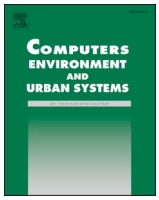
Computers, Environment and Urban Systems 101 (2023) 101938

Contents lists available at ScienceDirect

Computers, Environment and Urban Systems

journal homepage: www.elsevier.com/locate/ceus



Trackintel: An open-source Python library for human mobility analysis

Henry Martin [a,b,*,1], Ye Hong [a,1], Nina Wiedemann [a,1], Dominik Bucher [c,2], Martin Raubal [a]

[a] Institute of Cartography and Geoinformation, ETH Zurich, Zurich, Switzerland
[b] Institute of Advanced Research in Artificial Intelligence (IARAI), Vienna, Austria
[c] c.technology, Tessinerplatz 7, 8002 Zürich, Switzerland

ARTICLE INFO

Keywords:
Human mobility analysis
Open-source software
Transport planning
Data mining
Python
Tracking studies

ABSTRACT

Over the past decade, scientific studies have used the growing availability of large tracking datasets to enhance our understanding of human mobility behavior. However, so far data processing pipelines for the varying data collection methods are not standardized and consequently limit the reproducibility, comparability, and transferability of methods and results in quantitative human mobility analysis. This paper presents Trackintel, an open-source Python library for human mobility analysis. Trackintel is built on a standard data model for human mobility used in transport planning that is compatible with different types of tracking data. We introduce the main functionalities of the library that covers the full life-cycle of human mobility analysis, including processing steps according to the conceptual data model, read and write interfaces, as well as analysis functions (e.g., data quality assessment, travel mode prediction, and location labeling). We showcase the effectiveness of the Trackintel library through a case study with four different tracking datasets. Trackintel can serve as an essential tool to standardize mobility data analysis and increase the transparency and comparability of novel research on human mobility. The library is available open-source at https://github.com/mie-lab/trackintel.

1. Introduction

Human mobility studies using large-scale human digital traces have boomed over the last decade. On the collective level, researchers revealed that human movement can be universally described using statistical distributions, i.e., the power-law distribution of consecutive displacements (Brockmann et al., 2006; Rhee et al., 2011), stationary time between displacements (Rhee et al., 2011; Song et al., 2010), and characteristic distance traveled by individuals (i.e., the radius of gyration) (González et al., 2008; Pappalardo et al., 2015). Moreover, it has been shown that individuals exhibit markedly regular location visitation patterns (Schneider, Belik, Couronné, Smoreda, & González, 2013) with high theoretical predictability (Song et al., 2010). People spend most of their time in a few locations (González et al., 2008; Song et al., 2010) and maintain a stable number of important locations over time (Alessandretti et al., 2018).

To a large extent, this progress can be attributed to the widespread availability of large mobility datasets stemming from information and communications technology (ICT) and location-based services (LBS) that are now integrated into many aspects of our daily life (Huang et al., 2018; Keßler & McKenzie, 2018). Aside from the progress on the analysis of human movement itself, the increased availability of tracking data has led to the rapid growth of studies that use human mobility data to study phenomena related to human mobility, such as understanding of residential income segregation (Moro et al., 2021), quantifying urbanization levels and city livability (Bassolas et al., 2019), classify functional areas of a city (Yuan & Raubal, 2012), urban sensing (Ahas et al., 2015), developing infrastructure for sustainable mobility (Xu, Çolak, Kara, Moura, & González, 2018) and responding to epidemic spreading (Chang et al., 2021). However, the *raw* digital traces are often not the targeted unit of analysis; for example, a location where people perform an activity can not directly be derived from GPS track points or mobile phone tower data. Studies thus employ various steps to preprocess data into the desired format. These steps and their outcome are often different across studies (Chen et al., 2016) due to the variety of the datasets and the different understanding of the definitions, which has led to a vast collection of dataset-specific preprocessing and analysis methods. For example, the study by Feng et al. (2018), which proposes the DeepMove model that is now widely accepted as a deep learning baseline model for next location prediction (Luca et al., 2021), generally

* Corresponding author at: ETH Zurich, Institute of Cartography and Geoinformation HIL D 54.3, Stefano-Franscini-Platz 5, CH-8093 Zürich, Switzerland.
E-mail addresses: martinhe@ethz.ch (H. Martin), hongy@ethz.ch (Y. Hong), nwiedemann@ethz.ch (N. Wiedemann).
[1] Authors contributed equally. Order was determined randomly.
[2] The majority of this work was done while the author was at the Chair of Geoinformation Engineering, ETH Zurich.

https://doi.org/10.1016/j.compenvurbsys.2023.101938
Received 4 August 2022; Received in revised form 5 January 2023; Accepted 12 January 2023
Available online 7 February 2023

regards each raw position record as a *location* and does not perform preprocessing. However, focusing on the same problem, Urner et al. (2018) extract *staypoints* (i.e., all the points where a user stayed for at least a certain duration) from GPS track points and further aggregate them into locations using the *k*-means algorithm. Solomon et al. (2021) apply a similar processing concept but introduce the mean shift algorithm to detect staypoints, which are then merged into locations according to a distance threshold. These examples show how not using a standard movement model definition and a common preprocessing standard limit the reproducibility and comparability of the methods and analysis results.

To address these problems, we present Trackintel, an open-source python library for the processing and analysis of movement data. Trackintel is based on an established model for human mobility taken from transport planning, which defines hierarchical levels of movement centred around the concept of *activities*. Trackintel standardizes the definition and implementation of the data processing steps derived from this data model. Our work thereby makes the assumptions, parameters and filtering steps explicit and provides transparent preprocessing steps, whose implementations are known to have substantial effects on the analysis results. Trackintel further provides analysis, visualization and support functions to enrich the raw tracking data with human-mobility-specific information. Due to the versatility of the data model, Trackintel standardizes preprocessing for many types of tracking data. It thereby greatly simplifies the benchmarking of novel analysis methods, increases their reproducibility, and facilitates quantitative research based on tracking data.

The remainder of the paper is structured as follows. Section 2 provides an overview of existing libraries for analyzing and preprocessing movement data. Section 3 first introduces the hierarchical model for human mobility analysis and describes its implementation in Trackintel. This section then proceeds to present the most important functionalities of Trackintel to process movement data. In Section 4, we showcase the capabilities of Trackintel to simplify the analysis and comparison of several different tracking datasets. Finally, we summarize and conclude this work in Section 5.

2. Related work: Libraries for movement data

Due to the long history of research in transportation, human migration, and animal behavioral research, a large variety of libraries for (human) movement data processing exists. Joo et al. (2020) survey an impressive number of 58 packages for movement analysis in R. Based on this work and the overview provided by Graser (2020), we selected the libraries that aim at supporting movement analysis in Python, R and C++. In Table 1, these selected libraries are compared in terms of their user-friendliness (documentation and robustness), their focus and their provided functionality for human movement data analysis. To compare packages by the quality of their documentation, we evaluate them on a scale from 0 to 6 based on criteria used for peer-review of packages by pyOpenSci (Holdgraf et al., 2022) and ROpenSci.[3] See appendix A for the list of criteria.

Many of the surveyed R libraries have a strong focus on animal behavioral analysis (Joo et al., 2020) (not all included in Table 1). The packages that can (also) be applied to human mobility analysis have a focus on basic statistical analysis of trajectories, such as measuring the spatial extent of animal motion (e.g., adehabitatLT (Calenge, 2006)), or the duration and distance of movement trajectories (e.g., TrackR (Frick & Kosmidis, 2017)). Currently, no coherent framework is available in R that provides the functionalities specific to human movement analysis, e.g. trip detection and transport mode labeling. Furthermore, there are several libraries available in C++, such as Tracktable (Andrew, 2014), MEOS,[4] and MoveTK[5] that promise efficient and fast tools for trajectory data processing, although they may be less accessible for the research audience in human mobility and transportation. Furthermore, these libraries provide only highly specific functionalities and do not represent a comprehensive framework for movement data analysis.

ArcGIS Pro is a proprietary software for general spatial data processing with modules for movement data analysis such as speed and acceleration computation, trackpoint clustering and in particular trajectory visualization.[6] However, the different functionalities are scattered over different toolboxes and ArcGIS Pro does not provide a consistent framework for the analysis of movement data. Due to its proprietary nature, we could not evaluate documentation and testing as we did for the other packages, but we assume both are on a high level. We did not include QGIS,[7] a high quality open-source GIS Project, in the table, as there are no well-maintained plug-ins for movement or trajectory data analysis available. However, QGIS could be used in combination with Python libraries or the mobilityDB (Zimányi et al., 2020) library.

In Python, many open-source libraries have emerged as tools to both facilitate and standardize data processing and analysis. The geographic information science (GIScience) community in particular has benefited significantly from Python libraries, for example, the data models implemented in Shapely (Gillies, 2013) and the I/O formats for geographic data as offered in the Fiona package.[8] Most importantly, spatial data can be handled easily with the Geopandas library (Jordahl et al., 2022) that directly builds up on Pandas (The pandas development team, 2023), one of the most established Python libraries for data analysis and manipulation.

In the past years, Python has become the de-facto standard for data science and machine learning applications, which are increasingly important for the analysis of movement data (Luca et al., 2021; Toch et al., 2018). However, only a few libraries have attempted to provide preprocessing and analysis tools specifically for human mobility in a comprehensive Python package (see Table 1). Although many algorithms for trajectory data mining were developed in the last decade (Zheng, 2015), their open-source availability in Python is limited, and they often suffer from insufficient documentation and testing standards, such as HuMobi (Smolak et al., 2021) or MovinPy. Others are well-maintained but limited in scopes, such as Traja (Shenk et al., 2021) that targets animal movement, PTRAIL (Haidri et al., 2021) for parallel processing, and TransBigData (Qing & Yuan, 2022) which focuses on data analysis on a collective level, similar to the R library stplanr (Lovelace & Ellison, 2018).

Notable exceptions are MovingPandas (Graser, 2019) and scikit-mobility (Pappalardo et al., 2022). MovingPandas is based on Pandas and Geopandas and focuses on low-level trajectory manipulation, such as splitting, merging and visualizing trajectories. On the contrary, the scikit-mobility library targets high-level analysis functions, including computing human mobility metrics, generating synthetic trajectories and assessing privacy risks. Both libraries are actively maintained and contain various measures to ensure high code quality, but the definition of their data model implies a focus on movement trajectories (MovingPandas) or mobility flows (scikit-mobility), which omits important concepts describing individual human mobility such as activities, trips or tours (Axhausen, 2007).

We aim to close this gap with the Trackintel framework that utilizes an established data model from the transportation literature, which

[3] https://ropensci.org/

[4] https://github.com/adonmo/meos

[5] https://github.com/movetk/movetk

[6] https://pro.arcgis.com/en/pro-app/2.8/tool-reference/intelligence/an-overview-of-the-movement-analysis-toolset.htm

[7] https://www.qgis.org/en/site/

[8] https://github.com/Toblerity/Fiona

Table 1
Comparison of movement data libraries. Packages are predominantly available open source in R and Python and they are compared with regard to their focus, documentation and functionality. While other movement analysis libraries already provide well-maintained and documented code with rich functionality for trajectory analysis, only Trackintel provides robust and flexible methods to aggregate trajectories into locations, trips and tours.
(✓/✓̸/ x: available / partially available / not available).

Package name	Focus	Programming language Python (P)	Doc. score	Test coverage (* / **: not reported but low / high)	individual (I) / collective (C)	human (H), animal (A) and/or object(O)	Staypoint detection	Aggregation to location	Aggregation to trips	Aggregation to tours	Tracking quality assessment	Transport mode labelling	Home and work labelling	Visualization	Trajectory statistics (−/+/++: none / basic / rich)
Trackintel	Human mobility analysis	P	6	98%	I	H	✓	✓	✓	✓	✓	✓	✓	✓	+
Scikit-mobility (Pappalardo et al., 2022)	Human mobility analysis	P	5	**	I	H	✓	✓	x	x	✓̸	x	✓̸	✓	++
Movingpandas (Graser, 2019)	Movement data analysis	P	6	96%	I	H/A/O	✓	x	x	x	x	x	x	✓	++
PyMove	Querying and visualizing trajectories	P	5	85%	I	H/A/O	✓	x	x	x	x	x	x	✓	+
MovinPy	Mobility data analysis	P	3	0	I	H	x	x	x	x	✓̸	x	x	x	−
HuMobi (Smolak et al., 2021)	Human mobility prediction	P	3	0	I	H	✓	✓	x	x	✓̸	x	x	x	+
PTRAIL (Haidri et al., 2021)	Parallelization and feature extraction	P	4	**	I	H/A/O	x	x	x	x	x	x	x	✓	++
TransBigData (Qing & Yuan, 2022)	Transportation	P	5	90%	C	H	✓	x	x	x	✓̸	x	✓	✓	−
mobilityDB (Zimányi et al., 2020)	Storing and querying	SQL	6	97%	I	H/A	x	x	x	x	x	x	x	✓̸	+
Traja (Shenk et al., 2021)	Animal trajectories	P	6	76%	I	A	✓̸	x	x	x	x	x	x	✓	++
Tracktable (Andrew, 2014)	Moving object tracking	P/C++	2	**	I	O	✓̸	x	x	x	x	x	x	✓	++
MEOS	Spatio-temporal data analysis	C++	4	*	I	H/A/O	x	x	x	x	x	x	x	x	+
MoveTK	Movement analysis	C++	–	**	I/C	H/A/O	x	x	x	x	x	x	x	x	++
adehabitatLT (Calenge, 2011)	Animal habitat	R	4	**	I	A	x	x	x	x	x	x	x	x	++
moveVis	Visualization	R	6	93%	I	A	x	x	x	x	x	x	x	✓	−
stplanr (Lovelace & Ellison, 2018)	Sustainable transport planning	R	6	*	C	H	x	x	x	x	x	x	x	x	−
trajectories	Object tracking and interaction	R	5	*	I	H/A/O	x	x	x	x	x	x	x	✓	+
TrackR	Running and cycling data	R	6	52%	I	H	✓	x	x	x	x	x	x	✓	+
ArcGIS Pro	Spatial data	(P)	–	**	I/C	H/A/O	✓̸	✓̸	x	x	x	x	✓̸	✓	++

incorporates different semantic aggregation levels of tracking data specific to human mobility.

3. Trackintel framework

Trackintel is a library for the analysis of spatio-temporal tracking data with a focus on human mobility. The core of Trackintel is the hierarchical data model for movement data (Axhausen, 2007) that is widely adopted in GIScience (Bucher et al., 2019), transport planning (Chen et al., 2016) and related fields (Rout et al., 2021). We provide easy-to-use and efficient functionalities for the full life-cycle of human mobility data analysis, including import and export of tracking data of various types (e.g., GPS track points, location-based social network (LBSN) check-ins, call detail records), data model generation and preprocessing, analysis, and visualization. A conceptual overview of the different components of Trackintel can be found in Fig. 1.

Trackintel focuses on the mobility of individual persons or objects (e.g., as opposed to crowd flows), and all functionalities are implemented as user-specific, based on unique user identifiers that link data to the respective tracked users. Trackintel is implemented in Python and is built mainly on top of Pandas (The pandas development team, 2023) and GeoPandas (Jordahl et al., 2022) using accessor classes, a method to extend Pandas classes.[9] This design makes Trackintel easy to use for Python users and ensures its broad compatibility with other Python spatial analysis libraries.

3.1. The Trackintel data model

The modeling framework employed by Trackintel is based on the activity-based analysis framework in transport planning, which regards travel demand as derived from our need to perform activities at different locations. We follow the definition from Schönfelder & Axhausen, 2016 that people's daily mobility consists of staying at locations to perform activities and traveling between locations for the next activity. In this definition and following the description in Axhausen (2007), movement is separated from activities at different semantic levels. Trackintel implements six classes to represent movement data in this hierarchical model: *positionfix*, *staypoint*, *tripleg*, *trip*, *tour*, and *location*. Fig. 2 gives an overview of the hierarchical modeling structure and shows the classes in a UML diagram with their mandatory attributes and optional attributes in square brackets. All Trackintel classes are implemented as Pandas Dataframes or Geopandas Geodataframes. In order to be considered a valid Trackintel object, all mandatory attributes have to be present as columns with the correct names, as shown in Fig. 2. A more detailed explanation of the required and optional attributes of the Trackintel classes is given in Table 2. Geometries need to be of the defined type, with the exception of the *Location* class that can have multiple geometries. Furthermore, all timestamps for the time fields required by Trackintel have to be timezone-aware.[10] Besides these formal requirements, classes can contain any additional information required for specific analysis. In the following, the different classes and their semantics are introduced.

3.1.1. Positionfix

Positionfix is the smallest tracking unit in the Trackintel data model, consisting of timestamped position records, for example, generated by GNSS trackers or call detailed records (CDR) data. Positionfixes are often directly transferred from raw tracking data and are thus a natural entry point to the Trackintel data model, where it can further be processed and segmented into triplegs and staypoints. No inherent semantics are included since movements and activities cannot be distinguished from a Positionfix.

3.1.2. Staypoint

Staypoint represents a point in space, which is defined as an individual remaining within a defined geographical radius for a defined time. Compared to the raw *positionfix* points, staypoints can represent stationary points that carry particular semantics, such as the purpose of the stay, or they can represent an intermediate stay, such as waiting for a bus. To distinguish between these two types of staypoints, we introduce the concept of *activity*: an activity staypoint is usually the reason for a person to travel and has an important purpose with an attached activity label (e.g., home), while a non-activity staypoint only represents a trivial stationary point (e.g., waiting). The exact definition of an activity depends on the goal of the study. In Trackintel, activities are staypoints with the attribute *activity_flag* set to *True*, which can be obtained through user labels or directly inferred from data (see Section 3.5.2). While activity *staypoints* are the basic unit for constructing trips, which mark the start and end of a *trip*, non-activity *staypoints* can only be part of a *trip*. Additionally, *staypoints* can be spatially aggregated to form *locations*.

3.1.3. Tripleg

The most basic level of movement is defined as *tripleg* (referred to as stage in Axhausen (2007)), which formally represents a continuous movement without changing transport mode or vehicle. Therefore, triplegs contain semantics about the movement of an individual, such as the mode of transport that is stored in the attribute field *mode* if available. This information can be obtained from user labels (Hong et al., 2021; Zheng et al., 2010) or inferred using heuristics directly from the data, which is implemented as labeling functions in Trackintel (see Section 3.5.1). Triplegs can be created from positionfixes and can be aggregated to form trips.

3.1.4. Trip

Trip represents all travels between two activities and summarizes all triplegs and non-activity staypoints between two consecutive activity staypoints. Trips inherit the activity purpose from the activity label attribute of the destination staypoint. As they are often the primary quantity of interest in transport planning studies, trips, together with activities, are the core of the movement data model proposed in Axhausen (2007).

3.1.5. Location

Activity staypoints represent individual visits to places that are significant to the visitor. Trackintel models these significant places using the *location* class to enable the characterization of the place that is visited. While the information attached to staypoints is bound to the individual visit (e.g., the specific activity or the time of day), the semantics of locations are related to the place independent of the visit (e.g., land use or the opening hours of a shop). Locations are modeled with two different geometries, a point geometry for the center of the location and a polygon geometry to describe the extent of a location.

3.1.6. Tour

The mobility of individuals is centered around a few significant locations that act as the basis of their travel behavior. Individuals conduct several activities and trips if convenient but return home (or to a similar significant location) to plan their next activity. This behavior can be analyzed using the *tours* class, which is defined as "a sequence of trips starting and ending at the same location" (Axhausen, 2007, p. 4), referring to the location class defined above. A special case of a tour is the concept of *journey* that starts and ends at the home location of an individual. In Trackintel, a tour can be flagged as a journey using the *journey* attribute. A tour contains multiple trips, but one trip can also be part of several tours in case they are nested, e.g. the trip from the work location to the supermarket and back is part of a larger journey that

[9] https://pandas.pydata.org/docs/development/extending.html

[10] For an explanation, see https://docs.python.org/3/library/datetime.html#aware-and-naive-objects

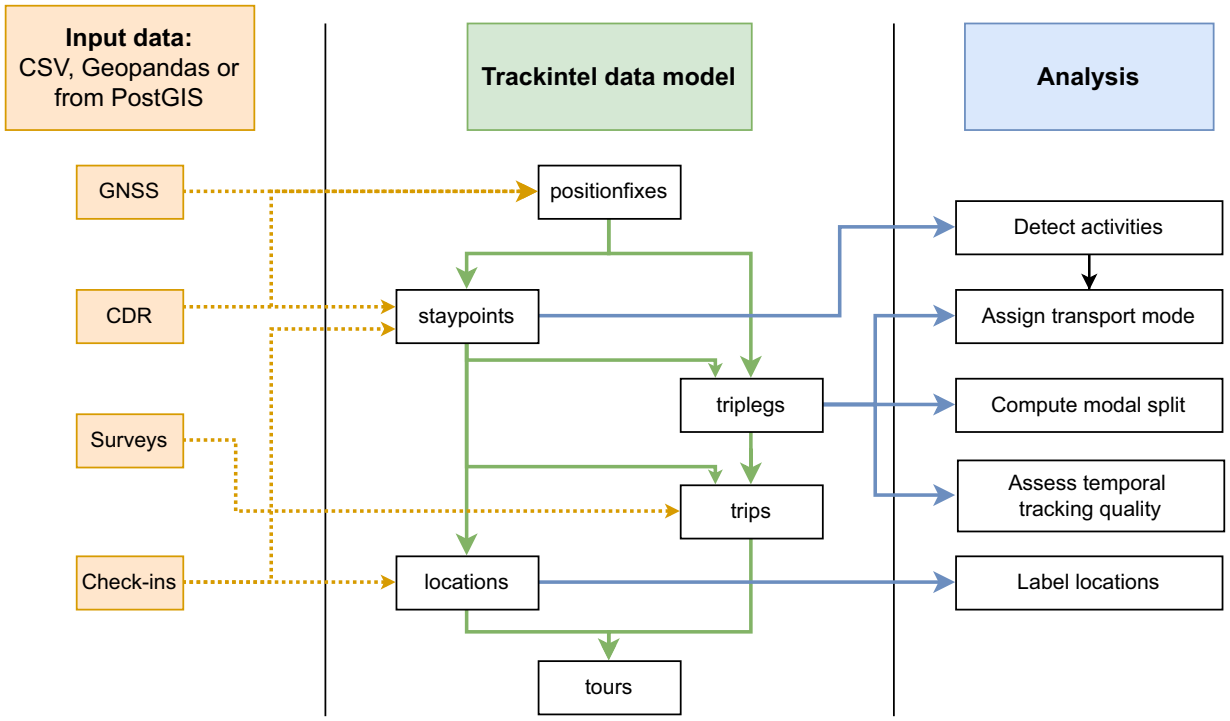


Fig. 1. Overview of the Trackintel framework.

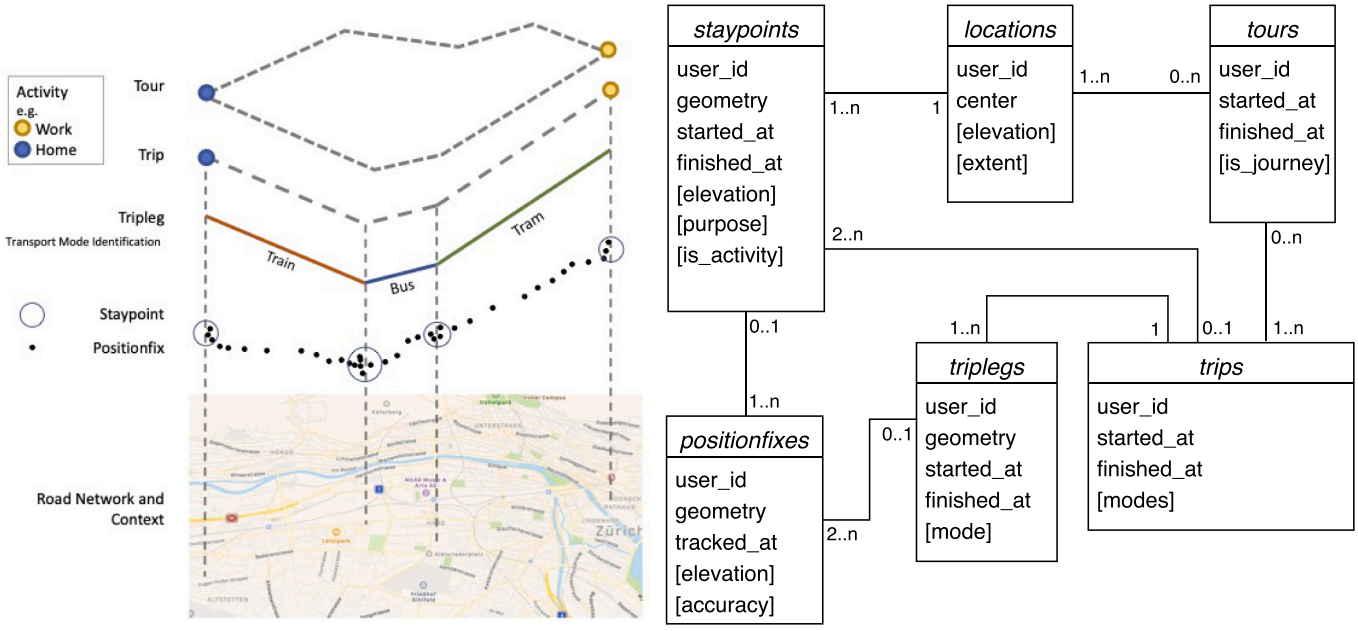


Fig. 2. Semantic visualization of the Trackintel data models and their UML diagram, with mandatory and optional attributes (shown in square brackets). The relations between the different classes are shown in the connecting lines. Figure adapted from Jonietz and Bucher (2018).

started at home.

3.2. Data model generation

The core functionality of Trackintel is to generate all classes defined in the movement data model from the raw tracking data. In practice, this refers to the generation of the entire hierarchical movement data model from positionfix data. However, it should be noted that it is not required and often not practical to enter the framework from positionfixes - the framework can be accessed at any semantic level depending on the available data (e.g., location-based social network (LSBN) check-ins represent staypoints without the availability of positionfixes; see Fig. 1 for examples of input levels for different tracking data types). The following section presents the implemented preprocessing steps necessary to aggregate data through the hierarchy levels. The output of all `generate` functions is a (Geo)DataFrame with the fields listed in Table 2.

3.2.1. Generate staypoints

In Trackintel, *staypoints* are generated from *positionfixes* based on the

Table 2
Description of the mandatory and optional columns for Trackintel data models.

Data models	Fields	Description
All	id	The unique identifier for the record
	user_id	The unique user identifier
	tracked_at	The timestamp for the point (only for positionfix)
	started_at	The starting time of the record (except for positionfix and location)
	finished_at	The ending time of the record (except for positionfix and location)
Positionfix	geometries	Point geometry
Staypoint	geometries	Point geometry
	purpose (optional)	Purpose label for the staypoint. This could be either an activity purpose (e.g., home), or an non-activity purpose (e.g., wait).
	is_activity (optional)	Boolean flag indicating whether the staypoint is an activity
Location	center	Point geometry representing the center
	extent (optional)	Polygon geometry representing the extent
Tripleg	geometries	Line geometry
	mode (optional)	Transport mode label
Trip	origin_staypoint_id	The identifier of the starting staypoint
	destination_staypoint_id	The identifier of the destination staypoint
	primary_mode (optional)	The main transport mode label
Tour	location_id	The start and end location identifier
	journey	Boolean flag indicating whether the tour is a journey (A tour is called a journey if the start and end location is home).

sliding window detection algorithm first reported in Li et al. (2008), which has become a standard algorithm for staypoint detection (Zheng, 2015). For each individual, the algorithm iterates over all positionfixes and determines groups of points that satisfy the predefined distance and time thresholds. Each output staypoint inherits the starting and ending time from the first and last positionfix that belongs to it, respectively, as well as the mean geometry coordinates of the group of positionfixes. The implemented staypoint detection algorithm extends the algorithm from Li et al. (2008) by an option to exclude temporal gaps in the tracking data, commonly observed in many datasets due to low temporal tracking coverage. This behavior can be controlled using a parameter representing the maximum time between two consecutive positionfixes such that they are still considered to belong to the same staypoint.

3.2.2. Generate locations

Locations can be generated by aggregating staypoints. Existing studies proposed community detection algorithm (Aslak & Alessandretti, 2020) and spatial clustering algorithms, such as OPTICS (Yuan et al., 2013), mean shift (Solomon et al., 2021), and DBSCAN (Luo et al., 2017) to perform this processing step. Here, we implement the most commonly employed DBSCAN algorithm to aggregate staypoints that are spatially close to locations (Hariharan & Toyama, 2004; Jonietz & Bucher, 2018). DBSCAN adopts a set of neighborhood characterization parameters ϵ and the minimum number of samples (*min_samples*) to define how dense the input data has to be considered a cluster. In the context of location generation, ϵ controls the distance of which nearby staypoints will be merged into a single location, and *min_samples* determines the minimum number of staypoints to form a location (i.e., how many visits are required at the same place to consider it as significant). Generated locations are equipped with two different geometries. The *center* is a point geometry, calculated as the mean coordinates from all staypoints assigned to the cluster; the *extent* is a polygon geometry, defined as the bounding box of all belonging staypoints. Furthermore, we provide the flexibility to generate locations that are significant to a single user (Fig. 3 right) or to all users present in a dataset (Fig. 3 left). While user locations regard staypoints of each tracked user separately in the clustering process and prevent generating locations that are excessively large (Aslak & Alessandretti, 2020), dataset locations consider all staypoints at the same time and output locations with shared semantics across users (e.g., train stations or shopping malls). In both options, the center and the extent of the clustered staypoints are attached to the generated locations, providing geometry information that facilitates further processing and analysis tasks.

3.2.3. Generate triplegs

Trackintel implements an algorithm that extracts triplegs from positionfixes based on the assumption that an individual is moving if he or she is not stationary, meaning that all positionfixes that do not belong to any staypoint are assigned to a tripleg. This assignment process requires the input of positionfixes with the identifier of the already generated staypoints. Internally, the function aggregates all positionfixes between two consecutive staypoints to form a tripleg, whose line geometry is constructed by connecting the point geometries in chronological order. Similar to the generation process of staypoints, the start and end timestamp of each tripleg are inherited from the first and last positionfixes that belong to it, respectively.

3.2.4. Generate trips

Trackintel implements a method to generate trips based on existing staypoints and triplegs. Trips summarize all movement and all non-activity staypoints (e.g. depending on the data, this could correspond to waiting at a bus stop) between two staypoints flagged as activity. This seems trivial at first sight however, there are no easy-to-use implementations available in other libraries, and there are several special cases related to gaps in the tracking data that should be considered during the trip generation. Another important feature of the implemented trip generation is the identifier management that connects trips with their associated staypoints and triplegs.

The trip detection implemented in Trackintel can handle incomplete tracking data and supports the detection of temporal gaps (pseudocode shown in Algorithm 1). A temporal gap is defined as missing tracking signals longer than a certain time period (Zhao et al., 2021), which can be specified using the θ_{trip_gap} input parameter to the function. If a temporal gap greater than θ_{trip_gap} is detected, we assume the individual performed an unobserved activity and, therefore, the destination of the current and the origin of the next trip is unknown (NaN in the resulting table). Finally, the function provides the flexibility to specify whether the trips table should include the geometry. The geometry of a trip consists of the points for the origin and destination staypoints. If the origin is unknown, we use the first point of the first tripleg instead, or analogously the last point for the destination.

Algorithm 1. Trip generation.

Given
  s: staypoints of a user
  t: triplegs of a user
  θ_{trip_gap}: Gap threshold to start new trip
procedure GENERATE TRIPS(s, t, θ_{trip_gap})
  $st \leftarrow$ merge s, t and sort by timestamp
  $n =$ length(st)
  $i = 0$
  $\Phi = \{\}$ ▷ Initialize trips as empty set
  inTrip = False
  while $i < n$ **do** ▷ iterate all elements of st
    if (inTrip is False) $\wedge$ ($st[i]$ is no activity) **then** ▷ go to next activity
      i++
      continue with next iteration
    end if
    if (InTrip is False) $\wedge$ ($st[i]$ is activity) $\wedge$ ($i + 1 < n$) $\wedge$ ($st[i + 1]$ is activity) **then**
      i++ ▷ Skip sequential activities w/o travel
      continue with next iteration
    end if
    if InTrip is False **then**
      inTrip = True ▷ Start trip
      initialize new trip $\phi_{current}$
      $\phi_{current}.origin = st[i]$ ▷ set $st[i]$ as origin of trip; add start time, activity label and geometry
      i++
      continue with next iteration
    end if

    δ_t = startTime($st[i]$) $-$ endTime($st[i-1]$) ▷ compute gap duration

    if (InTrip is True) $\wedge$ ($\delta_t \geq \theta_{trip\_gap}$) **then** ▷ check for gap in tracking data
      $\phi_{current}$.destination = unknown ▷ set trip destination as unknown
      $\Phi = \Phi \cup \phi_{current}$ ▷ Add trip to collection of trips
      inTrip = False
    else if (InTrip is True) $\wedge$ (st is no activity) **then**
      add $st[i]$ to trip $\phi_{current}$
    else if (InTrip is True) $\wedge$ (st is activity) **then**
      $\phi_{current}.destination = st[i]$ ▷ set $st[i]$ as trip destination
      $\Phi = \Phi \cup \phi_{current}$ ▷ Add trip to collection of trips
      inTrip = False
    end if
    i++
  end while
end procedure

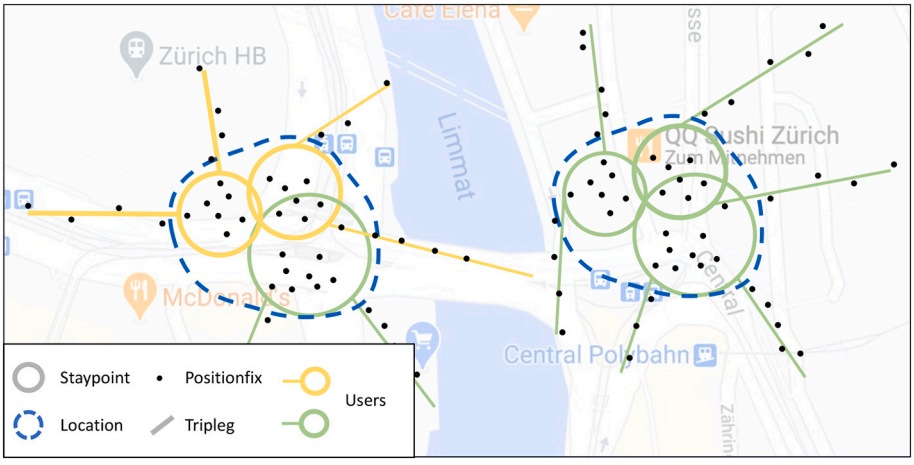


Fig. 3. Semantic visualization of the relations between positionfix, staypoint and locations. Staypoints are groups of positionfixes where the users are stationary, and locations are aggregations of staypoints that the user visits multiple times. Locations can be generated across users (left) or for each user individually (right). Map data ©2022 Google.

3.2.5. Generate tours

To the best of our knowledge, there is no standardized approach yet on how to combine trips into tours. Here, we take a rather broad definition of tours that includes nested tours as described in Axhausen (2007), leaving the user the choice to filter the outputs later. An example of a nested tour is shown in Fig. 4: the tour Work-Cafe-Work is part of the longer tour Home-Work-Cafe-Work-Home. This definition implies an n-to-n relationship between trips and tours: One tour contains multiple trips, and one trip can be part of multiple tours.

Algorithm 2. Tour generation.

```
Given
    Φ : {φ_0, ..., φ_n}: trips (sort by timestamp)
    loc: (Optional) location function that returns the location ID given trip origin or destination
    start: Function that returns the geometry of the start point of a trip
    end: Function that returns the geometry of the end point of a trip
    θ_max_dist: Maximum distance (in m) between the end of one trip and the start of the next trip on the same tour
    θ_max_time: Maximum duration of a tour
    θ_max_gaps: Maximum gaps that are allowed on a tour
procedure GENERATE TOURS(Φ, loc, θ_max_dist, θ_max_time, θ_max_gaps)
    C = [φ_0]: List of candidate trips to form a tour
    i = 1
    locAvail = True if loc is given, otherwise False                ▷ Are locations provided or not?
    while i < n do
                                  ▷ Part 1: Check if the previous and the current trip are connected
        gapInbetween = False
        if locAvail then                                            ▷ Option 1: Compare locations
            if loc(end(φ_{i-1})) ≠ loc(start(φ_i)) then
                gapInbetween = True
            end if
        else                                                        ▷ Option 2: Check distance
            if distance(start(φ_{i-1}), end(φ_i)) > θ_max_dist then
                gapInbetween = True
            end if
        end if
        if gapInbetween then
            C = C.append(gap)                                       ▷ Record a gap in the tour candidates
        end if
        C.append(φ_i)                                               ▷ Add trip to tour candidate
                                  ▷ Part 2: Check if the current trip closes a tour
        for φ_C ∈ C (iterate in reverse order) do
            closesTour = False                                      ▷ Check if φ_C is the start of a tour ending at φ_i
            if locAvail then                                        ▷ Option 1: Compare locations
                if loc(end(φ_i)) == loc(start(φ_C)) then
                    closesTour = True
                end if
            else                                                    ▷ Option 2: Check distance
                if distance(end(φ_i), start(φ_C)) ≤ θ_max_dist then
                    closesTour = True
                end if
            end if
            if closesTour then
                n_gaps = count gaps between φ_C and φ_i
                if ( n_gaps < θ_max_gaps ) ∧ ( endTime(φ_i) − startTime(φ_C) < θ_max_time ) then   ▷ Tour found!
                    Aggregate all trips from φ_C to φ_i into a tour
                end if
            end if
        end for
        Remove trips from C that are more than θ_max_time before φ_i
        i++
    end while
end procedure
```

Our algorithm to generate tours from trips is explained visually in Fig. 4, and shown as pseudocode in Algorithm 2. We iterate over the trips sorted chronologically and maintain a list C of tour-starting candidates. Each trip ϕ_i is a potential candidate to start a tour. At each iteration, that is, for each trip, we first check whether there is a spatial gap between the current and the previous trip ϕ_{i-1}. Two options are implemented: If the table *staypoints* with the attribute `location_id` is provided, we compare the location identifier of the end of ϕ_{i-1} to the one of the start of ϕ_i, formally $loc(end(\phi_{i-1})) = loc(start(\phi_i))$. Alternatively, if the staypoints are not available, the predefined spatial distance threshold θ_{max_dist} controls the maximum distance between the end and start points, i.e. $distance(start(\phi_i), end(\phi_{i-1})) \leq \theta_{max\_dist}$.

Additionally, our implementation offers the possibility to generate partially observed tours to accommodate tracking datasets with a low temporal tracking coverage, e.g., mobile phone data-based studies. A parameter θ_{max_gaps} determines how many spatial gaps are allowed within a single tour. Note that no gaps are allowed at the start or end of a

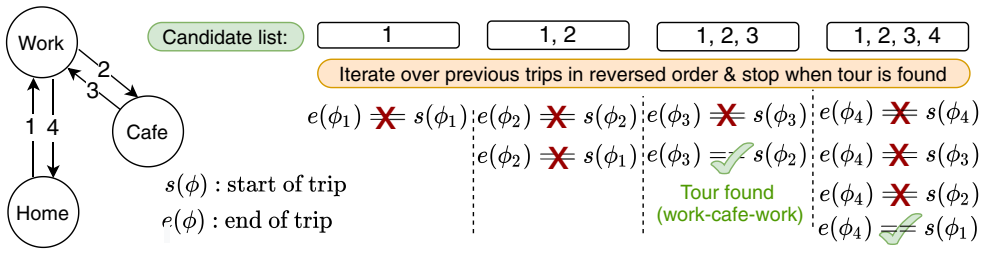


Fig. 4. The algorithm of tour generation implemented in Trackintel. A list of start candidates is maintained and iteratively checked for tour-closing trips.

tour, because a tour must start and end at the same location, or the start- and end-staypoints must lie within the permitted range. If the test described above yields a spatial gap between ϕ_{i-1} and ϕ_i, and $\theta_{max\_gaps} = 0$, the candidate list is reset to $[\phi_i]$. Otherwise, a gap is registered.

Next, we test whether ϕ_i concludes a tour (Algorithm 2, line 15). For this purpose, we iterate over all candidates in the reversed order, such that the shortest possible tour is found first. We compare the start point of a candidate ϕ_C to the end point of ϕ_i. Again, the points are compared either by the location identifier or via the θ_{max_dist} parameter. If they are the same, the trips $\{\phi_k \mid j \leq k \leq i\}$ form a tour, subject to two further conditions: A. While iterating over candidates, the encountered gaps are counted, and the time duration is checked. The parameter θ_{max_time} is used to certify whether the tour takes place within an appropriate time period, by default 24 h. B. When encountering more than θ_{max_gaps} in the reversed iteration, or when reaching a candidate that started more than θ_{max_time} hours ago, the loop ends, and no tour is found. Fig. 4 shows an example where two tours are found after considering ϕ_3 and ϕ_4 respectively.

3.3. Import and export

Reading and writing data are important steps in a standard movement data analysis pipeline. To simplify this process, Trackintel provides an I/O module for accessing movement data and storing intermediate or final results in a file or database. Three methods for converting movement data with attached attribute information to Trackintel-compatible formats are provided: 1) Reading from Pandas Dataframes and Geopandas Geodataframes, 2) reading and writing from CSV file formats, and 3) reading and storing from PostgreSQL databases with PostGIS extension. For every Trackintel data type, we provide I/O functions that internally check the validity of the input data formats. Also, Trackintel implements reading functions to convert tracking data from publicly available open-source datasets into the Trackintel data model. For example, raw tracking records from the Geolife dataset (Zheng et al., 2010) can be loaded with Trackintel into the positionfix format. In addition, we provide helper functions to attach transport mode labels, which are provided separately for some individuals in the Geolife dataset. The dataset reading functions facilitate and standardize the processing of public movement datasets using Trackintel, which also help to benchmark new methods on the same dataset.

3.4. Pre- and Postprocessing

Trackintel offers several pre- and postprocessing methods. First, to smoothen the trajectory of triplegs, we employ the Douglas-Peucker algorithm (Douglas & Peucker, 1973). Furthermore, staypoints that appear consecutively at the same location can be aggregated in time. Such repetitions are a common artefact in tracking data due to noise or outliers recordings in GNSS tracking data. We propose to merge two staypoints s_1, s_2 of one individual if the following conditions hold: a) s_1 and s_2 are consecutive in time, b) s_1 and s_2 are assigned to the same location, c) there is no tripleg registered between s_1 and s_2, and d) the time gap between the end time point of s_1 and the start of s_2 is shorter than a predefined threshold $\theta_{max\_time\_gap}$. The start time of s_1 and the end time of s_2 define the start and end time of the new staypoint. The aggregation of other staypoint attributes, e.g. the geometry, must be specified explicitly.

3.5. Analysis

While the main functionality of Trackintel is the implementation of the hierarchical data model, the framework also includes advanced analysis functions to label transport modes and activity purposes, as well as methods to assess the tracking quality of each individual.

3.5.1. Mode labeling

Applications in transport planning often require access to the travel modes of an individual (Kim, Kim, & Lee, 2022). Since Trackintel does not assume the availability of user-provided labels, context or advanced data from the tracking device (e.g., accelerometer), we implement a simple heuristic to determine the travel mode from the tracking data. This classification is done per *tripleg* based on speed. The speed is approximated by the tripleg length (the distance of individual points in its linestring geometry) divided by its total time duration. The triplegs are labeled based on a simple division into slow mobility (<15 km/h average speed), motorized mobility (<100 km/h) and fast mobility (>100 km/h). In future versions, a more in-depth analysis of travel patterns or map matching (Bachir et al., 2019; Huang et al., 2019; Prelipcean et al., 2017; Widhalm et al., 2012) could be incorporated into Trackintel.

3.5.2. Location labeling

An individual's home- and work-locations play a major role in mobility data analysis. As described in Section 3, staypoints may be associated with an activity label, but oftentimes this information is not available. We assign "home" and "work" activity labels to the staypoints with an adapted version of the OSNA algorithm proposed by Efstathiades, Antoniades, Pallis, & Dikaiakos, 2015. In detail, the OSNA algorithm divides weekdays into rest, work and leisure time frames. The location with the longest accumulated duration in the "rest" and "leisure" periods is labeled as home, while work is set to the most predominant location in the "work" periods. While the original algorithm derives the hours spent at a location from geo-tagged tweets, we take advantage of the *started_at* and *finished_at* attributes of a staypoint. Additionally, similar to in the *R* package proposed by Chen and Poorthuis (2021), we provide a fast method that simply assigns home and work labels to the two locations that are visited more often in the data (in this order). In both cases, the locations can optionally be pre-filtered in order to exclude locations with an insufficient number of staypoints or an insufficient length of stay.

3.5.3. Modal split

If mode labels for *triplegs* are available, Trackintel supports the calculation of the modal split in three different ways: Computing the modal split by count (i.e., how many triplegs with this mode exist), by

duration (i.e., sum of individual's tripleg duration) or by traveled distance. Furthermore, the frequency can be set according to the Pandas time series frequency syntax,[11] and the modes can either be aggregated by user or by dataset. An example for one user is visualized in Fig. 6 where the differences between a modal split by count (Fig. 6a) and by distance (Fig. 6b) stand out.

3.5.4. Tracking quality assessment

An important step in data analysis of tracking studies is the assessment of the tracking quality, i.e. the temporal coverage. Temporal tracking quality, here defined as the proportion of time where the user's whereabouts are recorded, is regarded as a basic measure of the temporal resolution of the dataset (Alessandretti et al., 2018). Trackintel supports the calculation of the daily, weekly or overall tracking quality of each user according to the required granularity levels, which enables individual-level temporal resolution assessment, providing support for filtering low-quality users for further analysis. Additionally, tracking quality of hours of the day and weekdays can be obtained for measuring the tracking data quality differences across time periods.

3.6. Visualization

Trackintel provides a module that supports the visualization of *positionfixes*, *staypoints* and *triplegs*. Our implementation standardizes these functions such that each data type can be displayed together with lower aggregation levels (see Fig. 1). For example, the locations can be optionally shown together with *positionfixes* and *staypoints*. In that case, *staypoints* and *locations* are displayed as circles with a predefined radius. Furthermore, Trackintel integrates osmnx (Boeing, 2017) to optionally show the street network from Open Street Maps as background. Fig. 5 shows example outputs of the plotting functions for *positionfixes*, *staypoints* and *triplegs* for one exemplary participant in the Geolife study.

Finally, Trackintel provides a flexible method to visualize changes in the modal split over time. The modal split by count, distance or duration, as explained in Section 3.5.3, is shown in a bar plot with one bar for each temporal bin. Different temporal resolutions (i.e., weeks and months) are handled internally. An example for one user is shown in Fig. 6 where the modal split has been aggregated by month.

4. A case study on multiple tracking datasets

Trackintel is a framework to standardize mobility data processing and analysis. We carried out a case study on four datasets to demonstrate its capability to handle data from various tracking studies. We read all data from a PostGIS database with the I/O module, preprocess them according to the Trackintel movement data model and compare the datasets in terms of tracking quality, trip characteristics, and modal split. The code of the case study is available in the supplemental material and the public repository.[12]

4.1. Tracking studies

We include the data from four tracking studies with two different tracking data types. An overview of the dataset properties is given in Table 3. The first study is the open-source Geolife dataset (Zheng et al., 2009) that covers the movement of employees of Microsoft Research Asia, who recorded their movement using GPS trackers. Second, we include two studies that were conducted in collaboration with the Swiss Federal Railway Systems (SBB) under the project name *SBB Green Class* (Martin et al., 2019). In both studies, participants were given full access to all public transport in Switzerland. In addition, the participants from the first Green Class study (Green Class 1) received an electric vehicle and those from the second study (Green Class 2) an e-bike. Study participants were tracked with a GNSS-based application (app) called *Myway*.[13] The app already provides the data partially preprocessed as staypoints and triplegs. The same app was further used in our fourth dataset, the yumuv study which investigated the impact of a Mobility-as-a-Service app that integrates shared e-scooters, e-bikes and public transport (Martin et al., 2021). In the yumuv study, participants were divided into control and treatment groups and were tracked for three months.

4.2. Standardized processing according to the Trackintel data model

The Trackintel framework offers a straightforward way to transform all data into the same format and aggregate the data into trips and tours with minimal code. First, the raw GPS data in the Geolife dataset are converted to staypoints and triplegs with the Trackintel `generate_staypoints()` and `generate_triplegs()` functions. Staypoints are created with a distance threshold of 100 m and a temporal threshold of 30 min, i.e. a user must have stayed within a 100 m radius for at least 30 min to generate a new staypoint, as suggested in the original paper (Li et al., 2008). Furthermore, consecutive positionfixes with a temporal gap of >24 h in between cannot belong to the same staypoint.

All further preprocessing steps based on staypoints and triplegs are applied with *the same* parameters for all four datasets. This ensures the comparability of the results across datasets. More specifically, we derive the user's locations from the staypoints with the `generate_locations()` function. The method uses the DBSCAN algorithm with $\varepsilon = 30$ meters and $min\_samples = 1$, such that one staypoint is sufficient to form a location. Furthermore, triplegs and staypoints are aggregated to trips with the `generate_trips()` function, with input parameter $\theta_{trip\_gap} = 25$ minutes. At last, tours are generated by merging trips based on a maximum distance (θ_{max_dist}) of 100 m between their start and end points, and with the default parameters $\theta_{max\_gaps} = 0$ and $\theta_{max\_time} = 24$ hours.

Table 3 provides the absolute numbers of locations, staypoints, triplegs, trips and tours per dataset. These quantities decrease from triplegs to trips and tours due to the aggregation steps. Note that for Geolife our parameter choices prevent triplegs from being merged (see Table 3 where the number of triplegs and trips are the same); however, parameters that are more suitable for the trip generation would have decreased the quality of other parts significantly due to the low tracking quality of Geolife. In total, the considered datasets include 769,957 staypoints and 1,123,931 triplegs. These quantities depend on the number of participants in the study and the total tracking duration. While the yumuv study has the largest sample size of 806 users, the Green Class 1 study participants have the longest tracking period, with each individual tracked for more than a year on average.

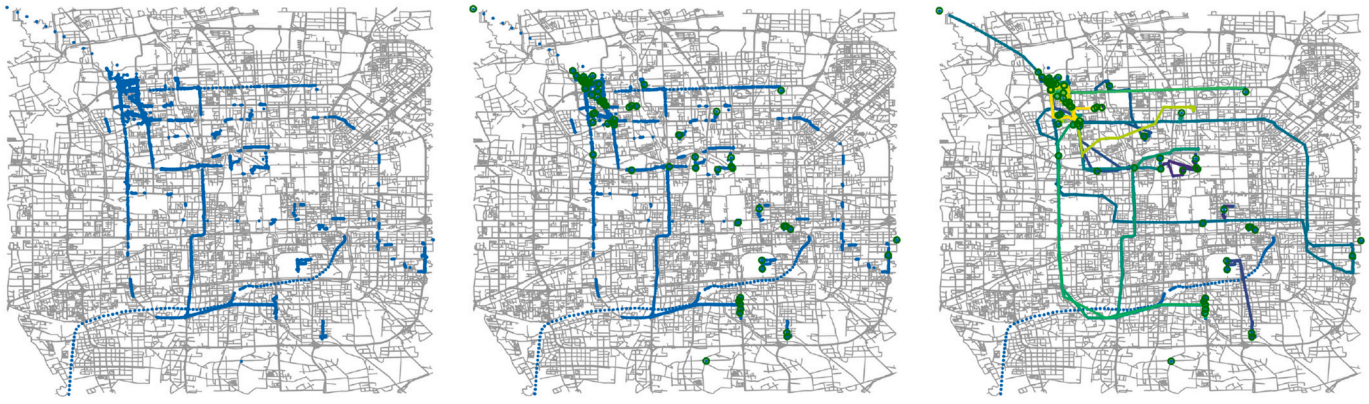
4.3. Analysis and comparison of tracking datasets

We now compare the mobility behavior of the study participants of all studies on the trip level as an exemplary usage of the Trackintel *analysis* module. The insights from this analysis are summarized in Table 4. First, we can derive the number of daily trips per individual from the absolute numbers given above. The study participants in Green Class 1 and Green Class 2 are most active in conducting trips. The low number of trips for Geolife users may be due to the low temporal tracking coverage of the dataset. Furthermore, we compare the average trip distances and duration across datasets. Interestingly, yumuv and Geolife users take longer trips on average in terms of duration. There is

[11] https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html

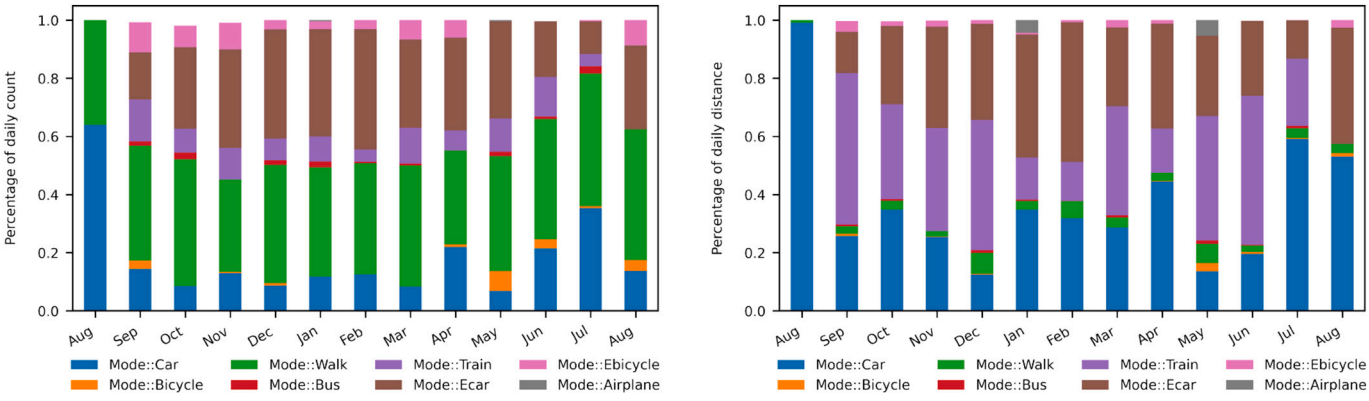
[12] https://github.com/mie-lab/trackintel/blob/master/examples/Trackintel_case_study.pdf

[13] https://www.sbb.ch/en/timetable/mobile-apps/myway.html



(a) Positionfixes (b) Staypoints (c) Triplegs

Fig. 5. The Trackintel framework offers functions to plot positionfixes (a), staypoints (b), and triplegs (c) together with the road network acquired from OpenStreetMaps. This example maps the movements of one Geolife participant.



(a) Modal split by count (b) Modal split by distance

Fig. 6. The visualization result of the Trackintel `plot_modal_split()` function of the triplegs recorded from one Geolife participant. Major differences can be observed between the aggregation by count (number of triplegs) (a) and distance traveled (b).

Table 3
Overview of basic features of the considered tracking studies. Locations, staypoints, triplegs, trips and tours are given in multiples of a thousand.

	Users	Tracking period in days (std)	Input	Study type	Locations	Staypoints	Triplegs	Trips	Tours
Green Class 1	139	401 (59)	Staypoints, Triplegs	GNSS (app)	104.5	326.9	465.2	241.8	95.0
Green Class 2	50	314 (76)	Staypoints, Triplegs	GNSS (app)	35.7	87.9	128.6	61.4	22.7
Yumuv	806	87 (38)	Staypoints, Triplegs	GNSS (app)	127.3	326.3	502.3	199.7	83.0
Geolife	177	193 (443)	Positionfixes	GPS tracker	13.6	28.9	30.2	30.2	7.2

Table 4
Overview of the mobility statistics for the considered tracking datasets.

	Trips per day	Trips per tour	Legs per trip	Trip distance in km (std)	Trip duration (std)	Tracking quality (std)
Green Class 1	4.32	2.73	1.92	27.4 (478.7)	0.52 (0.73)	0.85 (0.17)
Green Class 2	3.80	2.66	2.09	33.7 (568.2)	0.51 (0.75)	0.75 (0.24)
Yumuv	3.13	2.11	2.51	16.9 (100.4)	0.68 (0.91)	0.77 (0.23)
Geolife	1.70	2.37	1.00	36.1 (3163.5)	0.64 (0.94)	0.4 (0.32)

also a clear effect of the bias of yumuv participants towards urban areas, where the trips cover much shorter distances. The number of trips per tour and the number of triplegs that are part of the same trip do not differ much between studies.

Another key part of tracking data analysis regards the temporal tracking quality of a dataset. Here, temporal tracking quality is defined as the temporal coverage of the tracking data (i.e., the completeness) and is computed with the Trackintel function `temporal_tracking_quality()` as explained in Section 3.5.4. The results are given in the last column of Table 4. The three GNSS-based studies show a high coverage of >75% on average per user, whereas Geolife data only covers about 40 % of the time on average per user. Fig. 7 shows the distribution

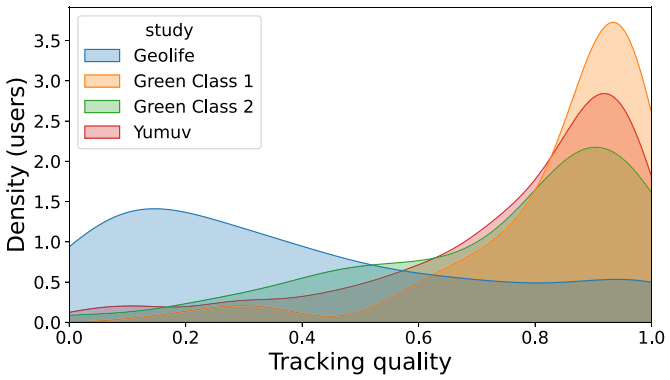
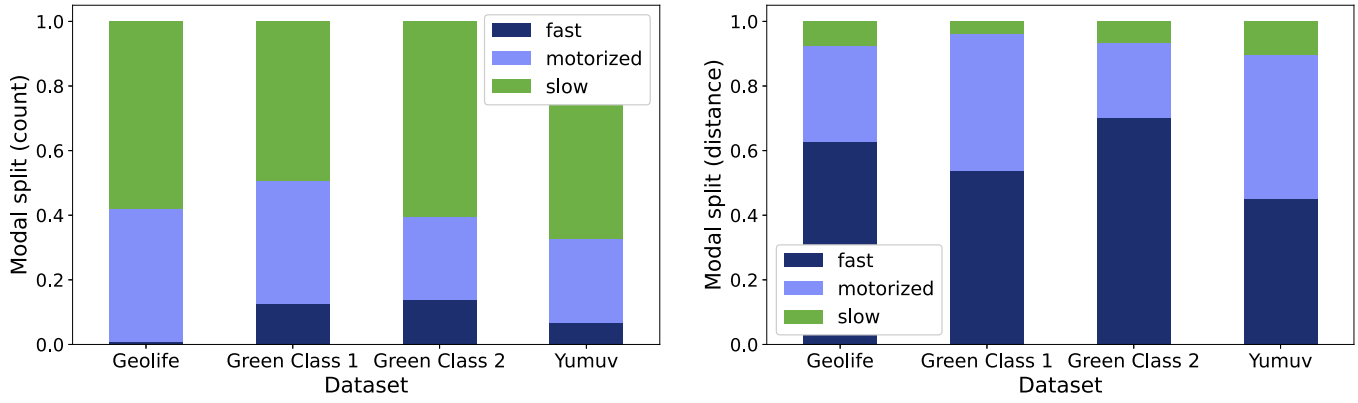


Fig. 7. Distribution of the individual temporal tracking quality for the considered datasets.

of the tracking quality over users. In the Geolife dataset, the temporal tracking quality largely differs across individuals. In comparison, the large majority of Green Class 1 participants reached a coverage of >0.7. The large difference between Geolife and the other datasets can be explained by the different hardware that was used in the studies. While the Geolife individuals were equipped with dedicated GPS-only trackers that are prone to localization problems when indoors or in urban canyons, the participants in the Green Class and yumuv studies were tracked with an app on their smartphone that uses the location API of the operating system. The latter has access to all GNSS systems in addition to GPS and can fall back to other technologies such as WIFI or cell tower triangulation if no satellites are available.

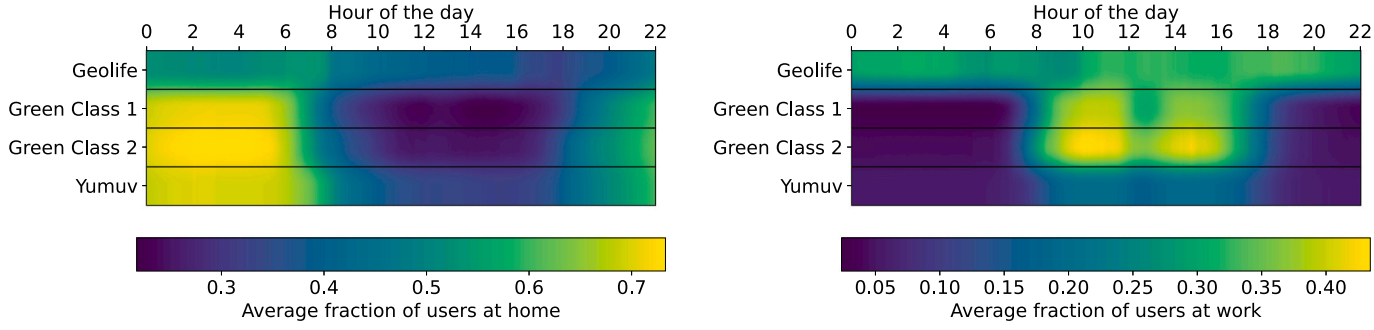
We further compare the modal split of the tracking studies. The split is computed first as the number of triplegs per mode and second as the covered distance per mode. We use the Trackintel function `predict_transport_mode()` to approximate the modes for the Geolife dataset, since the original mode labels are not available for all participants and not all the time. In all other studies, high-quality mode labels are provided, and we aggregate them into the simplified categories of slow mobility (walk, bicycle, scooter), motorized mobility (tram, bus, car and motorbike) and fast mobility (airplane and train). The results are shown in Fig. 8. The datasets differ significantly with respect to their modal split, which can be explained by the study target group, for example, Green Class participants were given full access to all public transport in Switzerland and are thus more likely to use trains (fast transport). Yumuv individuals on the other hand mostly live in urban areas and they were using the yumuv bundle of shared e-bicycles and scooters, which explains the higher proportion of slow mobility for yumuv.

Finally, we analyze the daily activity patterns of individuals. Specifically, the time periods when the individuals are at home and at work are computed. For the Green Class 1 & 2 studies, the activity label for each staypoint is provided by the participants. For the Geolife and yumuv datasets, on the other hand, we adopt the Trackintel `location_identifier()` function that implements the OSNA algorithm (Efstathiades, Antoniades, Pallis, & Dikaiakos, 2015) to infer the home and work locations. In Fig. 9, the distribution of home and work staypoints over the course of a day is shown. Specifically, the average fraction of users with a staypoint labeled home (or work respectively) is shown for every minute of the day. The fraction of users at home (work) is thereby computed as the number of staypoints per day divided by the



(a) Split by count (b) Split by distance

Fig. 8. Comparison of modal split between datasets. The users of different studies differ considerably in terms of their usage of slow, motorized or fast transport.



(a) Home-labeled staypoints (b) Work-labeled staypoints

Fig. 9. Distribution of activities over time.

number of actively tracked users, where a user is actively tracked if there is at least one staypoint on that day. The working time between 8 am and 5 pm as well as the lunch breaks are clearly visible in Fig. 9b for Green Class 1 & 2 and yumuv, although there are fewer work-staypoints for yumuv. While the home location is reliably identified for both yumuv and Geolife, the identification of the work location seems impaired for the Geolife dataset. As the OSNA algorithm simply selects the second-most visited location as work if the "home" and "work" labels overlap, the low tracking quality of the Geolife dataset (see Fig. 7) could have affected the accuracy of the identification.

In summary, our study demonstrates the ease of comparing data from different sources on all levels of the movement data model and concerning various labels for the movement data. The standardized preprocessing functions implemented in Trackintel also help compare methods and explain possible discrepancies in the analysis results from the different datasets.

5. Discussion and conclusion

Quantitative analysis of human mobility currently suffers from a lack of a common model for preprocessing movement data, limiting the reproducibility and comparability of scientific studies. Existing libraries focus on data analysis, leaving seemingly easy preprocessing steps up to the user, although design choices of these steps can significantly affect the results (Sambasivan et al., 2021). This article presented Trackintel, a new open-source tool to address these problems. Trackintel implements a widely accepted conceptual data model for movement data and provides functionalities for the full life-cycle of human mobility data analysis: import and export of tracking data collected through various methods, preprocessing, data quality assessment, semantic enrichment, quantitative analysis and mining tasks, and visualization of data and results.

A particular strength of Trackintel is that it greatly simplifies the joint analysis of several movement datasets with different properties. This was shown in a case study where four different datasets were jointly preprocessed and analyzed. We used the analysis methods implemented in Trackintel to compare the datasets with respect to their trip properties, their tracking quality, their modal split and their daily activity patterns. It was demonstrated in the supplementary material that rich insights about the characteristics of different tracking datasets could be easily obtained in Trackintel with few lines of code. Our library is thus also a response to recent calls in GIS for systematic benchmarking of new methods on several datasets (Konkol et al., 2019).

Importantly, the purpose of Trackintel is not to provide a comprehensive set of analysis functions, but rather a high-quality implementation of standard aggregation and semantics-enrichment steps that are relevant for most tracking studies. This goal is fulfilled in the current version of the library since functions for all aggregation steps in the data movement model are provided and were tested extensively on diverse datasets. Further work on the preprocessing module will focus on improvements, such as outlier filtering functions or methods to fill small gaps in the tracking data.

We plan to extend the analysis functionality of Trackintel and improve the integration with other open-source libraries. Currently, the goal of compatibility with arbitrary tracking datasets limits the capabilities of the analysis model. A good example is the transport mode prediction function provided by Trackintel, which is based on a simple heuristic. A more sophisticated and powerful method can in principle be implemented for a specific dataset, however, the applicability of this method to other datasets will be limited by the availability of specific input data or additional context data. Nevertheless, Trackintel will be continuously extended to incorporate the latest processing and analysis algorithms and to offer a wider variety of options for the preprocessing, analysis and visualization of movement data. In particular, we will work towards the integration of Trackintel with other popular Python libraries, such as the Open Street Maps package osmnx. The data analysis module can be substantially improved when considering mobility-related context information, such as enriching trips with point-of-interest data for transport mode identification. Moreover, we plan to provide a basic behavioral analysis module that allows insights into users' mobility behavior, for example, user mobility profiling and detecting changes in users' mobility behavior over time.

Finally, Trackintel does not aim to cover all preprocessing and analysis needs for every movement data study. However, due to the compatibility with Pandas and Geopandas, Trackintel can easily be integrated into a larger workflow that comprises a variety of Python data and spatial analysis libraries. In particular, it is targeted at providing the same reliability as these standard libraries. This is achieved through strong compliance with Python library standards, including a high coverage of unit tests with both real and synthetic data, a code review process and continuous integration. In this setup, new algorithms can be contributed without risking breaking existing functionality. We therefore believe that Trackintel can serve as a standard and well-trusted mobility processing tool.

CRediT authorship contribution statement

Henry Martin: Conceptualization, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Ye Hong:** Conceptualization, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Nina Wiedemann:** Conceptualization, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Dominik Bucher:** Conceptualization, Methodology, Software, Validation. **Martin Raubal:** Supervision, Funding acquisition, Writing – review & editing.

Acknowledgement

Funding: This work was supported by the Swiss Data Science Center [C17-14] and the ETH Zurich Foundation [MI-01-19]. Additionally, we would like to thank Christof Leutenegger, Sven Ruf, and Nishant Kumar for their code contributions to Trackintel, David Jonietz for helping to create the idea of Trackintel, and René Buffat and Jiří Kunčar for their technical input in the early stage of this project.

Appendix A. Documentation score

A.1. Python

The documentation score reported in Table 1 for python libraries is based on the pyOpenSci package peer-review evaluation critera (Holdgraf et al., 2022).

- Has an Open Software Initiative (OSI) approved license.
- Contains a README with instructions for installing the development version.
- Contains a vignette (notebook) with examples of its essential functions and uses.
- Has a test suite.
- Has continuous integration, such as Travis CI, AppVeyor, CircleCI, and/or others.
- Includes documentation with examples for all functions.

A.2. R

The documentation score reported in Table 1 for R libraries is based on the ROpenScie package peer-review evaluation critera.[4]

- Does the package have a CRAN accepted license?
- The package contains a reasonably complete readme with devtools install instructions.

- The package contains a vignette with examples of its essential functions.
- The package contains unit tests.
- The repository has continuous integration with Travis and/or another service.
- Package available on CRAN?

Appendix B. Case study

The notebook including code to reproduce the case study can be found online at https://doi.org/10.1016/j.compenvurbsys.2023.101938.

References

Ahas, R., Aasa, A., Yuan, Y., Raubal, M., Smoreda, Z., Liu, Y., … Zook, M. (2015). Everyday space–time geographies: Using mobile phone-based sensor data to monitor urban activity in Harbin, Paris, and Tallinn. *International Journal of Geographical Information Science, 29*(11), 2017–2039. https://doi.org/10.1080/13658816.2015.1063151

Alessandretti, L., Sapiezynski, P., Sekara, V., Lehmann, S., & Baronchelli, A. (2018). Evidence for a conserved quantity in human mobility. *Nature Human Behaviour, 2*(7), 485–491. https://doi.org/10.1038/s41562-018-0364-x

Andrew, T. (2014). *Wilson. Tracktable trajectory analysis*. Albuquerque, NM: Sandia National Lab (SNL-NM). https://doi.org/10.11578/dc.20210416.61

Aslak, U., & Alessandretti, L. *Infostop: Scalable stop-location detection in multi-user mobility data*. (2020). *arXiv:2003.14370*.

Axhausen, K. W. (2007). Definition of movement and activity for transport modelling. In *Handbook of transport modelling* (pp. 329–343). https://doi.org/10.3929/ethz-a-005278091

Bachir, D., Khodabandelou, G., Gauthier, V., El Yacoubi, M., & Vachon, E. (2019). Combining Bayesian Inference and Clustering for Transport Mode Detection from Sparse and Noisy Geolocation Data. In *Machine Learning and Knowledge Discovery in Databases* (pp. 569–584). Springer. https://doi.org/10.1007/978-3-030-10997-4_35.

Bassolas, A., Barbosa-Filho, H., Dickinson, B., Dotiwalla, X., Eastham, P., Gallotti, R., … Ramasco, J. J. (2019). Hierarchical organization of urban mobility and its connection with city livability. *Nature Communications, 10*(1), 4817. https://doi.org/10.1038/s41467-019-12809-y

Boeing, G. (2017). Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. Computers, Environment and Urban. *Systems, 65*, 126–139. https://doi.org/10.1016/j.compenvurbsys.2017.05.004

Brockmann, D., Hufnagel, L., & Geisel, T. (2006). The scaling laws of human travel. *Nature, 439*(7075), 462–465. https://doi.org/10.1038/nature04292

Bucher, D., Mangili, F., Cellina, F., Bonesana, C., Jonietz, D., & Raubal, M. (2019). From location tracking to personalized eco-feedback: A framework for geographic information collection, processing and visualization to promote sustainable mobility behaviors. *Travel Behaviour and Society, 14*, 43–56. https://doi.org/10.1016/j.tbs.2018.09.005

Calenge, C. (2006). The package "adehabitat" for the r software: A tool for the analysis of space and habitat use by animals. *Ecological Modelling, 197*(3–4), 516–519. https://doi.org/10.1016/j.ecolmodel.2006.03.017

Calenge, C. (2011). *Analysis of animal movements in r: The adehabitatlt package*. Vienna.: R Foundation for Statistical Computing.

Chang, S., Pierson, E., Koh, P. W., Gerardin, J., Redbird, B., Grusky, D., & Leskovec, J. (2021). Mobility network models of COVID-19 explain inequities and inform reopening. *Nature, 589*(7840), 82–87. https://doi.org/10.1038/s41586-020-2923-3

Chen, C., Ma, J., Susilo, Y., Liu, Y., & Wang, M. (2016). The promises of big data and small data for travel behavior (aka human mobility) analysis. *Transportation Research Part C: Emerging Technologies, 68*, 285–299. https://doi.org/10.1016/j.trc.2016.04.005

Chen, Q., & Poorthuis, A. (2021). Identifying home locations in human mobility data: An open-source R package for comparison and reproducibility. *International Journal of Geographical Information Science, 35*(7), 1425–1448. https://doi.org/10.1080/13658816.2021.1887489

Douglas, D. H., & Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization, 10*(2), 112–122. https://doi.org/10.3138/FM57-6770-U75U-7727

Efstathiades, H., Antoniades, D., Pallis, G., & Dikaiakos, M. D. (2015). Identification of key locations based on online social network activity. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (pp. 218–225). IEEE. https://doi.org/10.1145/2808797.2808877.

Feng, J., Li, Y., Zhang, C., Sun, F., Meng, F., Guo, A., & Jin, D. (2018). DeepMove: Predicting human mobility with attentional recurrent networks. In *Proceedings of the 2018 world wide web conference on world wide web - WWW '18* (pp. 1459–1468). ACM Press. https://doi.org/10.1145/3178876.3186058.

Frick, H., & Kosmidis, I. (2017). trackeR: Infrastructure for Running and Cycling Data from GPS-Enabled Tracking Devices in R. *Journal of Statistical Software, 82*, 1–29. https://doi.org/10.18637/jss.v082.i07

Gillies, S. (2013). The shapely user manual. https://shapely.readthedocs.io/en/stable/manual.html.

González, M. C., Hidalgo, C. A., & Barabási, A.-L. (2008). Understanding individual human mobility patterns. *Nature, 453*(7196), 779–782. https://doi.org/10.1038/nature06958

Graser, A. (2019). MovingPandas: Efficient structures for movement data in Python. *GI_Forum, 1*, 54–68. https://doi.org/10.1553/giscience2019_01_s54

Graser, A. (2020). Tools for the analysis of movement data. https://github.com/anitagraser/movement-analysis-tools.

Haidri, S., Haranwala, Y. J., Bogorny, V., Renso, C., da Fonseca, V. P., & Soares, A. *Ptrail–a python package for parallel trajectory data preprocessing*. (2021). *arXiv:2108.13202*.

Hariharan, R., & Toyama, K. (2004). Project Lachesis: Parsing and Modeling Location Histories. In M. J. Egenhofer, C. Freksa, & H. J. Miller (Eds.), *Geographic Information Science* (pp. 106–124). Springer. https://doi.org/10.1007/978-3-540-30231-5_8.

Huang, H., Cheng, Y., & Weibel, R. (2019). Transport mode detection based on mobile phone network data: A systematic review. *Transportation Research Part C: Emerging Technologies, 101*, 297–312. https://doi.org/10.1016/j.trc.2019.02.008

Huang, H., Gartner, G., Krisp, J. M., Raubal, M., & Van de Weghe, N. (2018). Location based services: Ongoing evolution and research agenda. *Journal of Location Based Services, 12*(2), 63–93. https://doi.org/10.1080/17489725.2018.1508763

Joo, R., Boone, M. E., Clay, T. A., Patrick, S. C., Clusella-Trullas, S., & Basille, M. (2020). Navigating through the R packages for movement. *Journal of Animal Ecology, 89*(1), 248–267. https://doi.org/10.1111/1365-2656.13116

Keßler, C., & McKenzie, G. (2018). A geoprivacy manifesto. *Transactions in GIS, 22*(1), 3–19. https://doi.org/10.1111/tgis.12305

Holdgraf, C., Solvik, K., Ogasawara, I., Brett, M., Sundell, E., gaow, Chen, Z., … Kashyap, S. (2022). *pyOpenSci/contributing-guide: Pre release 0.3 (v0.3)*. Zenodo. https://doi.org/10.5281/zenodo.7101778

Hong, Y., Xin, Y., Martin, H., Bucher, D., & Raubal, M. (2021). A Clustering-Based Framework for Individual Travel Behaviour Change Detection. In *, 208. 11th International Conference on Geographic Information Science (GIScience 2021) - Part II*. https://doi.org/10.4230/LIPIcs.GIScience.2021.II.4, 4.

Jordahl, K., Bossche, J.V. den, Fleischmann, M., McBride, J., Wasserman, J., Richards, M., Badaracco, A.G., Snow, A.D., Gerard, J., Tratner, J., Perry, M., Ward, B., Farmer, C., Hjelle, G.A., Taves, M., Hoeven, E. ter, Cochran, M., rraymondgh, Gillies, S., … Ren, C. (2022). *geopandas/geopandas: V0.12.2* (v0.12.2). Zenodo. https://doi.org/10.5281/zenodo.7422493.

Jonietz, D., & Bucher, D. (2018). *Continuous trajectory pattern mining for mobility behaviour change detection. LBS 2018: 14th international conference on location based services*. In (pp. 211–230). Springer. https://doi.org/10.1007/978-3-319-71470-7_11.

Kim, J., Kim, J. H., & Lee, G. (2022). GPS data-based mobility mode inference model using long-term recurrent convolutional networks. *Transportation Research Part C: Emerging Technologies, 135*, 103523. https://doi.org/10.1016/j.trc.2021.103523

Konkol, M., Kray, C., & Pfeiffer, M. (2019). Computational reproducibility in geoscientific papers: Insights from a series of studies with geoscientists and a reproduction study. *International Journal of Geographical Information Science, 33*(2), 408–429. https://doi.org/10.1080/13658816.2018.1508687

Li, Q., Yu, Z., Xie, X., Chen, Y., Liu, W., & Ma, W.-Y.. Mining user similarity based on location history. https://doi.org/10.1145/1463434.1463477.

Lovelace, R., & Ellison, R. (2018). stplanr: A Package for Transport Planning. *The R Journal, 10*(2), 7–23. https://doi.org/10.32614/RJ-2018-053

Luca, M., Barlacchi, G., Lepri, B., & Pappalardo, L. (2021). A survey on deep learning for human mobility. *ACM Computing Surveys, 55*(1). https://doi.org/10.1145/3485125

Luo, T., Zheng, X., Guangluan, X., Kun, F., & Ren, W. (2017). An improved DBSCAN algorithm to detect stops in individual trajectories. *ISPRS International Journal of Geo-Information, 6*(3), 63. https://doi.org/10.3390/ijgi6030063

Martin, H., Becker, H., Bucher, D., Jonietz, D., Raubal, M., & Axhausen, K. W. (2019). *Begleitstudie SBB Green Class - Abschlussbericht*. Working Paper No. 1439, Institute for Transport Planning and Systems, ETH Zürich. https://doi.org/10.3929/ethz-b-000353337.

Martin, H., Reck, D. J., Axhausen, K. W., & Raubal, M. (2021). *ETH mobility initiative project MI-01-19 empirical use and impact analysis of MaaS*. ETH Zurich: Technical report.

Moro, E., Calacci, D., Dong, X., & Pentland, A. (2021). Mobility patterns are associated with experienced income segregation in large US cities. *Nature Communications, 12*(1), 4633. https://doi.org/10.1038/s41467-021-24899-8

Pappalardo, L., Simini, F., Barlacchi, G., & Pellungrini, R. (2022). scikit-mobility: A Python Library for the Analysis, Generation, and Risk Assessment of Mobility Data. *Journal of Statistical Software, 103*(1), 1–38. https://doi.org/10.18637/jss.v103.i04

Pappalardo, L., Simini, F., Rinzivillo, S., Pedreschi, D., Giannotti, F., & Barabási, A.-L. (2015). Returners and explorers dichotomy in human mobility. *Nature Communications, 6*(1), 8166. https://doi.org/10.1038/ncomms9166

Prelipcean, A. C., Gidófalvi, G., & Susilo, Y. O. (2017). Transportation mode detection–an in-depth review of applicability and reliability. *Transport Reviews, 37*(4), 442–464. https://doi.org/10.1080/01441647.2016.1246489

Qing, Y., & Yuan, J. (2022). Transbigdata: A python package for transportation spatio-temporal big data processing, analysis and visualization. *Journal of Open Source Software, 7*(71), 4021. https://doi.org/10.21105/joss.04021

Rhee, I., Shin, M., Hong, S., Lee, K., Kim, S. J., & Chong, S. (2011). On the levy-walk nature of human mobility. *IEEE/ACM Transactions on Networking, 19*(3), 630–643. https://doi.org/10.1109/TNET.2011.2120618

Rout, A., Nitoslawski, S., Ladle, A., & Galpern, P. (2021). Using smartphone-GPS data to understand pedestrian-scale behavior in urban settings: A review of themes and approaches. *Computers, Environment and Urban. Systems, 90*, 101705. https://doi.org/10.1016/j.compenvurbsys.2021.101705

Sambasivan, N., Kapania, S., Highfill, H., Akrong, D., Paritosh, P., & Aroyo, L. M. (2021). Everyone wants to do the model work, not the data work": Data Cascades in High-

Stakes AI. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1–15). https://doi.org/10.1145/3411764.3445518

Schneider, C. M., Belik, V., Couronné, T., Smoreda, Z., & González, M. C. (2013). Unravelling daily human mobility motifs. *Journal of The Royal Society Interface, 10* (84), 20130246. https://doi.org/10.1098/rsif.2013.0246

Schönfelder, S., & Axhausen, K. W. (2016). *Urban rhythms and travel behaviour: Spatial and temporal phenomena of daily travel.* Routledge. https://doi.org/10.4324/9781315548715

Shenk, J., Byttner, W., Nambusubramaniyan, S., & Zoeller, A. (2021). Traja: A python toolbox for animal trajectory analysis. *Journal of Open Source Software, 6*(63), 3202. https://doi.org/10.21105/joss.03202

Smolak, K., Siła-Nowicka, K., Delvenne, J.-C., Wierzbiński, M., & Rohm, W. (2021). The impact of human mobility data scales and processing on movement predictability. *Scientific Reports, 11*(1), 15177. https://doi.org/10.1038/s41598-021-94102-x

Solomon, A., Livne, A., Katz, G., Shapira, B., & Rokach, L. (2021). Analyzing movement predictability using human attributes and behavioral patterns. *Computers, Environment and Urban Systems, 87*, Article 101596. https://doi.org/10.1016/j.compenvurbsys.2021.101596

Zheng, Y., Zhang, L., Xie, X., & Ma, W.-Y.. Mining interesting locations and travel sequences from GPS trajectories. https://dl.acm.org/doi/10.1145/1526709.1526816.

Zimányi, E., Sakr, M., & Lesuisse, A. (2020). Mobilitydb: A mobility database based on postgresql and postgis. *ACM Transactions on Database Systems, 45*(4), 1–42. https://doi.org/10.1145/3406534

Song, C., Koren, T., Wang, P., & Barabási, A.-L. (2010). Modelling the scaling properties of human mobility. *Nature Physics, 6*(10), 818–823. https://doi.org/10.1038/nphys1760

Song, C., Qu, Z., Blumm, N., & Barabasi, A.-L. (2010). Limits of Predictability in Human Mobility. *Science, 327*(5968), 1018–1021. https://doi.org/10.1126/science.1177170

The pandas development team. (2023). *pandas-dev/pandas: Pandas (v1.5.3)*. Zenodo. https://doi.org/10.5281/zenodo.7549438

Toch, E., Lerner, B., Ben-Zion, E., & Ben-Gal, I. (2018). Analyzing large-scale human mobility data: A survey of machine learning methods and applications. *Knowledge and Information Systems*. https://doi.org/10.1007/s10115-018-1186-x

Urner, J., Bucher, D., Yang, J., & Jonietz, D. (2018). Assessing the influence of spatio-temporal context for next place prediction using different machine learning approaches. *ISPRS International Journal of Geo-Information, 24*. https://doi.org/10.3390/ijgi7050166

Widhalm, P., Nitsche, P., & Brändie, N. (2012). Transport mode detection with realistic smartphone sensor data. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)* (pp. 573–576). IEEE.

Xu, Y., Çolak, S., Kara, E. C., Moura, S. J., & González, M. C. (2018). Planning for electric vehicle needs by coupling charging profiles with urban mobility. *Nature Energy, 3*(6), 484–493. https://doi.org/10.1038/s41560-018-0136-x

Yuan, N. J., Zheng, Y., Zhang, L., & Xie, X. (2013). T-finder: A recommender system for finding passengers and vacant taxis. *IEEE Transactions on Knowledge and Data Engineering, 25*(10), 2390–2403. https://doi.org/10.1109/TKDE.2012.153

Yuan, Y., & Raubal, M. (2012). Extracting Dynamic Urban Mobility Patterns from Mobile Phone Data. In N. Xiao, M.-P. Kwan, M. F. Goodchild, & S. Shekhar (Eds.), *Geographic Information Science* (pp. 354–367). Springer. https://doi.org/10.1007/978-3-642-33024-7_26.

Zhao, P., Jonietz, D., & Raubal, M. (2021). Applying frequent-pattern mining and time geography to impute gaps in smartphone-based human-movement data. *International Journal of Geographical Information Science*, 1–29. https://doi.org/10.1080/13658816.2020.1862126

Zheng, Y. (2015). Trajectory data mining: An overview. *ACM Transactions on Intelligent Systems and Technology (TIST), 6*(3), 1–41. https://doi.org/10.1145/2743025

Zheng, Y., Xie, X., Ma, W.-Y., et al. (2010). Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Engineering Bulletin, 33*(2), 32–39.