MOBIDATALAB

Labs for prototyping future mobility data sharing solutions in the cloud

# D4.10 Data Protection Tools (V2)

**The anonymization module of the MobiDataLab transport cloud prototype.**

29/11/2023

Author(s): Jesús A. MANJON (URV), Sergio MARTINEZ (URV), Benet MANZANARES (URV), Alberto BLANCO (URV)

MobiDataLab is funded by the EU under the H2020 Research and Innovation Programme (grant agreement No 101006879).

# Summary sheet

| | |
|---|---|
| **Deliverable Number** | D4.10 |
| **Deliverable Name** | D4.10 - Data Protection Tools V2 |
| **Full Project Title** | MobiDataLab, Labs for prototyping future Mobility Data sharing cloud solutions |
| **Responsible Author(s)** | URV |
| **Contributing Partner(s)** | - |
| **Peer Review** | HOVE, HERE |
| **Contractual Delivery Date** | 30-09-2023 (extended to 30-11-2023) |
| **Actual Delivery Date** | 29-11-2023 |
| **Status** | Final |
| **Dissemination level** | Public |
| **Version** | V1.0 |
| **No. of Pages** | 14 (62 including annexes) |
| **WP/Task related to the deliverable** | WP4/T4.5 |
| **WP/Task responsible** | AKKODIS/URV |
| **Document ID** | MobiDataLab-D4.10_DataProtectionToolsV2-v1.0 |
| **Abstract** | New anonymization methods, privacy-preserving analysis techniques and the capability of compute new utility and privacy measures have been added to the anonymization module during the second part of the project.<br><br>This deliverable contains a detailed description of all these methods, which have been integrated into the latest version of the anonymization module. |



Funded by the European Union

# Legal Disclaimer

MOBIDATALAB (Grant Agreement No 101006879) is a Research and Innovation Actions project funded by the EU Framework Programme for Research and Innovation Horizon 2020. This document contains information on MOBIDATALAB core activities, findings, and outcomes. The content of this publication is the sole responsibility of the MOBIDATALAB consortium and cannot be considered to reflect the views of the European Commission.

# Project partners

| Organisation | Country | Abbreviation |
|---|---|---|
| UNIVERSITAT ROVIRA I VIRGILI | Spain | URV |
| CONSIGLIO NAZIONALE DELLE RICERCHE | Italy | CNR |
| AKKODIS | France | AKKODIS |

# Document history

| Version | Date | Organisation | Main area of changes | Comments |
|---|---|---|---|---|
| v0.1 | 03/08/2023 | URV | - | Draft to be internally reviewed |
| v0.2 | 05/08/2023 | URV | All sections | Feedback from internal reviewers integrated |
| v0.3 | 6/11/2023 | URV | Annexes | Restructure deliverable by adding annexes |
| v0.4 | 17/11/2023 | HERE, HOVE, URV | - | Feedback from HERE and HOVE integrated |
| V0.5 | 24/11/2023 | AKKODIS | All sections | Coordinator QC |
| v1.0 | 29/11/2023 | AKKODIS | All sections | Submission |

# Executive Summary

A trajectory microdata set is a microdata set that contains trajectory data. This kind of datasets are special because the location information included in them can be considered both as quasi-identifiers and sensitive information. Trajectory microdata is prone to privacy attacks on individual users because of two defining characteristics: Trajectory data are highly unique and hard to anonymize.

The main goal of task T4.5 is to develop data processing modules that apply data protection and anonymization techniques that will be later uploaded the Transport Cloud.

The first version of the demonstrator was released in July 2022. It included 3 anonymization methods for protecting trajectory data, selected from the catalogue of techniques compiled in T2.2 considering the use case requirements elicited in T2.6, and the computation of utility metrics in trajectory databases. It also provides a command line interface (CLI) that lets users anonymize a mobility dataset and compute some utility measures over both the original and the anonymized datasets in a straightforward way.

This second version includes 6 anonymization methods, 1 privacy-preserving analysis method, 4 methods to compute different utility measures and 1 method to compute a privacy metric. The CLI has been extended to handle the new functionality and the demonstrator is now ready to be deployed into a server and to process requests through an API.

The anonymization module has been designed with a focus on modularity, where pseudonymization or anonymization methods can be built using different components dedicated to preprocessing, clustering, distance computation, aggregation, etc.

Deliverable D4.10 describes the characteristic of the final version of the demonstrator and includes a detailed user manual. Demonstrator is available at https://github.com/MobiDataLab/mdl-anonymizer.

A video demonstration is available at

https://raw.githubusercontent.com/MobiDataLab/mdl-anonymizer/master/docs/videos/MDL_demo_final_v1_compressed.mp4



Funded by the European Union

# Table of contents





Funded by the European Union

# List of figures

# List of tables



# Abbreviations and acronyms

| Abbreviation | Meaning |
|---|---|
| MDAV | Maximum Distance to Average Vector |
| CLI | Command Line Interface |
| DTW | Dynamic time warping |
| STLC | Spatio-temporal linear combine |

# 1. Introduction

## 1.1. Project overview

There has been an explosion of mobility services and data sharing in recent years. Building on this, the EU-funded MobiDataLab project works to foster the sharing of data amongst transport authorities, operators, and other mobility stakeholders in Europe. MobiDataLab develops knowledge as well as a cloud solution aimed at easing the sharing of data. Specifically, the project is based on a continuous co-development of knowledge and technical solutions. It collects and analyses the advice and recommendations of experts and supporting cities, regions, clusters, and associations. These actions are assisted by the incremental construction of a cross-thematic knowledge base and a cloud-based service platform, which will improve access and usage of data sharing resources.

## 1.2. Purpose of this deliverable

This document presents the anonymization module of the MobiDataLab Transport Cloud prototype. The anonymization tool includes methods for the protection of mobility data, methods to analyze a mobility dataset in a privacy-preserving way and the computation of privacy and utility metrics. All the implemented methods are described in detail in Annex 6.1. A user manual is provided in Annex 6.2. This document is a companion report to the demonstrator available at https://github.com/MobiDataLab/mdl-anonymizer and to the video demonstration available at

https://raw.githubusercontent.com/MobiDataLab/mdl-anonymizer/master/docs/videos/MDL_demo_final_v1_compressed.mp4

## 1.3. Structure of the deliverable

This deliverable is organized as follows. Section 2 gives a general overview of the Anonymization Module and introduces the components implemented in the final version of the module. Sections 3 and 4 give an overview of Command Line Interface and the API. Section 5 presents the conclusions. Finally, the annexes provide a detailed description of the design of the module, the developed components, and the Command Line Interface tool.


Funded by the European Union

# 2. Module overview

In D2.3 State of the Art on Transport and Mobility Data Protection Technologies, we studied a collection of anonymization methods for trajectory microdata in the literature. This study suggested that methods in that category tend to follow a common pattern that consists of data preprocessing, comparison or clustering of data according to some spatial (or spatiotemporal) distance metric between positions and/or trajectories and an optional final aggregation or filtering step. For this reason, we design our anonymization tool with a focus on modularity, where pseudonymization or anonymization methods can be built using different components dedicated to preprocessing, clustering, distance computation, aggregation, etc.

On the other hand, we want our module to be as format-agnostic as possible, and thus we choose to load data from comma-separated values (CSV), which makes it directly compatible with tools such as QGIS and mobility data analysis libraries such as GeoPandas and scikit-mobility. The module is also able to load parquet files. Apache Parquet is an open-source, columnar data format designed for efficient data storage and retrieval. It provides efficient data compression, so parquet files are smaller than CSV files, and they can be read and written much faster. This decision, however, does not limit the possibility of adding additional data loading components to deal with different data formats.

We implemented the following modules and methods (see Annex 6.3 for a detailed description):

- **Anonymization: methods for the anonymization of a dataset of trajectories.**
  - Simple generalization
  - Protected generalization
  - Microaggregation
  - Time partition microaggregation
  - SwapAllLocations
  - Swapmob
- **Trajectory distances: methods to measure the distance between two trajectories.**
  - Graph distance
  - Spatio-temporal distance
- **Aggregation: methods for the aggregation of trajectories (i.e., calculation of the centroid of a set of trajectories).**
  - Mean trajectory
  - Closest trajectory to centroid
- **Analysis: methods for the analysis of a dataset of trajectories.**
  - QuadtreeHeatMap
- **Clustering: methods for the clustering a set of trajectories.**
  - MDAV
- **Measures: methods to measure the utility and privacy of a dataset of trajectories.**
  - Information loss via RMSE

- Information loss via normalized RMSE
- Propensity score
- Disclosure risk via record linkage

The anonymization module can be used as a Python library but it also provides a command line interface (CLI) that lets users to use all the module functionalities in a straightforward way. The module is also ready to be deployed in a server and to process requests through an API.

# 3. Command line interface

The developed package provides a command line interface (CLI) that lets users anonymize a mobility dataset, perform privacy-preserving mobility analysis, compute some utility measures over both the original and the anonymized datasets and filter a dataset in a straightforward way.

```
$ python -m mdl_anonymizer anonymize -f parameters_file.json
```

Annex 6.4 provides a detailed description of how to use this command line interface.



Funded by the European Union

# 4. API

The anonymization module is also ready to be deployed in a server to provide all its functionality remotely.

To start the server application, use the following command:

```
$ uvicorn mdl_anonymizer.server.main_api:app --reload --host 0.0.0.0 --port 8000
```

## 4.1. End points

- **POST /anonymize/**
  - Upload a dataset to be anonymized.
  - Params:
    - Input_dataset
    - config_file
- **POST /analyze/**
  - Upload a dataset to be analyzed.
  - Params:
    - Input_dataset
    - config_file
- **POST /compute_measures/**
  - Compute measures over the original and the anonymized dataset
  - Params:
    - original_dataset
    - anonymized_dataset
    - config_file
- **POST /filter/**
  - Upload a dataset to be filtered
  - Params:
    - Input_dataset
    - config_file

Sometimes the previous tasks take some time to compute. For this reason, all the previous endpoints return a *task_id*:

```
{
  "status": "OK",
  "message": "Your task is being processed. Use the 'GET /task/' endpoint to obtain the
results later",
  "task_id": "30c2ad9df0f9470abfd8198b4cb4f86a"
}
```

After some time, the result can be obtained from the following endpoint:

- **GET /task/?task_id={task_id}**

## 4.2. Using the command line interface to query the API

The command line interface can also be used to send requests to the API.

First, define the server API address in the *'/server/config_api.json'* file:

```
{
    "api_server": "http://127.0.0.1:8000"
}
```

These are the available commands:

```
$ python -m mdl_anonymizer anonymize-api -f parameters_file.json

$ python -m mdl_anonymizer analysis-api -f parameters_file.json

$ python -m mdl_anonymizer measures-api -f parameter_file.json

$ python -m mdl_anonymizer filter-api -f parameter_file.json
```

The same parameter files than used in the standalone use case can be used in this case.

After some time, the result can be obtained by using the following command:

```
$ python -m mdl_anonymizer get-task -t {task_id}
```

# 5. Conclusions

This document presented the anonymization module of the MobiDataLab Transport Cloud prototype and accompanies the demonstrator available at https://github.com/MobiDataLab/mdl-anonymizer.

The final version of the anonymization module includes 6 anonymization methods, 1 privacy-preserving analysis method and 5 methods to compute different utility and privacy metrics. It also provides a command line interface (CLI) that lets users to use all the module functionalities in a straightforward way. The module is also ready to be deployed in a server and to process requests through an API.

The anonymization module has been designed with a focus on modularity, where pseudonymization or anonymization methods can be built using different components dedicated to pre-processing, clustering, distance computation, aggregation, etc. We have focused on making it easy to add new methods and components, in order to encourage contributions from other researchers.

The anonymization module has been used to anonymize, for example, a very large dataset provided by HOVE to make it public to the participants in the Mobidatalab X-athons. This dataset collects the daily requests sent to the Navitia route planner[1] during 2022 and the first quarter of 2023 starting or ending in the Île-de-France region. The responses included in the dataset are not real journeys but routes proposed by the planner. Every route includes the start and end points as well as some transit points. Since this information could jeopardize the privacy of the users, the dataset was anonymized using the "Time partition Microaggregation" method, a version of the well-known microaggregation method for very large mobility datasets where the application of microaggregation would not be feasible.

However, although the anonymization module let users to protect mobility datasets in a simple way, choosing the right anonymization method and parameters is a complex problem that requires a nuanced view and careful consideration. In general, one always has to make a decision on the desired trade-off between privacy and utility, favoring one or the other depending on the use case requirements.Complex anonymization procedures are typically manual processes involving several rounds of analysis and data transformation, tailored each technique and parameters to the dataset and user's needs. Therefore, we recommend that these anonymization procedures be informed and/or performed by anonymization experts in collaboration with domain experts.

[1] https://doc.navitia.io/


Funded by the European Union

# 6. Annexes

## 6.1. Entity class diagram

Entity classes implement the data model and the non-interactive functionalities (i.e., anonymization mechanisms of the anonymization module). To achieve a high performance, classes have been designed to be cohesive and decoupled, and associations between them have been defined with the logical navigation workflow.

As example of the structure of classes of a module, Figure 1 shows the entity class diagram of the anonymization module. The rest of modules have the same structure but changing the name of the interface, factory class and implemented method classes. The main highlights of the anonymization class diagram are:

- *Dataset* represents a data set of trajectories. Datasets are structured in several classes interrelated with navigable associations (*Dataset* → *Trajectory* → *TimestampedLocation*), so that all the data associated with a data set (*i.e.,* trajectory values) can be efficiently queried during the anonymization process.
- *AnonymizationMethodFactory* is a static class that has the method *get* called to obtain an instance of the anonymization method indicated in the parameter *methodName*. The dataset to be anonymized is passed in the *dataset* parameter and the parameters needed by the anonymization method are indicated in the *parameters* argument. This method creates and returns the instance of the specified anonymization method.
- Anonymization algorithms are defined as specializations of the *AnonymizationMethod Interface* interface, which defines the interfaces of the main anonymization operations (*run* and *getAnonymizedDataset*). Thus, new anonymization algorithms or variations of the three currently implemented ones can be easily added by specializing classes and reusing or extending the code.


Funded by the European Union

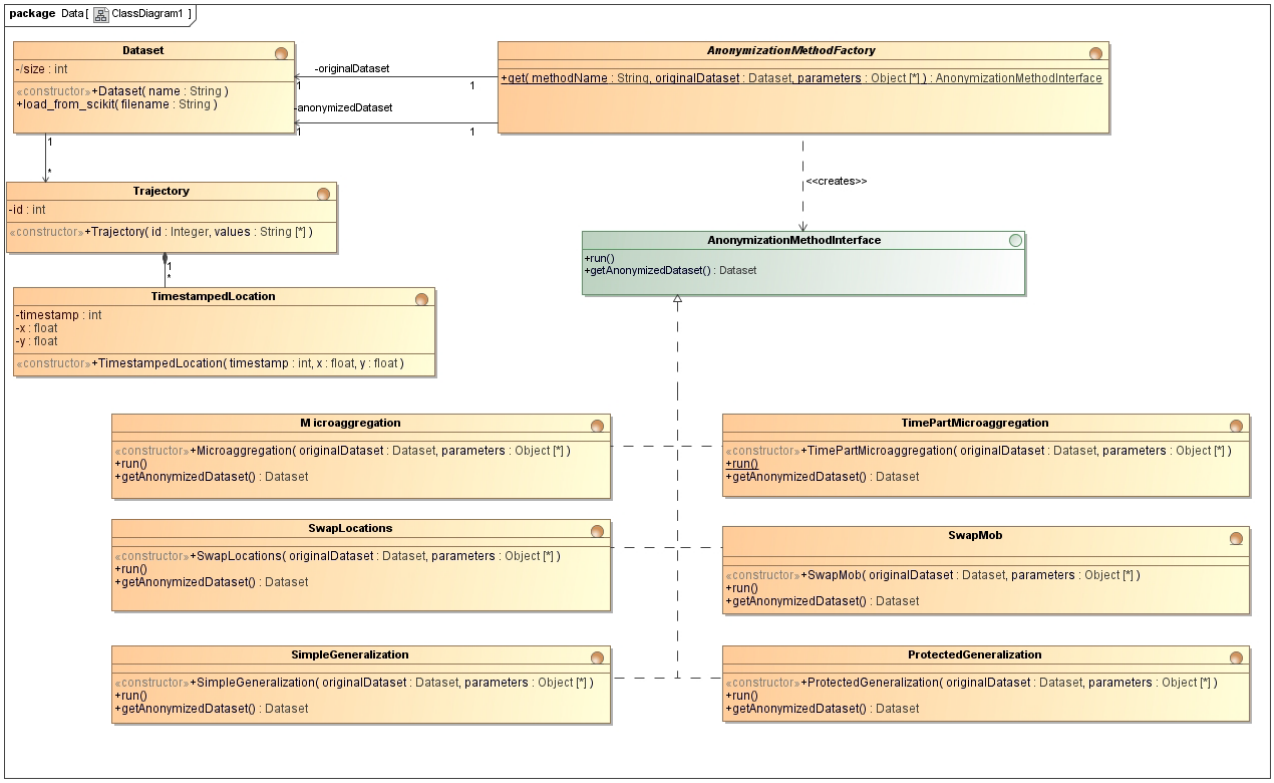


*Figure 1. Entity class diagram*

## 6.2. Adding new methods

The architecture design described above allows developers to easily add new methods, algorithms, or variations of the implemented ones mentioned in section 2. To do so, developers should simply follow the next steps:

1. Create a Python class that implements (inherits from), respectively:
    a. *AnonymizationMethodInterface*, for new anonymization methods
    b. *TrajectoryAggregationInterface*, for new aggregation methods
    c. *AnalysisMethodInterface*, for new analysis methods
    d. *ClusteringInterface*, for new clustering methods
    e. *MeasuresMethodInterface*, for new measure methods
    f. *DistanceInterface*, for new trajectory distance methods
2. The constructor of the new class must receive as arguments first the original dataset and then the necessary parameters for the new method.
3. Implement the inherited class method *run()* by including the code that executes the logic of the new method (*e.g.*, in the case of a new anonymization method, the routine that anonymizes the original dataset)

4. Include the reference description to the new class method in the *config.json* file located at the root of the project library (below is the structure of the *config.json* file including the references to the currently developed methods). The reference must be included inside the method type (anonymization, clustering, aggregation, trajectory_distances, analysis, or measures) and it must contain the name of the method and the path name of the new class.

```
{
  "anonymization_methods": {
    "SimpleGeneralization": {
      "class":
"mdl_anonymizer.anonymization_methods.Generalization.Simple.SimpleGeneralization"
    },
    "ProtectedGeneralization": {
      "class":
"mdl_anonymizer.anonymization_methods.Generalization.Protected.ProtectedGeneralization"
    },
    "SwapLocations": {
      "class":
"mdl_anonymizer.anonymization_methods.SwapLocations.SwapLocations.SwapLocations"
    },
    "SwapMob": {
      "class": "mdl_anonymizer.anonymization_methods.SwapMob.SwapMob.SwapMob"
    },
    "Microaggregation": {
      "class":
"mdl_anonymizer.anonymization_methods.Microaggregation.Microaggregation.Microaggregation"
    },
    "TimePartMicroaggregation": {
      "class":
"mdl_anonymizer.anonymization_methods.Microaggregation.TimePartMicroaggregation.TimePartM
icroaggregation"
    }
  },

  "clustering_methods": {
    "SimpleMDAV": {
      "class": "mdl_anonymizer.clustering.MDAV.SimpleMDAV.SimpleMDAV"
    }
  },

  "aggregation_methods": {
    "Mean_trajectory": {
      "class": "mdl_anonymizer.aggregation.Martinez2021.mean_trajectory.Mean_trajectory"
    },
    "Closest_locations_to_mean_trajectory": {
      "class":
"mdl_anonymizer.aggregation.Martinez2021.closest_locations_to_mean_trajectory.Closest_loc
ations_to_mean_trajectory"
    },
    "Closest_trajectory_to_mean_trajectory": {
      "class":
"mdl_anonymizer.aggregation.Martinez2021.closest_trajectory_to_mean_trajectory.Closest_tr
ajectory_to_mean_trajectory"
    }
  },

  "trajectory_distances": {
    "Martinez2021": {
      "class": "mdl_anonymizer.distances.trajectory.Martinez2021.Distance.Distance"
    }
  },
```

```
  "analysis_methods": {
    "QuadTreeHeatMap": {
      "class": "mdl_anonymizer.analysis_methods.QuadTreeHeatMap.QuadTreeHeatMap"
    }
  },

  "measures_methods": {
    "ScikitMeasures": {
      "class": "mdl_anonymizer.measures_methods.ScikitMeasures.ScikitMeasures"
    },
    "TrajectoriesRemoved": {
      "class": "mdl_anonymizer.measures_methods.TrajectoriesRemoved.TrajectoriesRemoved"
    },
    "Rsme": {
      "class": "mdl_anonymizer.measures_methods.Rsme.Rsme"
    },
    "PropensityScore": {
      "class": "mdl_anonymizer.measures_methods.PropensityScore.PropensityScore"
    },
    "RecordLinkage": {
      "class": "mdl_anonymizer.measures_methods.RecordLinkage.RecordLinkage"
    }
  }
}
```

Example of *config.json* file

Once the new method has been implemented and referenced in the *config.json* file as described above, the new method can be used in the same way as those already developed and currently included in the library as described in Annex 6.4.

## 6.3. Developed methods

This section presents a detailed description of the software components implemented in the current version of the anonymization module. As described in annex 6.1, anonymization mechanisms can be built from different components, including distance computation between locations and trajectories, aggregation and clustering algorithms, and postprocessing operations.

### *6.3.1. Trajectory distances*

Many of the trajectory anonymization mechanisms in the literature (refer to D2.3, Section 6) consist, in some way, in reducing the unicity of trajectories typically by grouping or aggregating the ones that are similar. This similarity is measured with a distance metric to quantify the resemblance of the trajectories to be grouped or aggregated. This distance must consider both the space and time dimensions. In the following subsections, we describe the methods to compute a distance between two trajectories that have been developed so far and integrated into the anonymization module.

### 6.3.1.1. Graph distance

This method, presented in [1], is based on the computation of a spatial distance between pairs of trajectories only when they are 'contemporaries' (that is, they overlap in time). The spatial distance between each pair of trajectories is only computed for locations within the time interval that they share. Then a graph is built where i) the nodes represent trajectories; ii) nodes $T_i$ and $T_j$ are adjacent only if they are contemporaries, and iii) the weight of the edge $(T_i, T_j)$ is the distance between the trajectories $T_i$ and $T_j$. Given the distance graph for $T = \{T_1,..., T_n\}$, the distance $d(T_i, T_j)$ for two trajectories is easily computed as the minimum cost path between the nodes $T_i$ and $T_j$, if such path exists.

Figure 2 shows an example of distance graph. $T_1$ and $T_2$ are adjacent because they overlap in time and a distance exists between them. However, $T_2$ and $T_4$ are not connected because they are not contemporaries and their distance is not defined. $T_1$ and $T_8$ are not connected (they do not overlap in time) but their distance can be computed as the minimum cost path between the nodes, in this case d = 6.01.

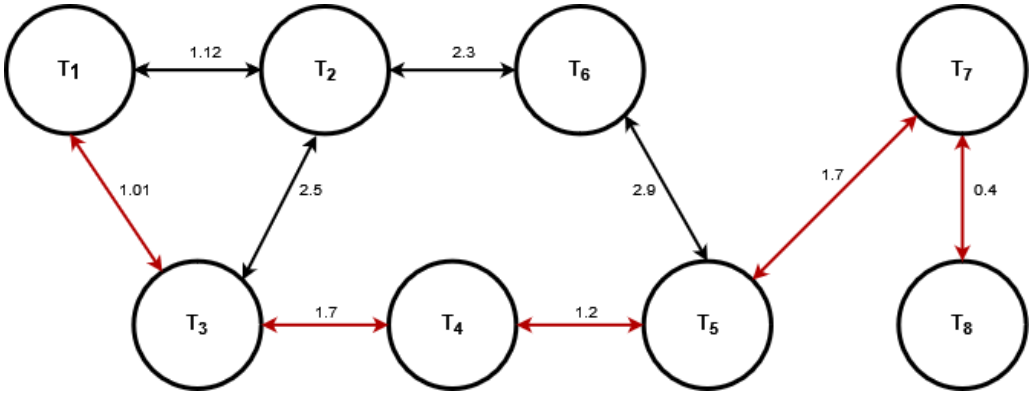


*Figure 2. A distance graph. Red arrows are the shortest path between T1 and T8.*

### 6.3.1.2. Spatio-temporal distance

Most of the spatial distances can be extended into spatio-temporal distances by balancing the weight of the spatial and temporal dimensions [5]. Additionally, since each trajectory may have a different number of points, a method to sample, match and compare individual points within trajectories is needed.

Dynamic time warping (DTW) [3] is one of the most popular algorithms to measure the distance between trajectories. The DTW algorithm searches through all locations in two trajectories for a pair of points at a minimum distance. The computational cost of DTW is $O(mn)$, where $m$ and $n$ are the number of points in each of the trajectories. In [5], authors propose the spatio-temporal linear combine (STLC) distance, where spatial and temporal similarities are linearly combined according to a parameter $\lambda$ which assigns a weight to both time and space similarities. The computational cost of STLC is quadratic $O(mn)$.

In [2] we defined a distance measure based on DTW and STLC that tries to find the best match between pairs of points in the trajectories with a low computation cost. The computational cost of the distance calculation is just O(*h*) where *h* is the average number of points in the two trajectories, which makes this distance suitable for large data sets. To calculate the distance, the algorithm selects a list of *h* representative pairs of points, proportional to the number of points in each trajectory. Only these points are considered during the distance calculation. Figure 3 shows an example of the pairs of points selected to compare trajectories $T_a$ and $T_b$. Origin and destination points of the trajectories are always selected.

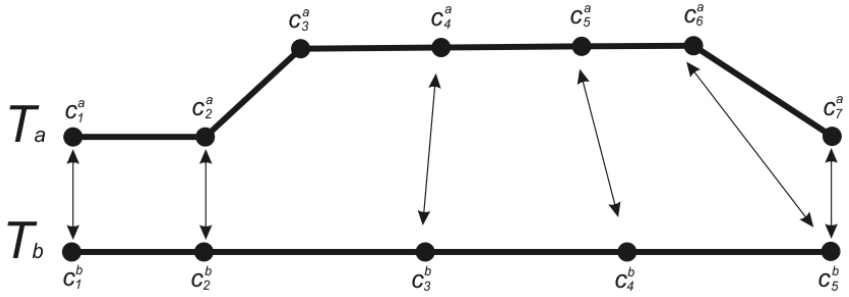


*Figure 3. Trajectory distance calculation. Arrows show the pairs of points selected to calculate the distance between trajectories Ta and Tb*

Once a pair of points has been matched, the distance between them is computed. Similar to STLC, we define a distance that linearly combines spatial and temporal distances between pairs of points:

$$dist(c_i, c_j) = dist_{spa}\left((x_i^a, y_i^a), (x_j^b, y_j^b)\right) + \lambda \cdot (|t_i^a - t_j^b| \cdot V_{ab})$$

where $dist_{spa}$ is the spatial distance between coordinates $(x_i^a, y_i^a)$ and $(x_j^b, y_j^b)$, and $V_{ab}$ is the mean velocity of trajectories $T_a$ and $T_b$. The temporal distance $|t_i^a - t_j^b|$ is multiplied by the mean velocity of trajectories $V_{ab}$ to convert it to a spatial distance that can be added to $dist_{spa}$. To mitigate the excessive weight of the temporal distance (because implicitly assumes that subjects are constantly moving away from each other) we weight the temporal component with a parameter $\lambda = \frac{D}{V \cdot T}$, where *D* is the maximum distance between points in the data set, *V* is the mean velocity of the trajectories in the data set and *T* is the maximum time difference between points in the data set.

## 6.3.2. Trajectory aggregation

Trajectory aggregation consists in replacing a set of trajectories by a single representative, namely the centroid of the set. Since the replacement causes information loss, the calculation of accurate centroid trajectories is crucial to retain the utility of the anonymized data set as much as possible.

### 6.3.2.1. Mean trajectory

In [2], we propose a trajectory aggregation algorithm with linear computational cost which makes it suitable for large data sets. The algorithm works as follows: being $Q$ a set of trajectories to be aggregated, the trajectory aggregation algorithm calculates $h$ as the mean of points in the trajectories included in $Q$. Similar to how the distance computation presented in 6.3.1.2 works, we select a sample of *h* points from each trajectory in $Q$, proportionally to the number of points. This yields *h* sets of $|Q|$ points each. The centroid $(t^c, x^c, y^c)$ of each of the $h$ sets is calculated as the component-wise mean of the $|Q|$ points in the set. Finally, the centroid trajectory $Q_c$ is obtained as the concatenation of the *h* centroids, that is, $Q_c = \{(t_1^c, x_1^c, y_1^c), \ldots, (t_h^c, x_h^c, y_h^c)\}$.

Figure 4 shows an example of aggregation of two trajectories $T_a$ and $T_b$. The arrows show the sample points selected from each trajectory. The resulting centroid trajectory $Q_c$ is depicted with a dotted line.

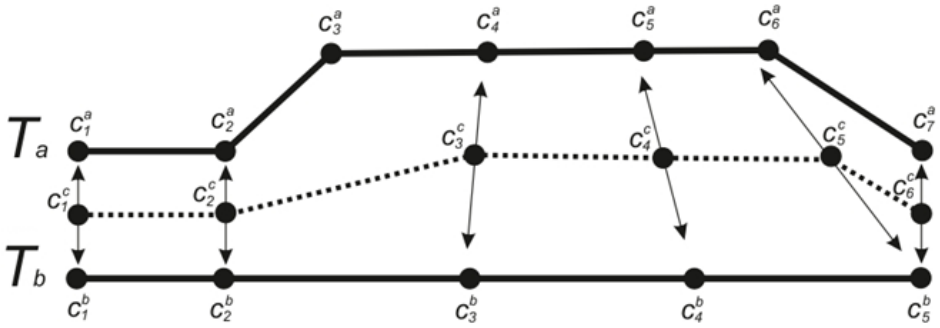


*Figure 4.Aggregation of trajectories. The arrows show the points selected to aggregate the trajectories Ta and Tb. The dotted line represents the centroid trajectory taken as output of the aggregation*

### 6.3.2.2. Closest trajectory to centroid

The aggregation process of a set of trajectories results in a centroid that represents the aggregated trajectories. This process implies some information loss because the original trajectories are replaced with the calculated centroid. In the mean trajectory calculation described in the previous section, the information loss is minimal because the centroid is the mean of the aggregated trajectories. However, the resulting centroid trajectory could contain locations that are not consistent with the domain of the dataset. For example, in a dataset of taxi cabs trajectories, the resulting trajectory could contain off road locations which are impossible to occur in this dataset. In certain cases, we might need that the aggregated trajectories are possible in the dataset. To do so, as alternative to the mean trajectory, we propose to calculate the centroid as the closest trajectory to the mean of the aggregated trajectories as follows: being $Q$ a set of trajectories to be aggregated, we calculate the mean trajectory $Q_c$ as described in the previous section, and, finally, we obtain the centroid trajectory $Q'_c$ as the trajectory in $Q$ that minimizes the distance to $Q_c$. The resulting trajectory contains locations existing in the dataset but, at the cost of more information loss with respect to the mean trajectory due to the additional replacement of the mean trajectory by the closest one.

### 6.3.3. Anonymization methods

This section presents the anonymization mechanisms that have been implemented in the anonymization module. Along with the description of each method we provide some images showing the visual results of the anonymization. These images were generated from the cabs dataset described in Section 6.3.4.5, with 10,282 trajectories and 155,690 locations. Below is a visual representation of the original dataset:

*Figure 5.Original locations*

*Figure 6.Paths generated from the original locations*

#### 6.3.3.1. Simple Generalization

One of the strategies to anonymize trajectory datasets are based on mitigating the disclosure risk. This kind of strategies follow the utility-first anonymization approach. They do not provide any formal privacy guarantees but aim to reduce reidentification risks by applying different techniques, such as noise addition, generalization and coarsening with heuristic parameter choice. After the application of such techniques, the disclosure risk is calculated (for some objective disclosure prevention, *i.e.*, identity disclosure or attribute disclosure). If the obtained risk is still too high, the techniques are applied with more strict parameters. Several of these techniques can be applied both in the context of location-based services and in static trajectory microdata sets.

A naive approach is to hide the original locations by means of generalization, specifically, replacing exact positions in the trajectories by approximate positions, *i.e.*, points by centroids of areas. If a tessellation file is not provided, the method first builds a regular spatial tessellation which covers the whole bounding box of the dataset. Then, we replace every location of the dataset with the centroid of the corresponding tile. If more than one consecutive locations of the same trajectory lie in the same tile, we can choose keeping them with their original timestamps or aggregating them by computing the average of all these timestamps.

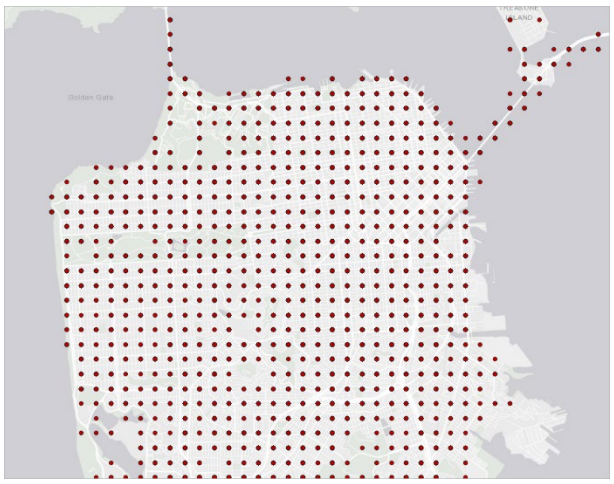

*Figure 7. Generalized locations generated using a squared tessellation*

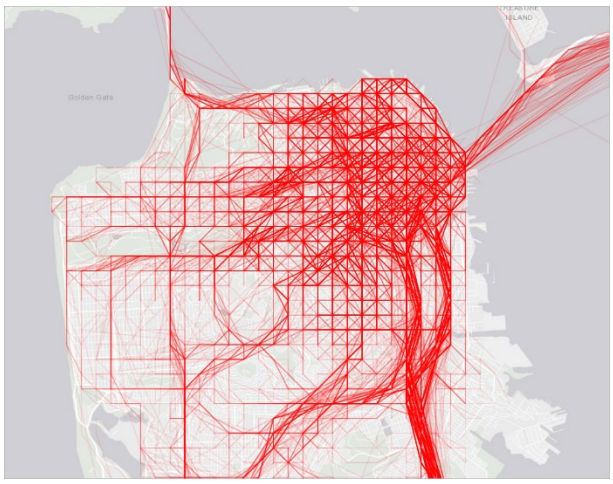

*Figure 8. Paths generated from the previous anonymized locations*

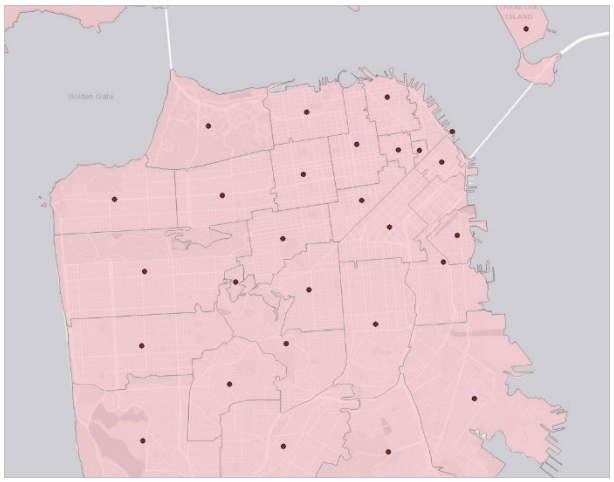

*Figure 9. Generalized locations using a custom tessellation. In this case, the zip codes of San Francisco*

*Figure 10. Paths generated from the previous anonymized locations*

However, although all the exact locations have been obfuscated, some “anonymized” trajectories could be still linked to the original ones because certain combination of spatiotemporal points could uniquely identify a record in a database or an individual. Hence, we need more refined methods to try to mitigate Record Linkage attacks (See Section 6.3.4.4).

### 6.3.3.2. Protected Generalization

An attacker with access to the anonymized trajectory dataset may know: 1) the details of the schema used to anonymize the data; 2) the fact that a given user *U* is in the dataset; and 3) a number *KL* of locations relative to *U* and the time at which they were visited (with a certain precision).

With this information, the attacker could try to infer the anonymized locations corresponding to the known ones and identify the entire trajectory relative to *U*.

*k*-Anonymity limits the capability of an attacker who knows a set of features on a subject (some locations and/or timestamps in the case of trajectory data) to successfully re-identify individuals in a released dataset. A trajectory dataset satisfies *k*-anonymity if, for each combination of locations and/or timestamps, at least *k* trajectories share the same combination. Thus, the probability of correct re-identification is at most *1/k*.

The *ProtectedGeneralization* method aims to build a *k*-anonymous version of the original dataset by means of generalization and suppression. It consists of several steps:

**ProtectedGeneralization algorithm**

Requires: *X*: dataset of trajectories
*KL*: the length of the attacker background knowledge
*k*: level of privacy (minimal number of trajectories sharing the same combination of locations)
*tiles_filename:* custom tessellation
*tile_size:* if a custom tessellation is not provided, size of every tile in a squared tessellation
*time_interval:* size of every time level
*strategy:* how to generalize locations within the same tile (compute average or take the tile centroid)
*time_strategy:* generalize timestamps?

Outputs: *X'*: *k*-anonymized version of *X*

1. If *tiles* is None:
   | generate a spatial squared tessellation of the territory covered by *X*
2. if *time_interval* is not None:
3. | expand the tessellation in a third dimension
4. If *tiles* is None:
5. | optimize the generated tessellation of every time level by merging some tiles
6. transform all trajectories into a sequence of tiles
7. compute all combinations of size *KL* and count appearances
8. while a combination appears less than *k* times:
9. | modify trajectory sequences by removing tiles appearing in bad combinations
10. | compute again combinations of size *KL* and count appearances
11. create dataset *X'* by generalizing trajectories depending on remaining trajectory sequences, and parameters *strategy* and *time_strategy*

If a custom tessellation is not provided, the first step of the *ProtectedGeneralization* method is to generate a square tessellation of the geographical area covered by the trajectories in the dataset. A user can define the size of each tile in the tessellation by specifying the parameter *tile_size*. The larger the size of every tile is, the more trajectories and locations will be preserved in the anonymized dataset, but with less precision.

The user can also protect the time dimension, by defining a *time_interval*. A kind of 3D tessellation is built, with a spatial tessellation defined for every time level containing the locations with timestamps between the defined bounds.

If a custom tessellation is not provided, after building it and mapping all the locations to the corresponding tiles, we perform an optimization preprocessing by merging some tiles in the same time level with a number of locations lower than *3k* (see Figure 12). This improves the probability of preserving combinations. Each trajectory is then transformed into a sequence of visits to the resulting tiles.

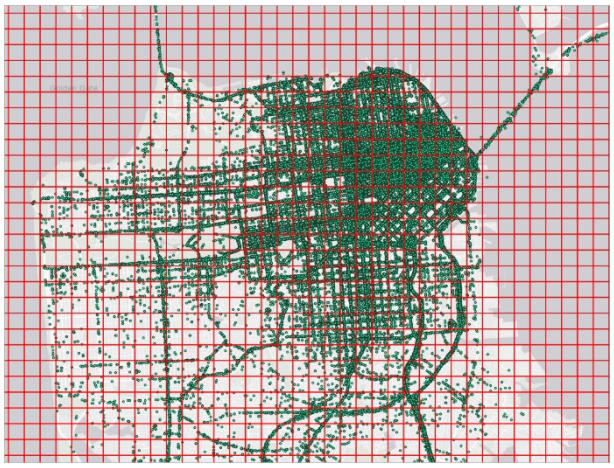

*Figure 11. Squared tessellation*

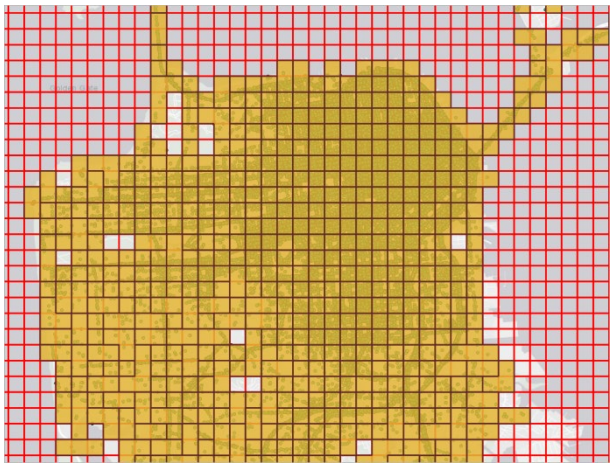

*Figure 12. Squared tessellation with merged tiles*

Now we can generate a generalized version of the dataset *X*. However, to complete the anonymization procedure, we need to ensure that *X'* is a k-anonymous version of *X*. Firstly, we compute all existing combinations of size *KL* and count the occurrences of each combination. *KL* is also a parameter defined by the user and represents the number of locations that an attacker could know from a user *U*. A larger *KL*, more locations removed from the original dataset.

Next, we identify and break the 'bad combinations' (those whose appear less than *k* times) by removing some tiles from the trajectory sequences. For each trajectory sequence, we remove the tiles appearing in the 'bad combinations', starting from the most to the least common. In case of tie, we check the tiles appearing the 'good combinations' to decide which tile remove first.

After processing all the trajectory sequences, we have broken all the 'bad combinations', but we might have created new ones. Therefore, we compute all existing combinations of size *KL* again and count the occurrences of each combination. If we find any new 'bad combinations', we repeat the removal process until no new 'bad combinations' are generated.

Finally, the last step is to create the anonymized dataset *X'* from the remaining trajectory sequences. To do so, a user can choose between computing the average of the locations within each tile (which improves utility) or taking the centroid of the tile as the generalized location, using the parameter *strategy*. Parameter *time_strategy* allows the user to decide whether to preserve the original timestamps or to take the same timestamp for every tile (improving privacy but harming utility). The decision should consider the *time_interval* parameter and the precision of the attacker's knowledge regarding the time when the known locations were visited.

For example, if we assume that an attacker only knows the date and the hour (but not the exact minute) when a location was visited, we can define a 60-minute time interval and preserve the original timestamps because all the locations in the same tile share the same hour and are indistinguishable to the attacker. On the other hand, if the user wants to protect the dataset assuming that an attacker knows the accurate timestamp of a specific location, we should use the same timestamp for all the locations, which will improve privacy but reduce the utility of the dataset.

*Figure 13. Generalized locations generated using a squared tessellation and protected with K=3 and KL=2*

*Figure 14. Paths generated from the previous anonymized locations*

Comparing the previous figures with the figures from section 6.3.3.1 (Simple Generalization) shows that a significant number of locations and trajectories have been removed, losing detail and, therefore, utility. However, we can ensure that all the remaining trajectories are indistinguishable from others given the predefined parameters.

### 6.3.3.3. Microaggregation

Even though *k*-anonymity has usually been enforced via generalization of values, this entails a large information loss for high-dimensional and spread data such trajectories. A more utility-preserving alternative to generalization is microaggregation [2]. Trajectory microaggregation is based on partitioning the data set into disjoint clusters containing each at least *k* similar trajectories. Once the clustering of the data set is complete, the trajectories in each cluster are aggregated by replacing them with the cluster centroid. During the clustering stage, trajectories are grouped minimizing their intra-cluster distance (see section 6.3.1). In the aggregation step, the centroid of the data set is calculated as the trajectory in the data set that minimizes the distance to the rest of trajectories (see section 6.3.2). In this manner, the resulting microaggregated data set minimizes the information loss incurred when enforcing *k*-anonymity.

The microaggregation algorithm works as follows:

| **Microaggregation algorithm** | |
|---|---|
| Requires: | *X*: dataset of trajectories<br>*k*: level of privacy (minimal number of trajectories in a cluster) |

Outputs: *X'*: *k*-anonymized version of *X*

1. calculate the centroid *c* of the complete dataset *X*
2. *X*_remaining = *X*
3. while |*X*_remaning| >= 3*k*:
4. | select the most distant trajectory *r* from the centroid *c*
5. | select the most distant trajectory *s* from the trajectory *r*
6. | generate a cluster formed by the trajectory *r* and the *k*-1 trajectories closest to *r*, and remove them from *X*_remaining
7. | generate a cluster formed by the trajectory *s* and the *k*-1 trajectories closest to *s*, and remove them from *X*_remaining
8. while |*X*_remaning| >= 2*k*:
9. | select the most distant trajectory *r* from the centroid *c*
10. | generate a cluster formed by the trajectory *r* and the *k*-1 trajectories closest to *r*, and remove them from *X*_remaining
11. form a cluster with the remaining trajectories in *X*_remaning
12. create dataset *X'* replacing each original trajectory by the centroid of the cluster it belongs to

The microaggregation algorithm gives a heuristic for *k*-anonymizing a set of trajectories. Given a set *X* of trajectories, the algorithm first calculates the centroid *c* of the dataset *X* (see section 6.3.2). Then, the algorithm constructs a cluster around the most distant trajectory *r* from the centroid *c* with *r* and the k-1 closest trajectories to *r*. After that, it takes the most distant trajectory *s* to *r*, and it creates a cluster with trajectory *s* and the *k*-1 closest trajectories to *s*. Then, all trajectories clustered in steps 4 to 10 are removed from the set of remaining trajectories. The algorithm iterates until fewer than 2*k* trajectories unclustered remain, and, in step 11, it forms a final cluster with them. Finally, in step 12, the algorithm replaces each original trajectory with the centroid of the cluster to which the trajectory belongs (see section 6.3.2 about the centroid calculation of trajectories).

Since the microaggregation algorithm builds clusters by grouping the closest trajectories together, the resulting microaggregated dataset minimizes the information loss incurred when enforcing *k*-anonymity.

Figure 15 shows an example of the anonymization of trajectories by microaggregation. The anonymized trajectory (red) is the result of the aggregation of *k* nearest trajectories (depicted in blue, green and orange) included in a cluster, in this case, of size $k=3$.


*Figure 15. Example of anonymization with the microagreggation method*

The next figures show the general result of the anonymization. The number of original locations and trajectories is preserved although, as shown in Figure 16, some locations are not consistent with the domain of the dataset (*e.g.*, those appearing in the ocean). Nevertheless, the number of these 'impossible' locations is very small compared to the size of the dataset. If we need to generate real locations, we can use an alternative aggregation method as described in section 6.3.2.2.

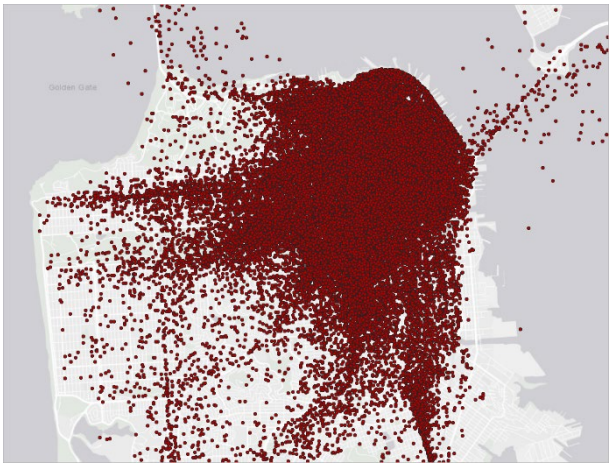

*Figure 16. Anonymized locations using microaggregation with k=3*

*Figure 17. Paths generated from the previous locations*

### 6.3.3.4. Time Partitioned Microaggregation (TimePartMicroaggregation)

As stated in the previous section, microaggregation is a utility-preserving method to enforce *k*-anonymity in high-dimensional and spread data such trajectories. The partition and aggregation steps produce some information loss. The goal of microaggregation is to minimize the information loss according to some utility metric. For this reason, microaggregation is considered a good approach to anonymize trajectories when the utility of original data should be preserved as much as possible. On the other hand, the runtime of the microaggregation algorithm can be unfeasible for large datasets and small *k* values. This is because the computational cost of the microaggregation algorithm scales $O\left(n^2/_k\right)$, where $n$ is the number of trajectories in the dataset and *k* is the minimum number of trajectories sharing the same combination (that is, the desired level of privacy).

To solve this issue in large datasets, we have based on microaggregation to propose the method Time Partitioned Microaggregation of trajectories (TimePartMicroaggregation) with the following features:

- Since the runtime lowers as $\boldsymbol{n}$ decreases, we propose to first partition the original dataset into smaller datasets based on the time dimension of the trajectories.
- Each partition contains the trajectories in a given time interval. In this way, the trajectories contained in the resulting small datasets are similar in time and can be clustered based only on their spatial distance.
- This method results in a feasible runtime for large datasets at the cost of a higher (but assumable) information loss compared to the microaggregation method. This is because the microaggregation method searches for similar trajectories through the entire dataset to group clusters while the TimePartMicroaggregation method searches only in the smaller partitions of the dataset.

The Time Partitioned Microaggregation algorithm works as follows:

**Time Partitioned Microaggregation (TimePartMicroaggregation) algorithm**

Requires: *X*: dataset of trajectories
*K*: level of privacy (minimal cluster size of the microaggregation)
*Interval*: time interval in each partitioned dataset
Outputs: *X'*: *k*-anonymized version of *X*

1. *X*_sorted = sort(*X*) in function of the mean of timestamps of each trajectory in *X*
2. datasets = ∅
3. while size *X*_sorted >= *k*:
4. partition = ∅
5. add to partition all trajectories inside the time interval and remove them from *X*_sorted
6. while size partition < *k*:
7. add to partition the first trajectory of *X*_sorted and remove it from *X*_sorted
8. add partition to datasets


Funded by the European Union

9. add to the last partition the remaining trajectories in *X*_sorted
10. microaggregate each partition in datasets (as explained in section 6.3.3.3)

The input of the algorithm consists of the dataset of trajectories (X), the level of privacy (k), that is the minimal number of trajectories in a cluster, and the desired time (interval) in which the dataset X will be partitioned. First, the dataset X is sorted according to the mean of the timestamps of each trajectory. Then, as long as there are at least k trajectories left to include in a partition, the algorithm creates a partition with all trajectories in X that are included in the desired time interval (steps 4 through 8). Each partition should have at least k trajectories to create at least one cluster of size k during the microaggregation process. If there are less than k trajectories in the time interval, trajectories from the next interval are added to complete the partition (steps 6 and 7). The remaining trajectories are added to a final partition (step 9). In this way, the algorithm outputs a set of datasets, each one with at least k time-similar trajectories. Finally, in step 10, each partition is microaggregated as described in section 6.3.3.3.

Since the trajectories contained in each partition dataset are similar in the time dimension, we can use only the spatial distance to microaggregate trajectories during the clustering process. This makes it easier to compute the distance between trajectories.

If the input trajectory dataset is too large to be anonymized using the microaggregation method (due to an unfeasible runtime), the TimePartMicroaggregation method should be considered.

*Figure 18. Anonymized locations using TimePartMicroaggregation with k=3 and dividing the dataset with an interval=1 hour*

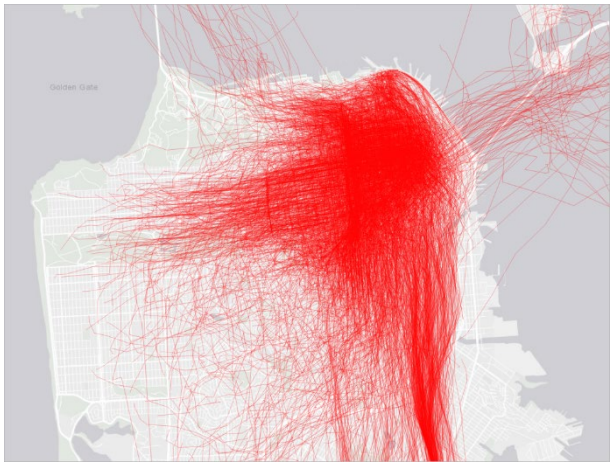

*Figure 19. Paths generated from the previous locations*

Comparing the results with those obtained with the Microaggregation method, a loss of information and utility can be observed.

### 6.3.3.5. Swap All Locations

This method is based on the *ReachLocation* mechanism proposed by [1]. Protection is achieved by permuting the locations in trajectories among other trajectories. The method works as follows: first, a cluster is created around a randomly selected location based on some spatial and temporal threshold parameters provided by the user. If the cluster does not include at least $k$ locations from at least $k$ different trajectories, the thresholds are increased until we obtain the required number of locations or the thresholds reach user-defined maximum values. If a valid cluster is built, the locations are swapped among the $k$ trajectories (by changing their trajectory IDs) and marked as *swapped*. If no valid cluster can be found around a location, it is removed. This process continues until no more "unswapped" locations appear in the data set.

This method provides great utility since locations in the resulting anonymized trajectories are true, fully accurate original locations. No fake, generalized or perturbed locations are given in the anonymized data set of trajectories. Besides that, the flow and directions of the original trajectories are well preserved. However, this mechanism does not offer a formal guaranty of privacy. If a whole trajectory is unique, the user could be identified.

To mitigate this problem, we have developed a simple trajectory anonymization mechanism like that described in section 6.3.3.2. After swapping all the locations in the dataset, we build a square grid and convert each trajectory into a sequence of tiles. We then analyse the generated sequences to find combinations of 2 consecutive tiles that occur less than KL times in the anonymized dataset. If such a combination is found, the locations within one of the tiles are removed.

Figure 20 shows an example of this anonymization mechanism. The yellow circles are the original locations. In green are depicted all the locations from the original dataset that meet the spatial and temporal requirements to be swapped with one of the original locations to be anonymized. Finally, in red we can see the locations that have been selected to build the anonymized trajectory.


Funded by the European Union

*Figure 20. Example of anonymization with the SwapAllLocations method*

Figure 21 shows the resulting anonymized locations using SwapAllLocations. All these locations are real, and, at first glance, it looks like that only some locations (those that are too unique in terms of space and/or time) have been removed. However, Figure 22 shows that trajectories have also been distorted, although they preserve the flow and direction of the original ones.

*Figure 21. Locations in the anonymized dataset using SwapLocations, with a spatial threshold between 100 and 500 m and a temporal threshold between 1 and 2 minutes.*

*Figure 22. Paths generated from the anonymized locations*

### 6.3.3.6. SwapMob

SwapMob, proposed by J. Salas, D. Megías and V. Torra [4] is a perturbative anonymization method based on swapping segments of trajectories with other trajectories. When two locations in two different trajectories are close enough (when they cross each other), according to some threshold of proximity and time set by the user, the two remaining subtrajectories are swapped between the two original trajectories (that is, the ID of the previous locations of each trajectory are swapped). Changing pseudonyms (IDs) is equivalent to swapping the partial trajectories. If a trajectory does not cross any other one, and therefore, no subtrajectory is swapped with other trajectories, it is removed.

Hence, the relation between data subjects and their data is obfuscated while keeping a precise aggregated data, such as the number of users and their directions on any given zone at a specific time, the locations that have been visited by different anonymous users or the average length of trajectories.

Nevertheless, this comes at the cost of modifying the trajectories and losing individual trajectory mining utility.

In Figure 23, we can see an example of the anonymization method. The anonymized trajectory (dark red) is made up of subsegments of several original trajectories. The first subsegment comes from the green trajectory and the following subsegments come from the pink, orange, yellow, purple, and blue trajectories.

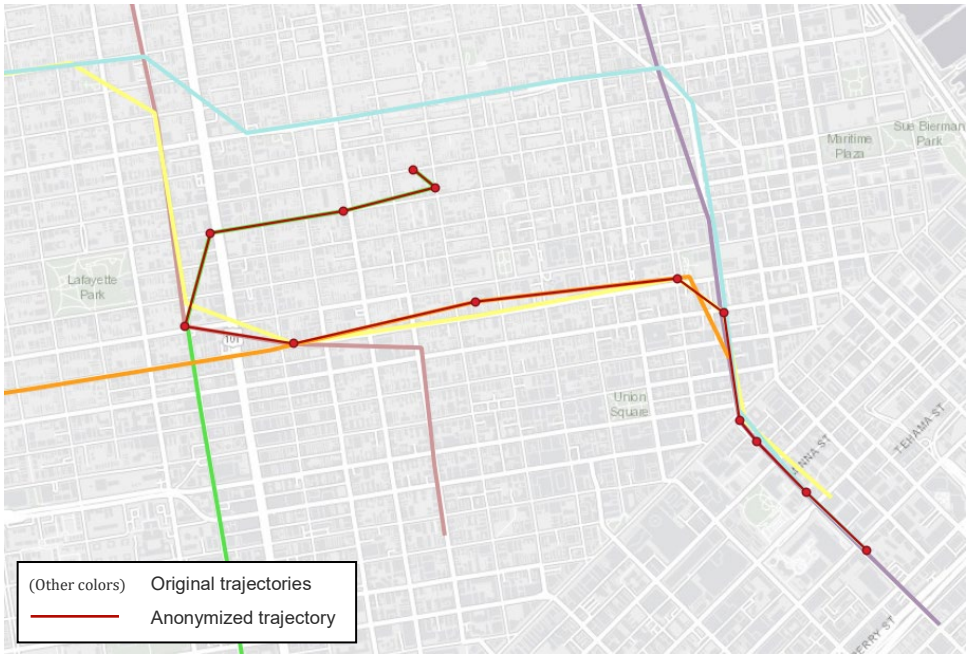


*Figure 23. Example of anonymization with the SwapMob method*

Looking at the figures below, it looks like that both locations and trajectories are very well preserved. However, the individual trajectories are completely different from the original ones.

*Figure 24. Anonymized locations using the SwapMob method*

*Figure 25. Generated paths from previous locations*

### 6.3.4. Utility and privacy metrics

To quantify the quality of the anonymized data, we estimate the utility of the resulting dataset by measuring the information loss caused by replacing the original values with their masked versions. On the other hand, we quantify the practical privacy obtained in the anonymized dataset by measuring the disclosure risk of the resulting data. In addition, we also measure the runtime required during the anonymization process to estimate the computational feasibility of the included methods. In the next sections, we present several utility and privacy metrics to quantify the quality of anonymized datasets.

#### 6.3.4.1. Information loss via RMSE

The utility of the anonymized dataset is measured as the information loss resulting from the anonymization of trajectories. Information loss measures the differences between the original and anonymized datasets. To do this, we use the well-known Root Mean of Square Errors (RMSE). For a given anonymized dataset *X'*, RMSE is defined as the sum of squares of distances between original trajectories in *X* and their masked versions in the anonymized dataset *X'*. We measure the information loss as the RMSE calculated as follows:

$$RMSE = \frac{1}{n}\sqrt{\sum_{i=1}^{n} dist(T_i, T_i')^2}$$

where $n$ is the number of trajectories in the dataset, $T_i$ is a trajectory in the original dataset and $T_i'$ is the anonymized version of the trajectory $T_i$. To calculate the distance between trajectories ($dist$), we use the spatio-temporal distance described in section 6.3.1.2.

Notice that with a high RMSE, that is, a high information loss, a lot of data are damaged, leading to a decrease of the utility of the anonymized dataset.

### 6.3.4.2. Information loss via normalized RMSE

When it is possible to calculate the maximum distance between trajectories in the dataset *X*, the normalized version of RSME can be calculated in order to obtain the information loss value relative to the maximum possible information loss in the dataset. The normalized RMSE is calculated by dividing the distance between two trajectories by the maximum distance between any two trajectories in the dataset. The resulting distance is in the range [0,1] and it is calculated as follows:

$$normalized\ RMSE = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\left(\frac{dist(T_i, T_i')}{max\ distance}\right)^2}$$

where $n$ is the number of trajectories in the dataset, $T_i$ is a trajectory in the original dataset and $T_i'$ is the anonymized version of trajectory $T_i$. The spatio-temporal distance described in section 6.3.1.2 is used to calculate the distance ($dist$) between trajectories and the $max\ distance$ is the previously calculated maximum distance between trajectories in the original dataset *X*.

### 6.3.4.3. Propensity score

In statistics, propensity score matching is a method commonly used in inference studies to compare outcomes among subjects that received a treatment, policy, or other intervention versus those that do not. To use Propensity score as a measure of utility, we need to train a model to estimate group membership between original and transformed datasets (*e.g.*, anonymized or synthetic data). If original and anonymized datasets cannot be distinguished (small distinguishability score), then the utility of the anonymous data is high.

Unlike RMSE metric, the propensity score is a data-agnostic utility metric that does not require a specific distance calculation for the specific data type. In addition, the propensity score is sensitive to anonymization via suppression of records because it results in an unbalanced input data for the model. On the other hand, the choice of the model for the propensity score calculation might influence the utility estimation of the anonymized data. With this in mind, logistic regression models are commonly used.

The computation of the propensity score in mobility data works as follows:

| **Propensity score calculation algorithm** | |
|---|---|
| Requires: | *X*: dataset of original trajectories<br>*X'*: dataset of anonymized trajectories |
| Outputs: | Propensity score |

1. Preprocess original and anonymized datasets so that they are amenable for machine learning model training as follows:
   Use of tessellation to discretize the domain
   0-pad sequences to make them all the same length
   (optionally) normalize data in range (1,1) or (0,1)
2. Merge original and anonymized datasets and add a binary label *A* with value 1 for anonymized records and 0 for original records, shuffle the records in the merged dataset
3. Train a ML model to regress the label *A* based on the rest of attributes (positions $x_{i,j}$) and call it $\hat{A}$
4. Let the propensity score $\hat{\boldsymbol{p}}_i$ of record $i$ be the probability $p(A = 1|x_{i,j})$
5. The propensity score is then: $$U = \frac{4}{n}\sum_{i=1}^{n}\left(\hat{p}_i - \frac{1}{2}\right)^2$$

---

To calculate the propensity score, first the original and anonymized datasets are pre-processed to allow for training a model. To do this, in step 1, tessellation is applied to generalize locations to obtain tractable trajectory sequences. The trajectory sequences are padded with "0" at the left to equalize their lengths to the length of the largest sequence. Additionally, the data is normalized for better model performance. Then, in step 2, the original and anonymized datasets are merged, labelling each anonymized trajectory sequence with a label A with value *1*, and for original trajectory sequences with value *0*. Then, the resulting merged dataset is shuffled. Once the input data is pre-processed, the machine learning model is trained to estimate the probability that a sequence belongs to the class *A=1*. Finally, the utility is estimated as the propensity score mean-squared error described in equation of step 5, where *n* is the total number of sequences in the input dataset and $\hat{p}_i$ is the probability that a sequence *i* belongs to the class *A=1*.

The resulting propensity score is in the range $[0,1]$ where the higher the score, the higher the information loss (the lower the utility of data).

#### 6.3.4.4. Disclosure risk via record linkage

To measure the practical privacy resulting of the anonymization process, we measure the disclosure risk of the resulting dataset. The disclosure risk estimates the percentage of trajectories of the original dataset that can be correctly matched to the trajectories in the anonymized dataset, that is, the percentage of correct Record Linkages (RL). For an anonymized dataset *X* of trajectories, the record linkage is calculated as follows:

$$RL = 100 \times \frac{\sum_{x_j \in X} \Pr(x_j')}{n}$$

where $n$ is the number of original trajectories and $\Pr(x'_j)$ is the record linkage probability for an anonymized trajectory calculated as:

$$\Pr(x'_j) = \begin{cases} 0 & if \;\; x_j \;\; not \in G \\ \dfrac{1}{|G|} & if \;\; x_j \in G \end{cases}$$

where $G$ is the set of original trajectories that are at minimum distance from $x'_j$. The spatio-temporal distance described in section 6.3.1.2 is used to calculate the minimum distance between trajectories. The lower RL, the lower the probability of identity disclosure and the higher the privacy of the anonymized dataset.

The computational cost of the record linkage calculation scales $O(n^2)$.This is because the possible matching of each original trajectory in the anonymized dataset is searched through the entire original dataset. This search that can be unfeasible from the point of view of runtime for large datasets. To deal this problem, we propose to search possible record linkage matching in a subset of trajectories of size $m$ instead of the entire dataset. This faster record linkage works as follows:

**Fast Record linkage calculation algorithm**

Requires: *X*: dataset of original trajectories
*X'*: dataset of anonymized trajectories
*window*: the desired size of the subset of anonymized trajectories
Outputs: The Record linkage percentage

1. calculate the centroid *c* of the complete dataset *X* (see section 6.3.2)
2. calculate the distance to *c* of all trajectories in datasets *X* and *X'*
3. sort trajectories in *X* in function of the distance to *c*
4. for each trajectory $x'_j$ in *X'*:
5. take the *window* number of trajectories from *X* that have the most similar distance to the centroid *c* that have the trajectory $x'_j$
6. calculate the $\Pr(x'_j)$, as described previously in this section, for the *window* subset of original trajectories
7. Calculate RL as described previously in this section

As the dataset *X* is sorted in function of the distance to the centroid *c*, we can take the subset *window* with a computational cost of $O(\log n)$. Then, the computational cost of the fast record linkage scales $O(n'm)$, where $n'$ is the number of anonymized trajectories and $m$ is the size of *window*, that is, the desired number of trajectories in the subset. This record linkage can estimate the disclosure risk in a feasible runtime, depending on the size of the *window* parameter, at the cost of lose some linkages.

The higher the *window* size the higher the accuracy of the record linkage estimation and the higher the runtime.

### 6.3.4.5. Comparison of the anonymization methods

As described in the previous section, every anonymization method preserves different utility metrics. To demonstrate the performance and accuracy of the anonymization methods, we have anonymized two real-world datasets with the anonymization methods described in section 6.3.3. Then, we measured and compared the resulting utility and privacy metrics described in section 6.3.4. The experiments were run on an Intel i5-8250U with 16 GB of RAM. In the next sections, we summarize some utility and privacy results.

#### Evaluation Data

The first dataset we have used was built from the mobility data of taxi cabs in San Francisco, USA, provided by the Exploratorium Museum within the Cabspotting project[2]. The data set contains the trajectories of approximately 500 taxi cabs in the San Francisco Bay Area recorded during May 2008. The data capture the usual features of realistic trajectories (*i.e.*, short trajectories in areas with a dense population). Each record contains the GPS coordinates and absolute times of all trajectory points. To obtain a large and dense dataset, we joined trajectories recorded in a day. We considered only the trajectories that correspond to cabs that were occupied by a customer. This resulted in realistic trajectories with meaningful and precise origins, paths, and destinations, rather than seemingly random routes of cabs wandering or waiting for customers. We omitted trajectories with fewer than 5 locations and trajectories with some wrong locations (detected by computing the speed between two consecutive locations). The resulting dataset contains 10,282 trajectories and 155,690 locations, with a mean of 15 locations per trajectory.

The second dataset has been built from logs provided by Hove of real queries sent to the Navitia travel planner[3]. In this case, the records correspond to trajectories that the planner has created from an origin and a destination location that the user introduced. The dataset contains 192,855 trajectories including 759,123 locations which is useful to test the performance of anonymization and metric methods in large datasets. As each trajectory represents a trip, many of the locations are common, such as train, bus stations or tourist locations. For this, this dataset has a mean of 4 points per trajectory, which is considerably less than the taxi cabs dataset.

#### Evaluation of Microaggregation methods

In this section, we compare the utility, privacy and runtime obtained by microaggregation methods. We applied the original microaggregation algorithm, described in section 6.3.3.3, to anonymize the evaluation datasets. As stated above, the fact that Microaggregation exhaustively searches the closest trajectories to be grouped, we expect that this method will obtain the best utility (at the cost of long runtime). For this, we consider microaggregation results as baseline for the comparatives. Then, we have anonymized the same datasets with the proposed Time Partitioned Microaggregation algorithm, described in section 6.3.3.4, and we compared the metrics described in section 6.3.4 on the resulting datasets. Taking into consideration the cardinalities of evaluation datasets, the *k*-anonymity levels for all methods (parameter *k*) have been set between 3 and 100.

[2] http://www.exploratorium.edu/id/cab.html
[3] https://navitia.io/

For the TimePartMicroaggregation algorithm (tpm in figures), the *interval* parameter has been ranged between 60 and 3600 seconds which corresponds from 5 min. to 1 hour.

We first evaluate to what extent each microaggregation algorithm preserves the data utility for a certain privacy level. As stated, the utility of an anonymized output is evaluated in terms of information loss, that is, the differences between the original and anonymized data. To quantify the information loss, we measured the RSME and propensity score of the resulting data as described in section 6.3.4. Figure 26 and Figure 27 depict respectively the RSME and propensity score values obtained in the two datasets (on the left the taxi cabs dataset, on the right the Navitia dataset) for the different parameterizations of *k* for microaggregation and *k* and *interval* for timePartMicroaggregation.

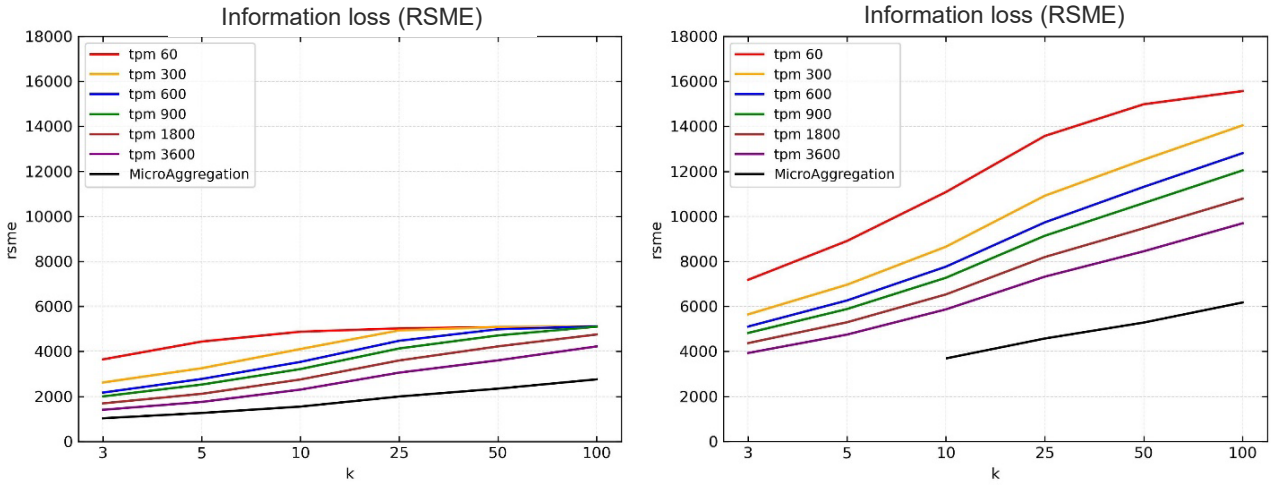


*Figure 26. Taxi cabs (left) and Navitia (right) datasets show the evolution of RSME in the timePartMicroaggregation methods for different k and interval values, and in the microaggregation method for different k values.*

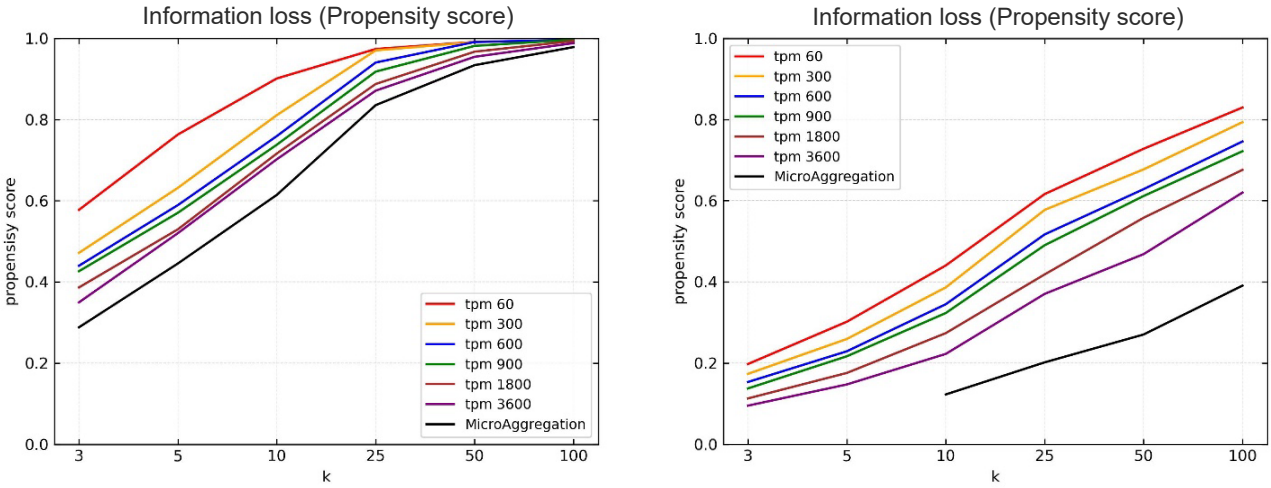


*Figure 27. Taxi cabs (left) and Navitia (right) datasets show the evolution of propensity score in the timePartMicroaggregation methods for different k and interval values, and in the microaggregation method for different k values*

Regarding the evolution of RSME shown in Figure 26, we observe for the two datasets that the microaggregation algorithm (black line) reduces the information loss compared with the timePartMicroaggregation method in all *k* and *interval* values. This was expected because microaggregation exhaustively searches through the entire dataset. We can also observe for all methods that the higher the *k* value the higher the information loss. This is because high *k* values imply more trajectories in each cluster which are masked in same trajectory in the aggregation step. In the case of interval values for the timePartMicroaggregation, the lower the time *interval* the higher the information loss. This is because the time *interval* delimits the subset of trajectories where the algorithm searches the closest ones to be grouped and aggregated, leading to higher information loss. The evolution of the propensity score has a correlated behavior with respect to RSME, so that, high values of *k* and low values of *interval* parameters entail high information loss. As stated in section 6.3.4.3, the difference here is that, while the RSME requires the calculation of distances between trajectories, the propensity score does not.

Notice that for *k*=3 and *k*=5, the microaggregation algorithm is unable to anonymize the Navitia dataset, this is due to the size of the data and the excessive runtime needed to anonymize it. For this reason, as stated in section 6.3.3.4, we propose the timePartMicroaggregation method to solve the high computational cost of microaggregation algorithm in large datasets. Figure 28 shows the evolution of the necessary runtime to anonymize the evaluation datasets comparing microaggregation and timepartitonMicroaggregation algorithms.

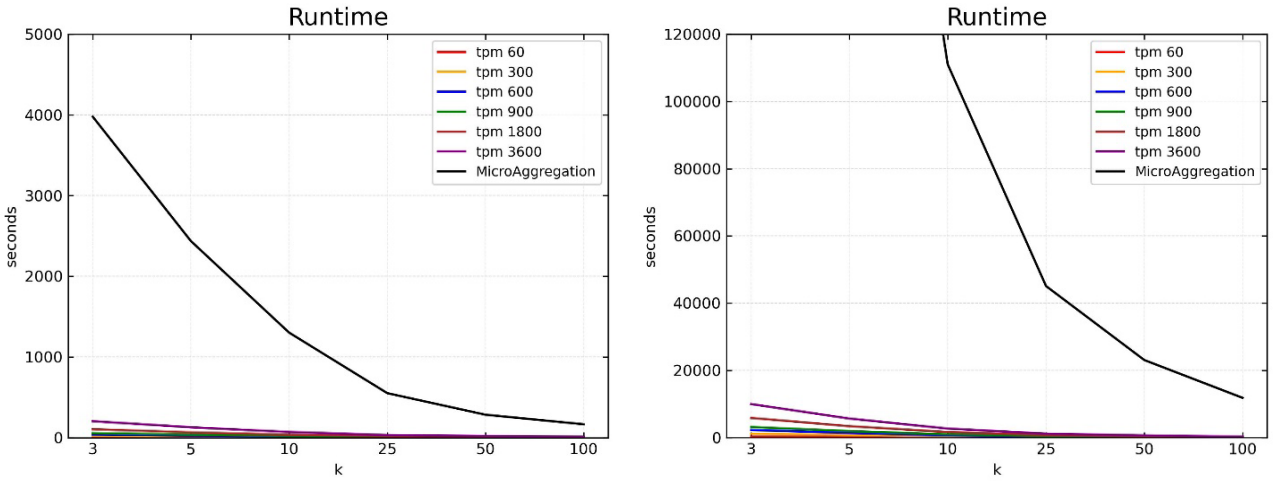


*Figure 28. Taxi cabs (left) and Navitia (right) datasets show the necessary runtime in seconds to execute the timePartMicroaggregation method for different k and interval values, and the microaggregation method for different k values*

Regarding the runtime comparative shown in Figure 28 we can see that the timePartitionMicroaggregation algorithm (tpm in figure) can anonymize the two datasets in a feasible amount of time. We observe that the lower the interval the lower the runtime. This is because lower intervals imply splitting the dataset into smaller subsets in which the closest trajectories are searched, requiring less time in the search. In addition, the runtime is decreased when the *k* level is increased.

This is consistent with what was explained in section 6.3.3.4 about the computation cost of the microaggregation algorithm, that is $O\left(n^2/_k\right)$, where higher *k* values decrease the computational cost. This is also the reason why the microaggregation algorithm is not able to anonymize large datasets for low values of *k*. We can conclude that, when the input trajectory data are too large to be anonymized by means of the microaggregation method, the timePartitionMicroaggregation method should be considered.

## Evaluation of anonymization methods

The second analysis aims to compare all anonymization algorithms described in section 6.3.3.3. The *k*-anonymity level for methods which require it (parameter *k* for microaggregation and timePartitionMicroaggregation) has been set between 3 and 100. For the timepartitionMicroaggregation algorithm, we take a mean value of *interval*=900. Figure 29 and Figure 30 show the information loss measured, respectively, with the RSME and propensity score obtained from the anonymized datasets applying microaggregation, timePartitionMicroaggregation, Simple generalization, SwapMob and SwapAllLocations algorithms described in section 6.3.3.3.

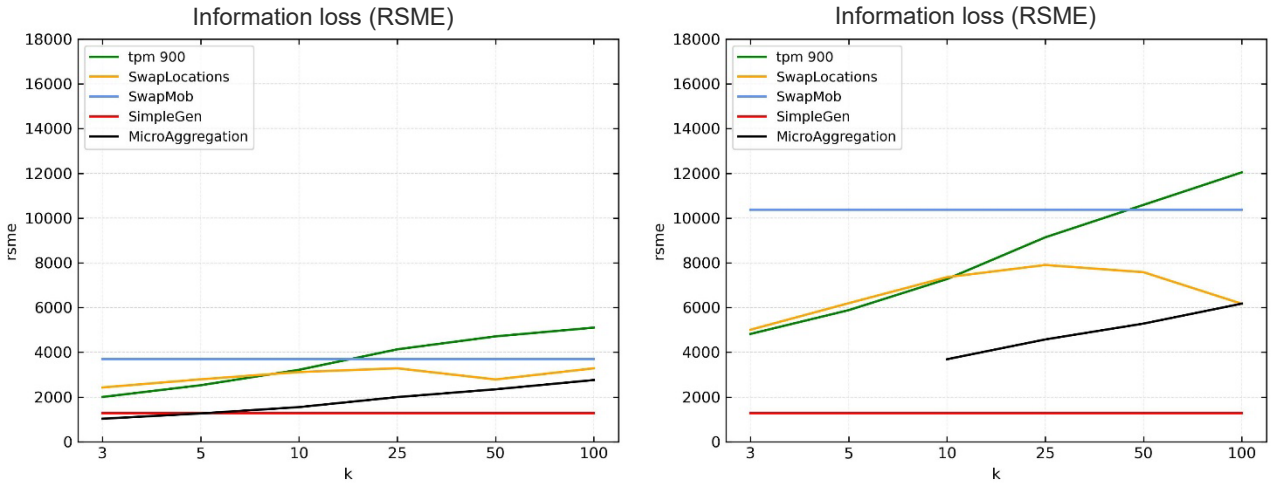


*Figure 29. Taxi cabs (left) and Navitia (right) datasets show the evolution of RSME in the timePartMicroaggregation, SwapAllLocations, SwapMob, SimpleGen and Microaggregation methods for different k values.*

Information loss (Propensity score) Information loss (Propensity score)

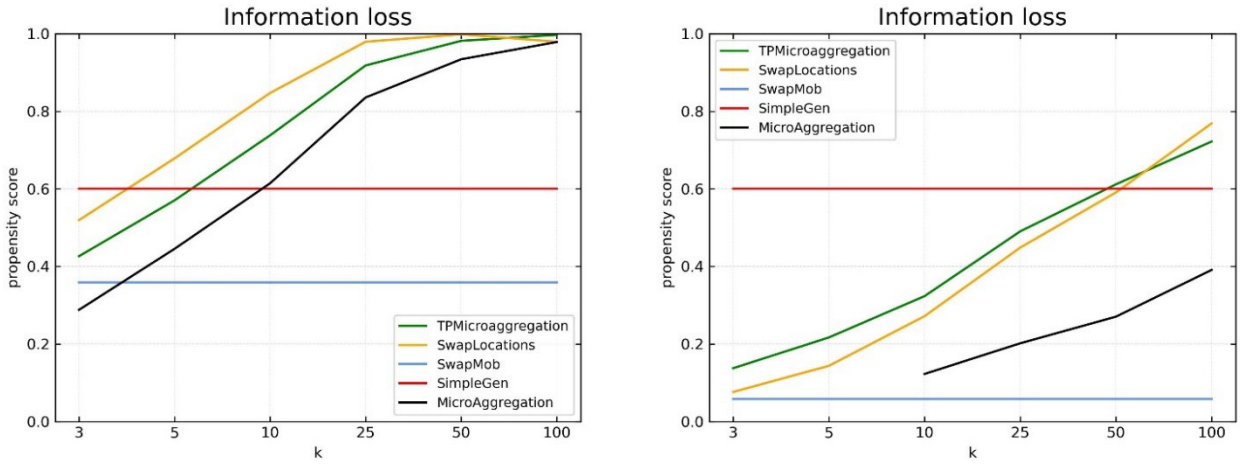


*Figure 30. Taxi cabs (left) and Navitia (right) datasets show the evolution of propensity score in the timePartMicroaggregation, SwapAllLocations, SwapMob, SimpleGen and Microaggregation methods for different k values.*

Regarding microaggregation, timePartitionMicroaggregation and SwapAllLocations we observe that microaggregation obtains the best utility for all *k* values. Comparing timePartitionMicroaggregation and swapLocations we observe that the later obtains best results for very high k values. The swapMob and simpleGen methods results are immutable for the *k* values, this because they do not require a *k* parameter (which indicates the privacy level). The SimpleGen method obtains the best utility when it is measured calculating the RSME while the SwapMob method obtains the lower information loss if it is measured the propensity score. This is coherent with the fact that propensity score does not require a distance calculation and it is sensitive to the suppression of trajectories.

The fact that some methods do not require the *k* privacy level parameter leads to the necessity of measure and compare the practical privacy of anonymization methods. To do this, we measure the disclosure risk of the anonymized datasets. Figure 31 show the disclosure risk comparative of all previously described methods by measuring the percentage of correct record linkages between trajectories in the anonymized and original datasets, as described in section 6.3.4.4.

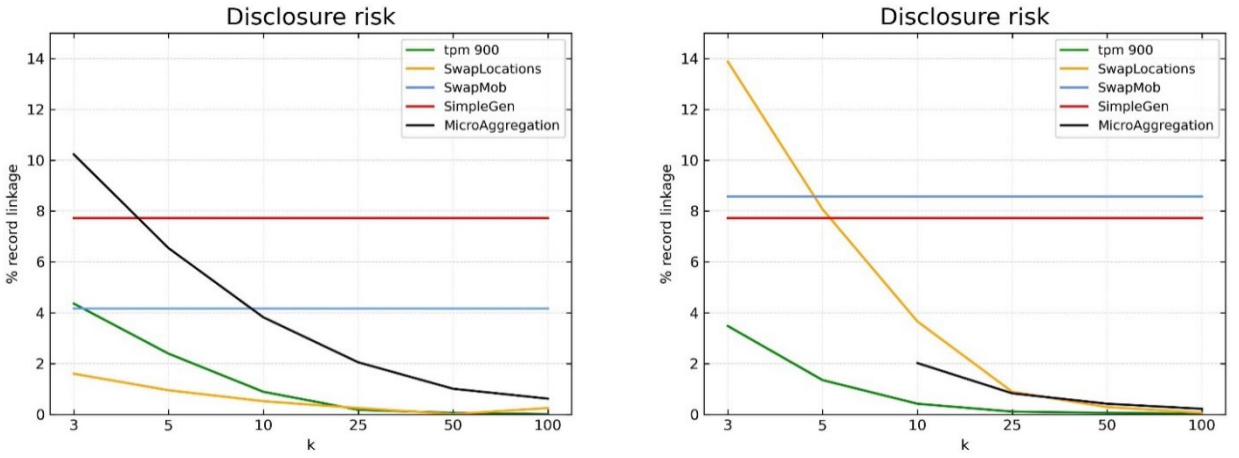


*Figure 31. Taxi cabs (left) and Navitia (right) datasets show the evolution of disclosure risk in the timePartMicroaggregation, SwapAllLocations, SwapMob, SimpleGen and Microaggregation methods for different k values*

Regarding the disclosure risk, as expected, we observe that, for microaggregation, timePartitionedMicroaggregation and swapAllLocations methods, the higher the *k* privacy level the lower the disclosure risk which is inversely correlated with the information loss studied above. Regarding swapMob and simpleGen methods, we observe that they incur in high disclosure risk in almost *k* privacy level of the other methods, only microaggregation in the taxi cabs dataset and swapAllLocations in the Navitia dataset have higher disclosure risk for very low values of *k* privacy level. This is coherent with the fact that swapMob and simpleGen methods obtains high utility but they do not guarantee a privacy level in the anonymized datasets.

Finally, in Figure 32 we compare the necessary runtime to anonymize the evaluation datasets. We observe that the smaller dataset (taxi cabs) can be anonymized by all methods and k values in a feasible amount of time. However, the larger dataset (Navitia dataset) requires almost 12 hours to be anonymized by the swapAllLocations method and microaggregation requires an unfeasible amount of time to anonymize the dataset with low *k* privacy values. The rest of methods, timePartitionedMicroaggregation, swapMob and simpleGen can anonymize the larger dataset rapidly.

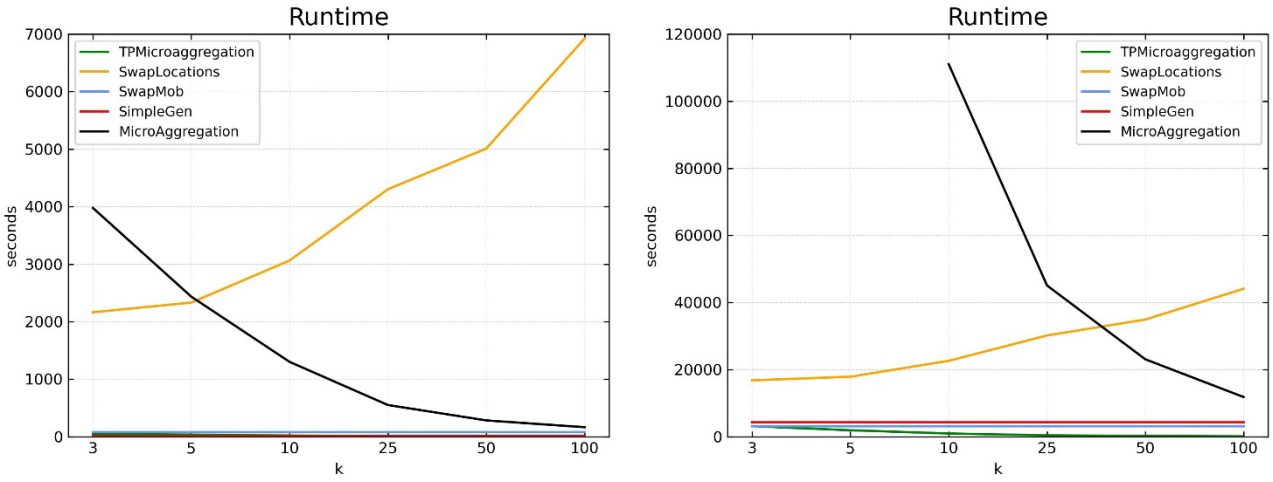


*Figure 32. Shows the evolution of the necessary runtime to anonymize the taxi cabs (left) and Navitia (right) datasets with the timePartMicroaggregation, SwapAllLocations, SwapMob, SimpleGen and Microaggregation methods for different k values*

Table 1 and Table 2 show, respectively, the percentage of trajectories that have been removed during the anonymization process of the taxi cabs dataset and Navitia dataset. Note that the anonymized dataset via microaggregation and timePartMicroaggregation methods maintain the same number of trajectories that the original dataset.

This is due to the aggregation step where each trajectory is substituted by the centroid of the cluster where it belongs, resulting in the same number of trajectories. Simple generalization removes very few trajectories (0.1%) and the other methods, swapAllLocations, protectedGen, and swapMob need to suppress the trajectories that could disclose privacy, specially, for high values of *k*.

*Table 1. Percentage of trajectories removed in the anonymized version of the taxi cabs dataset for each anonymization method and k values*

| k | Microaggregation | TimePart Microaggregation | Swap AllLocations | Protected Generalization | Simple generalization | SwapMob |
|---|---|---|---|---|---|---|
| **3** | 0.00 | 0.00 | 1.46 | 19.65 | 0.1 | 18.74 |
| **5** | 0.00 | 0.00 | 3.40 | 37.21 | 0.1 | 18.74 |
| **10** | 0.00 | 0.00 | 12.82 | 59.02 | 0.1 | 18.74 |
| **25** | 0.00 | 0.00 | 42.99 | 79.33 | 0.1 | 18.74 |
| **50** | 0.00 | 0.00 | 97.64 | 83.33 | 0.1 | 18.74 |
| **100** | 0.00 | 0.00 | 97.70 | 99.43 | 0.1 | 18.74 |

*Table 2. Percentage of trajectories removed in the anonymized version of the Navitia dataset for each anonymization method and k values*

| k | Microaggregation | TimePart Microaggregation | Swap AllLocations | Simple generalization | SwapMob |
|---|---|---|---|---|---|
| **3** | 0.00 | 0.00 | 16.23 | 0.2 | 10.68 |
| **5** | 0.00 | 0.00 | 24.97 | 0.2 | 10.68 |
| **10** | 0.00 | 0.00 | 37.23 | 0.2 | 10.68 |
| **25** | 0.00 | 0.00 | 53.82 | 0.2 | 10.68 |
| **50** | 0.00 | 0.00 | 66.91 | 0.2 | 10.68 |
| **100** | 0.00 | 0.00 | 82.64 | 0.2 | 10.68 |

Considering the results obtained in the comparative study, we can conclude that we should use microaggregation or swapLocations to retain high utility of the anonymized dataset for high *k* privacy levels. In the case of large datasets or low k values, we should use timePartitionMicroaggregation. SwapMob and simpleGen methods can be only considered when we need to anonymize in a very fast way and the *k* privacy level is unknown.

### 6.3.5. Privacy-preserving analysis methods

#### 6.3.5.1. QuadTreeHeatMap

For analysing the locations density without compromising privacy, a k-anonymous heat map generation method has been implemented. This consists of dividing the map into a set of size-variant rectangular sectors containing at least *k* locations. The locations density for each sector represents the “heat” of that area.

The approach is based on quadtree spatial decomposition, which recursively divides the 2D space into groups of four rectangular sectors. Initially, the quadtree consists of a single empty rectangular sector. As locations from the dataset are inserted, if a sector contains enough locations, it is divided into four equally sized subsectors. In this way, denser areas will have smaller sectors. To generate the heatmap, a top-bottom processing of the quadtree sectors is performed to obtain a set of sectors containing at least *k* locations (k-anonymous). To satisfy this requirement in a straightforward way, if a sector (equivalent to a quadtree node) has a child with less than *k* locations, all the child sectors are ignored, and only the parent sector is added to the heatmap. Alternatively, we can try to merge the child sectors to get new sectors with at least *k* locations. Furthermore, a new quadtree can be created using the locations from the merged sectors, which may result in fine-grained (smaller) k-anonymous sectors. For each sector of the resulting heatmap, the number of locations it contains is divided by its area, obtaining its location density.

In Figure 35 the heatmap obtained for the city of San Francisco is shown. Each sector is painted according to its density value, showing that most of the locations are gathered in the city center and the southern airport.


Funded by the European Union

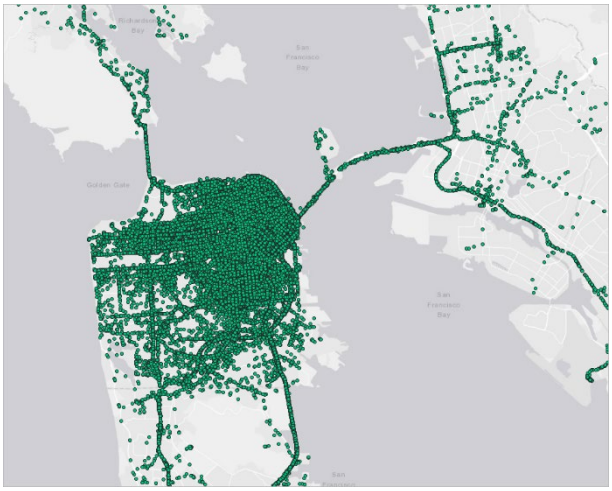

Figure 33. Original locations

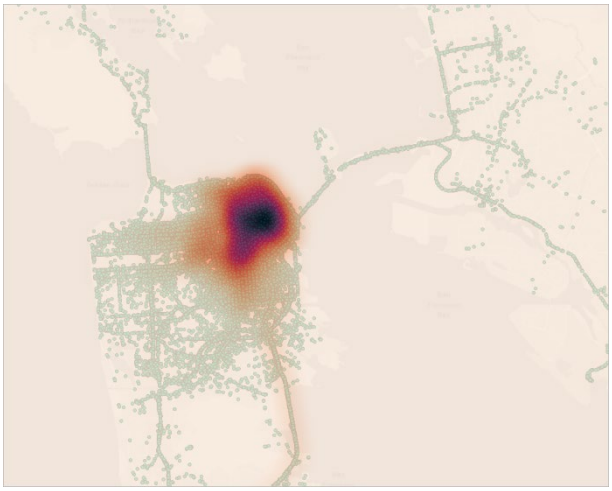

Figure 34. Heatmap computed from the original locations

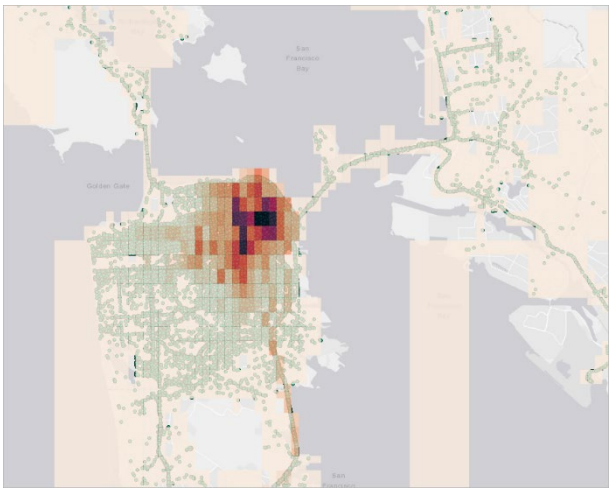

Figure 35. Privacy-preserving heatmap

## 6.4. Command line user guide

### 6.4.1. Anonymization methods

The parameter values to configure the anonymization methods are provided to the application using a JSON file:

```
$ python -m mdl_anonymizer anonymize -f parameter_file.json
```

There are some common parameters to all the anonymization methods:

| Parameter | Description |
|---|---|
| **method** | `Type: string`<br><br>The name of the anonymization method to be executed. Must be one of the included in the main configuration file '`config.json`'. For the moment, must be one of the following:<br><br>• SimpleGeneralization<br>• ProtectedGeneralization<br>• Microaggregation<br>• TimePartMicroaggregation<br>• SwapAllLocations<br>• SwapMob |
| **input_file** | `Type: string`<br><br>The dataset to be anonymized |
| **output_folder** | `Type: string`<br><br>Folder to save the generated output datasets |
| **main_output_file** | `Type: string, optional`<br><br>The name of the anonymized dataset |
| **params** | `Type: JSON object, optional`<br><br>Specific parameters of the corresponding anonymized method |

#### 6.4.1.1. Specific configuration parameters

Each of the anonymization methods has some specific parameters that can be added to the parameters file:

**SimpleGeneralization**

| Parameter | Description |
|---|---|
| **tiles_filename** | `Type: str, optional`<br><br>Tiles files for tessellation (geojson or shapefile)<br><br>`Default: None` |
| **tile_size** | `Type: int, optional` |



Funded by the European Union

| | |
|---|---|
| | If a tiles file is not provided, a squared tessellation of size 'tile_size' is generated (in meters)<br><br>default: 500 |
| **overlapping_strategy** | Type: str, enumerate ("all", "one"), optional<br><br>If a several locations of the same trajectory end up in the same tile, keep all ("all") or take just one and compute the average timestamp ("one")<br><br>Default: "all" |

JSON config file example for the Simple Generalization method:

```
{
  "method": "SimpleGeneralization",
  "input_file": "examples/data/cabs_dataset_20080608_0700_0715.csv",
  "output_folder": "examples/output",
  "main_output_file": "cabs_dataset_20080608_0700_0715_anonymized_simple.csv",
  "params":  {
    "tiles_filename": "examples/data/bb_SF_zipcodes.geojson"
  }
}
```

## ProtectedGeneralization

| Parameter | Description |
|---|---|
| **tiles_filename** | Type: str, optional<br><br>Tiles file for tessellation (geojson or shapefile)<br><br>Default: None |
| **tile_size** | Type: int, optional<br><br>If a tiles file is not provided, a squared tessellation of size 'tile_size' is generated (in meters)<br><br>default: 500 |
| **time_interval** | Type: int, optional<br><br>Size of every time level (In minutes)<br><br>default: None |
| **k** | Type: int, optional<br><br>Level of privacy (minimal number of trajectories sharing the same combination of locations)<br><br>default: 3 |
| **knowledge** | Type: int, optional |

| | |
|---|---|
| | Number of timestamped locations known by the attacker (attacker background knowldege)<br><br>default: 2 |
| **strategy** | Type: str, enumerate ("centroid", "avg"), optional<br><br>To generate the generalized locations compute the centroid of the tile ("centroid") or the average of the locations within the tile ("avg")<br><br>Default: "avg" |
| **time_strategy** | Type: str, enumerate ("same", "keep"), optional<br><br>Keep the original timestamps ("keep") or also generalize time by taking a specific timestamp for every time_level ("same")<br><br>Default: "keep" |

JSON config file example for the Protected Generalization method:

```
{
  "method": "ProtectedGeneralization",
  "input_file": "examples/data/cabs_dataset_20080608_0700_0715.csv",
  "output_folder": "examples/output",
  "main_output_file": "cabs_dataset_20080608_0700_0715_anonymized_prot.csv",
  "params":  {
    "k": 3,
    "knowledge": 2,
    "strategy":"avg"
  }
}
```

## Microaggregation

| Parameter | Description |
|---|---|
| **k** | Type: int<br><br>Minimum number of trajectories to be aggregated in a cluster |
| **clustering_method** | Type: JSON object<br><br>Name and parameters (if any) of the method to cluster the trajectories. Must be one of those defined in 'config.json'<br><br>default: SimpleMDAV |
| **aggregation_method** | Type: JSON object<br><br>Name and parameters (if any) of method to aggregate the trajectories within a cluster. Must be one of those defined in 'config.json'<br><br>Default: Mean_trajectory |

JSON config file example for the Microaggregation method:

```
{
  "method": "Microaggregation",
  "input_file": "examples/data/cabs_dataset_20080608_0700_0715.csv",
  "output_folder": "examples/output",
  "main_output_file": "cabs_dataset_20080608_0700_0715_anonymized_micro.csv",
  "params": {
    "k": 3,
    "clustering_method": {
      "name": "SimpleMDAV",
      "params": {
        "trajectory_distance": {
          "name": "Martinez2021",
          "params": {
            "p_lambda": 0.00657901067783612
          }
        }
      }
    },
    "aggregation_method": {
      "name": "Mean_trajectory"
    }
  }
}
```

## TimePartMicroaggregation

| Parameter | Description |
|---|---|
| **k** | `Type: int`<br><br>Minimum number of trajectories to be aggregated in a cluster<br><br>`default: 3` |
| **Interval** | `Type: int`<br><br>Time interval in each partitioned dataset (in seconds)<br><br>`Default: 900 (15 min)` |
| **clustering_method** | `Type: JSON object`<br><br>Name and parameters of the method to cluster the trajectories. Must be one of those defined in *'config.json'*<br><br>`default: SimpleMDAV` |
| **aggregation_method** | `Type: JSON object`<br><br>Name and parameters of method to aggregate the trajectories within a cluster. Must be one of those defined in *'config.json'*<br><br>`Default: Martinez2021.Aggregation` |

JSON config file example for the Time Partitioned Microaggregation method:

```
{
  "method": "TimePartMicroaggregation",
  "input_file": "examples/data/cabs_dataset_20080608_0700_0715.csv",
  "output_folder": "examples/output",
  "main_output_file": "cabs_dataset_20080608_0700_0715_anonymized_tpmicro.csv",
  "params": {
    "k": 3,
    "interval": 60,
    "clustering_method": {
      "name": "SimpleMDAV",
      "params": {
        "trajectory_distance": {
          "name": "Martinez2021",
          "params": {
            "p_lambda": 0
          }
        }
      }
    },
    "aggregation_method": {
      "name": "Mean_trajectory",
      "params": {}
    }
  }
}
```

## SwapAllLocations

| Parameter | Description |
|---|---|
| **k** | Type: int, optional<br><br>Minimum number of locations of the swapping cluster<br><br>default: 3 |
| **min_r_s** | Type: int, optional<br><br>Minimum spatial radius of the swapping cluster (in meters)<br><br>default: 100 |
| **max_r_s** | Type: int, optional<br><br>Maximum spatial radius for building the swapping cluster (in meters)<br><br>default: 500 |
| **min_r_t** | Type: int, optional<br><br>Minimum temporal threshold for building the swapping cluster (in seconds)<br><br>default: 60 |
| **max_r_t** | Type: int, optional<br><br>Maximum temporal threshold for building the swapping cluster (in seconds) |

| | |
|---|---|
| | default: 120 |
| **tile_size** | Type: int, optional<br>Size of tessellation to improve privacy at trajectory level<br>default: 1000 |
| **seed** | Type: int, optional<br>Seed for the random swapping process<br>default: None |

JSON config file example for the SwapAllLocations method:

```
{
  "method": "SwapLocations",
  "input_file": "examples/data/cabs_dataset_20080608_0700_0715.csv",
  "output_folder": "examples/output",
  "main_output_file": "cabs_dataset_20080608_0700_0715_anonymized_swap.csv",
  "params": {
    "k": 5,
    "max_r_s": 600,
    "max_r_t": 200,
    "min_r_s": 150,
    "min_r_t": 10,
    "seed": 42
  }
}
```

## SwapMob

| Parameter | Description |
|---|---|
| **spatial_thold** | Type: int, optional<br>Maximum distance to consider two locations as close (in kilometers)<br>default: 0.2 |
| **temporal_thold** | Type: int, optional<br>Maximum time difference to consider two locations as coexistent (in seconds)<br>default: 30 |
| **min_n_swap** | Type: int, optional<br>Minimum number of swaps for a trajectory for not being removed.<br>default: 1 |
| **seed** | Type: int, optional |

| | |
|---|---|
| | Seed for the random swapping process<br><br>`default: None` |

JSON config file example for the SwapMob method:

```
{
  "method": "SwapMob",
  "input_file": "examples/data/cabs_dataset_20080608_0700_0715.csv",
  "output_folder": "examples/output",
  "main_output_file": "cabs_dataset_20080608_0700_0715_anonymized_swapmob.csv",
  "params": {
    "spatial_thold": 0.1,
    "temporal_thold": 60,
    "seed": 42
  }
}
```

### 6.4.1.2. Clustering methods

Some anonymization methods (such as Microaggregation or Time Partitioned Microaggregation) require a clustering method to partition the dataset into disjoint clusters.

#### SimpleMDAV

| Parameter | Description |
|---|---|
| **trajectory_distance** | `Type: JSON object, optional`<br><br>Name and parameters (if any) of the method to compute the distance between two trajectories. Must be one of those defined in 'config.json'<br><br>`default: Martinez2021` |
| **aggregation_method** | `Type: JSON object, optional`<br><br>Name and parameters (if any) of the method to aggregate the trajectories within a cluster. Must be one of those defined in 'config.json'<br><br>`default: Mean_trajectory` |

### 6.4.1.3. Aggregation methods

#### Mean_Trajectory

No params

### Closest_trajectory_to_mean_trajectory

| Parameter | Description |
|---|---|
| p_lambda | Type: float<br><br>Weight of the temporal component (see section 6.3.1.2) |

#### 6.4.1.4. Trajectory distances

### Martinez2021

| Parameter | Description |
|---|---|
| p_lambda | Type: float<br><br>Weight of the temporal component (see section 6.3.1.2). If p_lambda is not provided, the value is computed. If p_lambda = 0, the temporal component is not considered. Use this option if all locations of the dataset are from a short time interval.<br><br>default: None |

See the config file example for the Microaggregation anoymization method.

## *6.4.2. Utility metrics*

As previously mentioned, the anonymization module also includes tools to compute and compare some utility and privacy metrics of original and anonymized datasets. Again, the parameter values to compute the measures are provided to the application through a JSON file:

```
$ python -m mdl_anonymizer measures -f parameter_file.json
```

The parameter file contains all the measures to be computed with their corresponding parameters:

| Parameter | Description |
|---|---|
| original_dataset | Type: str<br><br>Path of the original dataset |
| anonymized_dataset | Type: str<br><br>Path of the anonymized dataset |
| output_folder | Type: str<br><br>Output folder |

| | |
|---|---|
| **main_output_file** | `Type: str`<br><br>Name of the main output file |
| **measures** | `Type: Array of JSON objects`<br><br>List of measures to be computed. Every measure includes their own parameters (if any). They should appear in the main configuration file (*config.json*). Currently, these are the developed measures:<br><br>• ScikitMeasures<br>• Rsme<br>• PropensityScore<br>• RecordLinkage<br>• TrajectoriesRemoved |

Example of a 'measures' parameter file:

```
{
  "original_dataset": "examples/data/cabs_dataset_20080608_0700_0715.csv",
  "anonymized_dataset": "examples/data/cabs_dataset_20080608_0700_0715_an.csv",
  "output_folder": "examples/output/",
  "main_output_file": "measures_cabs_dataset_20080608_0700_0715_simple.json",
  "measures": [
    {
      "name":  "ScikitMeasures",
      "params":  {
        "sort": true,
        "tesselation_meters": 250
     }
    },
    {
      "name": "TrajectoriesRemoved",
      "params": {}
    },
    {
      "name": "Rsme",
      "params": {
        "trajectory_distance": {
          "name": "Martinez2021",
          "params": {
            "p_lambda": 0.005492628237142597
          }
        }
      }
    },
    {
      "name": "PropensityScore",
      "params": {}
    },
    {
      "name": "RecordLinkage",
      "params": {
        "percen_window_size": null,
        "trajectory_distance": {
          "name": "Martinez2021",
          "params": {
            "p_lambda": 0.005492628237142597
          }
```

```
          }
        }
      }
    ]
}
```

### 6.4.2.1. ScikitMeasures

We leverage on the well-known scikit-mobility[4] library to compute some utility metrics:

- visits_per_location
- distance_straight_line
- uncorrelated_location_entropy
- random_location_entropy
- mean_square_displacement

To compute some of these measures, the datasets to be compared need to be generalized using a simple squared tessellation.

| Parameter | Description |
|---|---|
| **sort** | `Type: Boolean, optional`<br><br>Sort the datasets by timestamp<br><br>`default: True` |
| **tessellation_meters** | `Type: int, optional`<br><br>Size of the squared tessellation (in meters)<br><br>`Default: 250` |

### 6.4.2.2. RSME

Information loss measures the differences between the original and anonymized datasets (see section 6.3.4.1). This measure only needs the trajectory distance to use:

| Parameter | Description |
|---|---|
| **trajectory_distance** | `Type: JSON object, optional` |

[4] https://github.com/scikit-mobility/scikit-mobility

| | |
|---|---|
| | Name and parameters (if any) of the method to compute the distance between two trajectories. Must be one of those defined in '*config.json*'<br><br>`default: Martinez2021` |

### 6.4.2.3. PropensityScore

In order to compute the propensity score (section 6.3.4.3).

| Parameter | Description |
|---|---|
| **tiles_size** | `Type: int, optional`<br><br>Size of the squared tessellation (in meters)<br><br>`default: 200` |
| **time_interval** | `Type: int, optional`<br><br>Consider the temporal component, size of every time interval (in seconds)<br><br>`default: None` |
| **seed** | `Type: int, optional`<br><br>Seed for the random process<br><br>`Default: none` |

### 6.4.2.4. RecordLinkage

To measure the practical privacy resulting of the anonymization process, we can also measure the disclosure risk of the resulting dataset (section 6.3.4.4).

| Parameter | Description |
|---|---|
| **trajectory_distance** | `Type: JSON object, optional`<br><br>Name and parameters (if any) of the method to compute the distance between two trajectories. Must be one of those defined in 'config.json'<br><br>`default: Martinez2021` |
| **percen_window_size** | `Type: float, optional`<br><br>Percentage of trajectories to be considered. If it is not provided, a percentage is computed depending on the size of the dataset.<br><br>`Default: none` |

#### 6.4.2.5. TrajectoriesRemoved

This measure has no parameters. It computes the percentage of locations and trajectories removed in the anonymized dataset.

### *6.4.3. Privacy-preserving analysis methods*

The parameter values to configure the anonymization methods are provided to the application using a JSON file:

```
$ python -m mdl_anonymizer analysis -f parameters_file.json
```

There are some common parameters to all the analysis methods:

| Parameter | Description |
|---|---|
| **method** | `Type: string`<br><br>The name of the anonymization method to be executed. Must be one of those defined in 'config.json'. For the moment only 'QuadTreeHeatMap' is implemented |
| **input_file** | `Type: string`<br><br>The dataset to be analyze |
| **output_folder** | `Type: string`<br><br>Folder to save the outputs |
| **main_output_file** | `Type: string, optional`<br><br>The name of the main output file |
| **params** | `Type: JSON object, optional`<br><br>Parameters of the analysis method |

Each of the analysis methods has some specific parameters that can be added to the parameters file:

#### 6.4.3.1. QuadTreeHeatMap

| Parameter | Description |
|---|---|
| **min_k** | `Type: int, optional` |

| | |
|---|---|
| | Minimum number of locations allowed to coexist in a QuadTree sector<br><br>`Default: 5` |
| **min_sector_length** | `Type: int, optional`<br><br>Minimum side length for a QuadTree sector (in meters)<br><br>`Default: 100` |
| **merge_sectors** | `Type: Boolean, optional`<br><br>If True, sector with an insufficient number of locations will be merged with neighbouring sectors<br><br>`Default: True` |
| **split_n_locations** | `Type: int, optional`<br><br>Max number of locations allowed in a QuadTree sector before it is split into 4 subsectors. Must be greater than *min_k*.<br><br>`Default: min_k` |

JSON config file example:

```
{
  "method": "QuadTreeHeatMap",
  "input_file": "examples/data/cabs_dataset_20080608_0700_0715.csv",
  "output_folder": "examples/output",
  "main_output_file": "cabs_dataset_20080608_0700_0715_QuadTreeHeatMap.json",
  "params": {
    "min_k": 5,
    "min_sector_length": 50,
    "merge_sectors": true
  }
}
```

## 6.4.4. Filtering

In addition, the anonymization module also provides some filtering functionality. Again, these functionalities can be extended by developing further filtering methods.

```
$ python -m mdl_anonymizer filter -f parameter_file.json
```

In this case, the config file has the following parameters:

| Parameter | Description |
|---|---|
| **input_filename** | `Type: string` |

| | |
|---|---|
| | Filename of the dataset to be filtered |
| **output_filename** | `Type: string, optional`<br><br>The name of the output file |
| **methods** | `Type: array of JSON objects`<br><br>Name and value of each filtering methods that should be applied to the dataset. Currently just two filtering mechanisms are implemented:<br><br>• min_locations: Remove trajectories with less that {value} locations<br>• max_speed: Remove trajectories with a speed greater than {value} between two of its locations. |

JSON config example:

```
{
  "input_filename": "examples/data/cabs_dataset_20080608_0700_0715.csv",
  "output_filename": "examples/output/filtered_cabs_dataset_20080608_0700_0715.csv",
  "methods": [
    {
      "min_locations":  5
    },
    {
      "max_speed":  300
    }
  ]
}
```



Funded by the European Union

# 7. Bibliography

1. J. Domingo-Ferrer and R. Trujillo-Rasua, "Microaggregation- and permutation-based anonymization of movement data", Information Sciences, Vol. 208, pp. 55-80, Nov 2012, ISSN: 0020-0255.

2. J. Domingo-Ferrer, S. Martínez and David Sánchez, "Decentralized k-anonymization of trajectories via privacy-preserving tit-for-tat", Computer Communications, Vol. 190, pp. 57-68, Jun 2022, ISSN: 0140-3664.

3. A. Gasmelseed, N. Mahmood, “Study of hand preferences on signature for right-handed and left-handed peoples,” International Journal of Advances in Engineering & Technology, vol. 1 (5), pp. 41-46, 1963.

4. J. Salas, D. Megías and V. Torra, “SwapMob: Swapping Trajectories for Mobility Anonymization”. Privacy in Statistical Databases – PSD2018. Lecture Notes in Computer Science vol. 11126, pp. 331-346. Sep 2018.

5. S. Shang, L. Chen, Z. Wei, C.S. Jensen, K. Zheng, P. Kalnis, “Trajectory similarity join in spatial networks,” in Proceedings of the VLDB Endowment, vol. 10 (11), pp. 1178-1189, 2017.

6. H. Su, S. Liu, B. Zheng, X. Zhou, K. Zheng, “A survey of trajectory distance measures and performance evaluation,” The VLDB Journal, vol. 29, pp. 3-32, 2020.



Funded by the European Union

# MobiDataLab consortium

The consortium of MobiDataLab consists of 10 partners with multidisciplinary and complementary competencies. This includes leading universities, networks and industry sector specialists.

@MobiDataLab
#MobiDataLab

https://www.linkedin.com/company/mobidatalab

For further information please visit **www.mobidatalab.eu**

MobiDataLab is co-funded by the EU under the H2020 Research and Innovation Programme (grant agreement No 101006879).