is just described as "hour_of_the_day", thus not considering the date. For both cases do the following:

(a) Find the pairs (r,t) with highest frequency, and do a similar analysis as for theory 1

(b) Select the 3 most interesting pairs (s,t), and study what trajectories of T_S actually crossed it, by plotting them on the map highlighting the road segments with highest frequency (e.g. associating colors based on the number of trajectories passing through the segment).

Project 24: Schelling and The Pursuit of Happyness

This project is an extension of Project 14. The question we want to answer is the following: can we learn relocation strategies in a Schelling simulation that improve convergence time?

Phase 1: MESA-based simulation. Write and run a Mesa script implementing a *mild random policy* for relocation, as described in Project 14. Collect the complete traces of the simulation, extracting at least the sequence of relocations performed along the simulation.

Phase 2: build a training set. Analyzing the simulation traces, create a dataset that contains a row for each relocation happened, composed of <agent-feat, start-feat, end-feat, happiness_duration>, where:

- agent-feat: this is a set of computed features describing the agent. Basic features should include the number of time steps where the agent relocated. The student can propose others.
- start-feat: this is a set of computed features describing the context of the location where the agent was before relocating. Basic features should include the coordinates in the grid and the number of similar agents in the neighborhood. The student can propose others.
- end-feat: same as start-feat, but for the destination location.
- happiness_duration: a number describing for how many simulation steps, starting from the relocation, the agent remained in the destination location. This is equivalent to say, for how many time steps the agent remained happy – before relocating again.

Phase 3: learn to relocate. Discretize the field happiness_duration into two classes: “short” and “long”, based on a duration threshold that you decide through analysis of the distribution of values. Use the dataset you obtain to train a classification model that predicts the happiness duration class based on the other fields (excepted happiness_duration, of course). Try with multiple classification methods and follow the usual process to select the best combination of model + parameters. If the training set is too small, run step 1 multiple times (on different initial configurations), to enlarge it. We call the model “ML-Schelling”.

Now, modify the Mesa script in such a way that when an agent has to relocate it uses the classification model to choose the destination. More precisely, we compute all the features adopted in the training phase (namely: agent-feat, start-feat and end-feat) for all pairs *(o,d)*, where “o” is fixed and corresponds to the actual location of the agent, and “d” is any of the remaining locations. Clearly, the components agent-feat and start-feat are always the same, whereas end-feat changes with each possible destination. We apply the model to each pair and select the one that predicts “long” happiness with the highest confidence/probability.

Phase 4: test the model. Run a set of simulations on different initial conditions using both the “mild random” and the ML Schelling policies, and evaluate, on each simulation setting, how many iterations are needed with the two methods to converge and how long, on average, does happiness of agents last. Does ML give an advantage?

Project 25: Moving Clusters

The student should implement a function `moving_clusters` to be integrated in scikit-mobility, which takes in input a TrajDataFrame, tdf, and returns a DataFrame describing the moving clusters contained in tdf. The implementation should follow the instructions given in the moving cluster paper by Kalnis-Mamoulis-Bakiras.

Resampling. Moving clusters require that trajectories have points at a given fixed sampling rate, e.g. every 60 seconds. The function should take care of preprocessing the input TrajDataFrame in order to interpolate points at timestamps that are multiples of a given input time parameter, e.g. if the requested sampling is 120 seconds, we should interpolate points for each timestamp of form 2022/11/24 10:08:00, 2022/11/24 10:10:00, 2022/11/24 10:12:00, etc. The interpolation should assume that objects move on a straight line at uniform speed between two consecutive points.

The signature of the new function should be as follows:

```
moving_clusters(tdf, time_sampling_seconds=60,
                     dbscan_radius_mt=500,  dbscan_min_pts=5,
                     theta=0.8)
```

where:

- `time_sampling_seconds=60` indicates the (re)sampling rate to apply to the input trajectories, which is by default set to 1 minute.
- `dbscan_radius_mt=500` indicates the radius (or “epsilon”) of the DBSCAN clustering algorithm used by moving clusters, by default set to 500 meters.
- `dbscan_min_pts=5` is the minimum number of points used by DBSCAN, set to 5 by default.
- `theta=0.8` is the minimum “integrity threshold” used to link clusters between time slices (see reference paper), by default set to 80%.

The output DataFrame should contain a row for each moving cluster found, consisting of: timestamp of the time slice where the pattern starts; the pattern duration; the minimum, maximum and average number of cluster members; a list containing a sub-list for each time slice, each listing the trajectory IDs that belong to the cluster at that time.

The student should test the function on at least two public datasets, and discuss some sample results.

The developed code should be delivered as a `moving_clusters.py` file. Provide also a subfolder named `examples` in which you show how the developed function works on at least one sample dataset.

Project 26: T-Patterns

Implement a simple version of Trajectory Patterns, based on a pre-defined grid, where transition times are computed a posteriori – as simple average transition time, considering if variance is not too large.

The student should implement a function `t_patterns` to be integrated in scikit-mobility, which takes in input a TrajDataFrame, tdf, and returns a DataFrame describing the Trajectory Patterns contained in tdf, together with a reference grid. The implementation should be a simplified version of what described in the Trajectory Pattern Mining paper.

The function should realize the following steps:

- First, a spatial regular grid is created, composed of square cells of given side length.
- Second, each trajectory is translated to a sequence of cell IDs with associated timestamp (the same of the original point in the trajectory)
- Third, a sequential pattern mining task is run over the set of sequences obtained above, according to a minimum support threshold
- Fourth, for each pattern P the function should identify the set S of input sequences where it occurs. On each input sequence in S, the function should compute the transition time for each transition A→B in the pattern[1], computed as tt(A,B) = time(B)-time(A), where time(X) is the timestamp of cell X in the input sequence. Thus, each transition A→B of the pattern is now associated with all its transition times. Compute the mean and standard deviation of these values.

The signature of the new function should be as follows:

```
t_patterns(tdf, cell_size=250,
               min_sup=0.20,
               grid=None)
```

where:

- `cell_size=250` indicates the side length of each cell in the regular grid, by default set to 250 meters.

[1] Given a sequential pattern A→B→C→D→..., the transitions are A→B, B→C, C→D, ...

- `min_sup=0.20` is the minimum support of the patterns, expressed as the fraction of input trajectories containing them (0<=`min_sup`<=1), by default set to 20%.
- `grid=None` is an optional parameter that provides the grid to use in the algorithm. If `grid==None`, then a new grid is computed, based on the cell_size.

The output should be formed by a pair (output_grid, patterns), where `output_grid` is the grid used by the function (equal to the input grid, if provided) and patterns is a DataFrame containing, for each pattern: support, list of cell IDs, list of mean transition times, list of transition time standard deviations.

The student should test the function on at least two public datasets, and discuss some sample results.

The developed code should be delivered as a `t_patterns.py` file. Provide also a subfolder named `examples` in which you show how the developed function works on at least one sample dataset.

Project 27: Trajectory Prediction with Higher order Markov Chains

Preliminary notions. In standard Markov Chains (MC) the probability of the next event in a sequence depends only on the current one (i.e. the last event seen so far), while the previous history of the sequence is completely neglected. An extension of this is given by k-th order Markov Chains, where we consider the last k events in the sequence, i.e.

$$P(X_n \mid <X_1, X_2, \dots, X_{n-1}>) \quad = \quad P(X_n \mid <X_{n-k}, X_{(n-k)+1}, \dots, X_{n-1}>)$$

Standard MCs are a particular case with k=1.

The student should implement two functions: `mc_traj_train` and `mc_traj_predict`, to be integrated in scikit-mobility. The first one takes in input a TrajDataFrame “tdf_train” and returns an MC model; the second takes as input a TrajDataFrame “tdf_test” and an MC model, and returns a DataFrame describing the predictions made by the model for each trajectory in tdf_test.

Function **`mc_traj_train`** should realize the following steps:

- First, a spatial regular grid is created, composed of square cells of given side length.
- Second, each input trajectory is translated to a sequence of cell IDs.
- Third, the parameters of a k-order MC are learned from the set of sequences obtained above, with *k* being an input parameter.
- Fourth, the grid and a data structure containing the MC model parameters are returned.

The signature of `mc_traj_train` should be as follows:

`mc_traj_train`(tdf, cell_size=250, mc_order=1, grid=None)

where:

- cell_size=250 indicates the side length of each cell in the regular grid, by default set to 250 meters.
- mc_order=1 defines the order of the MC model to adopt, by default set to 1, namely the standard MC.
- grid=None is an optional parameter that provides the grid to use in the algorithm. If grid==None, then a new grid is computed, based on the cell_size.

The output should be formed by a pair (output_grid, mc_model), where output_grid is the grid used by the function (equal to the input grid, if provided) and mc_model is the data structure containing the MC model parameter.

Function **`mc_traj_predict`**, instead, should realize the following steps:

- First, each input trajectory is translated to a sequence of cell IDs using the grid passed as input.
- Second, the k-order MC passed as input is applied to the set of sequences obtained above, yielding the next cell visited for each sequence.
- Third, each predicted cell is added to the input trajectories as an extra point having coordinates corresponding to the center of the cell.

The signature of `mc_traj_predict` should be as follows:

`mc_traj_predict`(tdf, mc_parameters, grid)

where:

- mc_parameters contains the parameters of the MC model to adopt. This is a mandatory parameter.
- grid is the grid to use in the algorithm. This is a mandatory parameter.

The output should be a TrajDataFrame where each input trajectory is now extended with a new point.

Implementation notice. k-th order MCs are relatively easy to implement (remember that there are no hidden states…) by computing conditional probabilities. The student can either implement them from scratch, or make use of existing libraries, like pomegranate. Reading the documentation of the latter is anyway suggested, as it helps understanding a few details of the problem – e.g. how should be make predictions if a test trajectory has less than k points?

The student should test the functions on at least two public datasets, dividing them into a training set and a test set, studying the effect of the order of the MC on the performances.

The developed code should be delivered as a `markov_chains.py` file. Provide also a subfolder named `examples` in which you show how the developed function works on at least one sample dataset.

Project 28: A matter of resilience

When a road is closed or becomes extremely slow, its impact on the overall mobility of the city depends on various factors, including its centrality and the existence of alternative routes. The objective of this project is to identify the road segments that are more critical and can thus create resilience issues to the network.

As a case study, take the road network of Pisa, let say a square area of 10km x 10km. The ideal process to run is the following:

1. choose one road segment r
2. consider all possible pairs of origin-destination (o,d) in the city, where o and d are nodes in the road network
3. compare the fastest path travel times obtained for all (o,d) pairs on the full road network vs. the network obtained removing segment r
4. define an impact measure for r based on the increase of travel times (e.g. the ratio between the average route durations)
5. repeat the process (steps 1–4) for all segments r in the road network
6. identify the roads that are less resilient, i.e. that create the highest impact on the city mobility.

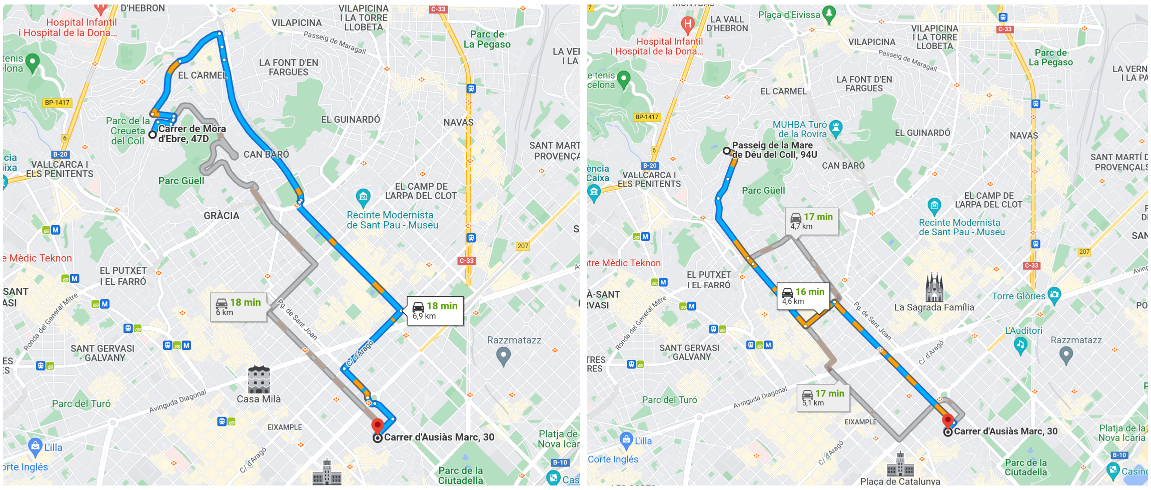
If the number of nodes and segments in the network is too high to compute all segments and all fastest paths, a form of sampling can be adopted.

Create a map showing the impact of each road segment, for instance through colors, and discuss the results. Question: is the impact measure equivalent to centrality of the edge? If not, provide a rationale and – possibly – some empirical proof in Pisa.

Organize the result in a well-commented notebook.

Project 29: Traveling as a (dis)continuous function

Sometimes, the best path to reach a destination D can be very different from the best path to reach a different destination D' (from the same starting point), even if D and D' are very close. We name these cases "best path discontinuities". E.g. (same origin, destination changed by ~50 meters):



In some cases the difference is in terms of path, while the length/duration are the same; in others (as the example above) also length/duration can change.

The objective of this project is to measure the phenomenon and then use it to compare different cities.

Phase 1: the process. The student should implement a function `city_discontinuity` that takes as input the road network of a city and outputs two distributions of discontinuity measures (defined below).

The process is the following:

1. consider all possible pairs of origin-destination (o,d) in the city, where o and d are nodes in the road network and such that distance(o,d) >= min_travel_dist (which is an input parameter)
2. for each (o,d), randomly choose another destination d' such that distance(d,d') lies within a predefined interval [d_min, d_max] – other input parameters
3. compare the fastest path for (o,d) with that for (o,d'), with two measures:
    a. continuity$_1$(o,d) = min(time(o,d), time(o,d')) / max(time(o,d), time(o,d'))
    b. continuity$_2$(o,d) = Jaccard(path(o,d), path(o,d'))

   where time(o,d) is the duration of the fastest path, and path(o,d) is the set of road segments traversed along the fastest path from o to d.

4. return an histogram of N_{bins} bins ranging from 0 to 1 (which is the theoretical maximal range of both continuity measures) for the $continuity_1$ values, and another similar histogram for $continuity_2$.

Phase 2: the use case. Select 10 European cities, including Rome, London and Barcelona, downloading their road networks cut to a bounding box of similar (and reasonable) size. Then, compare the resulting distributions, discussing the results. Questions:

- Which cities “suffer” from discontinuity?
- Are there specific areas of the city that increase discontinuity?

Organize the result in a well-commented notebook.

Project 30: Random Perturbation of Paths

In this project, the student should define (at least) two randomization algorithms for a path between an origin and a destination.

Given a sequence of SUMO edges representing the path between an origin and a destination (e.g., computed with the shortest or fastest path), propose two algorithms to create a random perturbation of the path (Figure 1).

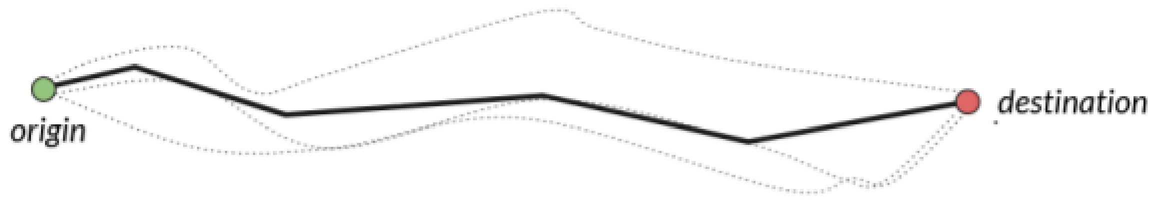


Figure 1. Example of perturbations (dotted lines) of a given path (solid line)

Each algorithm should have the following properties:

- A parameter $w \in [1, \infty)$ that controls the level of randomization ($w = 1$ means no randomization). The higher the value of w, the more the path is perturbed. Provide some empirical proof of this property.
- A "spatial" parameter s to control the spatial extension of the perturbed path. Provide some empirical proof of this property.
- The algorithm should be non-deterministic, i.e., given the same input (sequence of edges, s, and w), the algorithm may propose a different path perturbation. Provide some empirical proof of this property.
- The algorithm should quantify the difference between the original and perturbed paths, in terms of:
  - difference in paths length (in meters)
  - Jaccard coefficient between the two sets of roads (the originale one and the perturbed path)

First, provide a mathematical definition and a Python implementation of the algorithms (they should work on SUMO road networks). Visualize on Folium some examples of a path and the perturbation obtained through the designed algorithms for different parameters' values.

Then test the algorithms as follows:

1) Download the road network of Pisa (Italy) from OSMWebWizard.

2) Create a Traffic Demand TD describing the movements of 1500 vehicles. Assign the starting and ending edge associated with each vehicle's movement selecting the edges randomly. Assign the departure time uniformly at random in the interval $[0,\ 300]$.

3) Starting from TD, compute the following traffic demands using the perturbation algorithms implemented (select the w and s parameters of your algorithms as you prefer):

- TD_{alg1}: assign the path from origin to destination as a perturbation of the fastest path using the first algorithm proposed;
- TD_{alg2}: assign the path from origin to destination as a perturbation of the fastest path using the second algorithm proposed;
- TD_{dr15}: assign the path from origin to destination using the tool duarouter with a random factor $w\ =\ 15$.

4) For each TD computed in point 3, calculate the following quantities for each vehicle.

- Total distance traveled;
- Travel time;
- NOx emissions.

5) Compare the vehicles' distribution of the measures computed in point 4 for each TD and discuss the results.

6) According to the results obtained:

- Is path perturbation beneficial?
- Which one of the proposed algorithms has the best impact on each of the measured quantities?
- Are the algorithms proposed “better” than duarouter with respect to NOx emission and travel time? Why? provide an interpretation.

Project 31: SUMO Map Matching

In this project, the student should design, develop, and test a map matching algorithm. Given a SUMO road network and a sequence of lat/lon points, the algorithm should map the sequence of points on SUMO edges. The algorithm's output is a route (i.e., a sequence of connected edges) representing and describing the lat/lon sequence.

The signature of the function should be:

```python
def map_match(road_network, lat_lon_points):

      # Your code (well commented)
```

The proposed algorithm should:

- Be resilient to point inaccuracies;
- Be resilient to "noise" points (e.g., points that lie in intersections);
- Do not depend heavily on the density of the points.

Quantify the goodness of the map matching between the original edge list and the map-matched one in terms of:

  - Difference in paths length (in meters)
  - Jaccard coefficient between the two sets of roads (the original one and the map-matched one)

First, provide a mathematical definition and a Python implementation of the algorithm (it should work on SUMO road networks). Visualize on Folium some examples of a path and the map-matched path obtained through the map matching. Explain **all** the choices you have made.

Then, to test the algorithm, the student should:

1) Download the road network of Milan (Italy) from OSMWebWizard.

2) Perform the following steps:

A. Create a Traffic Demand TD describing the trips of 5,000 vehicles. Assign the starting and ending edge associated with each vehicle's movement selecting the edges randomly. Ensure that the origin and destination are connected and that the shortest path has a length ≥ 1.5 km;

B. Given TD, use duarouter with $w = 10$ to compute a traffic demand DR that describes the routes (sequence of connected edges) associated with each trip;

C. Transform each route $route_i \in DR$ into a sequence of lat/lon points $p_i \in P$ that describes the selected route (tip use SUMO and collect the lat/lon points). Can you imagine a way to collect the lat/lon points without using SUMO?

D. For each $p_i \in P$ apply the proposed algorithm ($mapmatch$) and verify whether $mapmatch(p_i) = route_i$.

   a. For how many routes the map matching algorithm reconstructs **exactly** the route? (i.e., Jaccard coefficient equal to 1);
   b. What is the total distribution of the Jaccard coefficient?

E. Reduce the "density" (number of points) as follows (Figure 1):

   a. Randomly discard a fraction d of points from $p_i \in P$ obtaining $\bar{p}_i \in \bar{P}$ (do **not** discard the first and last point);
   b. Vary the values of $d \in \{0.1,\ 0.25,\ 0.5,\ 0.75,\ 0.9\}$ and investigate how the map-matching performance changes; i.e., plot the distribution of the Jaccard coefficient of $mapmatch(\bar{p}_i) = route_i \ \forall\ \bar{p}_i \in \bar{P}$;
   c. For how many routes the map matching algorithm reconstructs exactly the route? (i.e., Jaccard coefficient equal to 1).

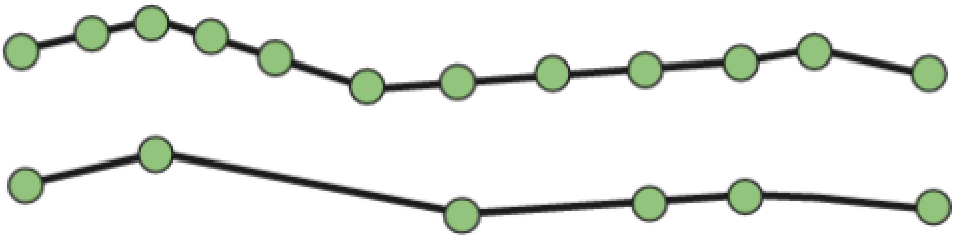


Figure 1. The original sequence of GPS points (top) and the sequence after the removal of random GPS points (bottom).

According to the results obtained:

- When does your algorithm fail to correctly map-match a sequence of points? provide an interpretation of why it happens.
- Is there some correlation between the length of the path (in km) and the accuracy of the map matching algorithm?