Workflow of processing Cuebiq data with each module explained

Overview

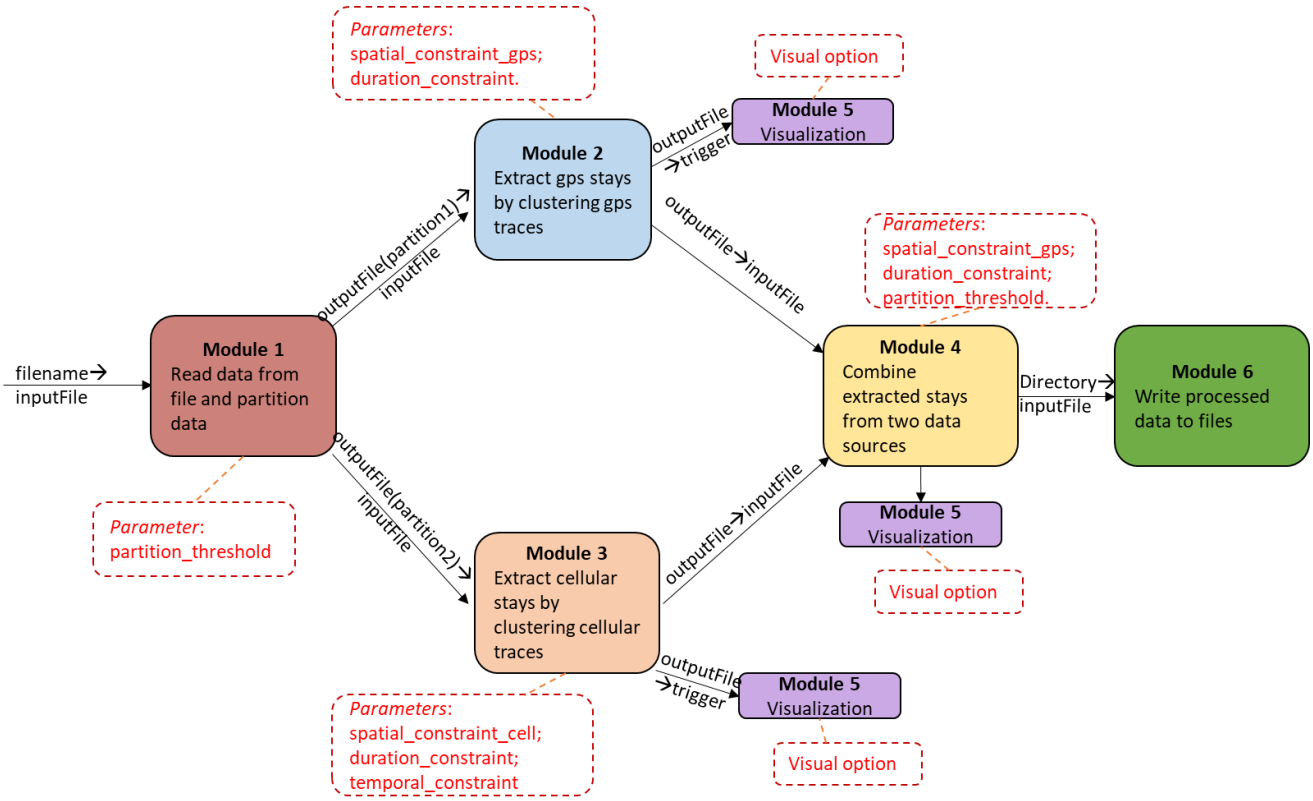


Figure 1. An overview of the workflow.

Figure 1 gives an overview of the workflow. The workflow is to extract trips from app-based data following the “Divide, Conquer and Integrate” (DCI) framework proposed by (Wang et al. 2019).

The DCI framework consists of three steps: (1) partitioning data, (2) extracting trips from each partition, and (3) integrating trips that are extracted from each partition. The first step aims to partition a multi-sourced dataset into multiple data sets such that each one contains small variances in terms of temporal and spatial properties. The second step is to extract trips from each data set independently using methods that are designed according to the unique characteristics of the data set. Inspired by how a geographic information system (GIS) manages and organizes data from multiple sources, the third step treats extracted trips in each data set as a layer with each trip as a feature in the layer and integrates these trips by joining layers according to the spatiotemporal relationship of their features.

The workflow includes the following six modules, including

- Module 1: read data from file and partition data
- Module 2: extract GPS stays by clustering GPS traces
- Module 3: extract cellular stays by clustering cellular traces
- Module 4: combine extracted stays from two data sources

- Module 5: visualization
- Module 6: write processed data to files

The dashed box close to each module gives parameters of the module. Module 1 implements the first step of the DCI framework, which partitions the data into two parts, with one containing GPS traces and the other containing cellular traces. The second step is implemented by Module 2 and 3, which extract GPS stays and cellular stays independently. Module 4 implements the third step, integrating GPS and cellular stays. Besides implementing the DCI framework, we design Module 5 and Module 6 to visualize intermediate and final results and to write them to files, respectively.

In the following, we introduce each module, including its input, parameters, and output. Both the module for extracting GPS stays (i.e., Module 2) and the module for extracting cellular stays (i.e., Module 3) contain sub-modules, each of which is also introduced in detail.

Modules

Module 1. Read data from files and partition data

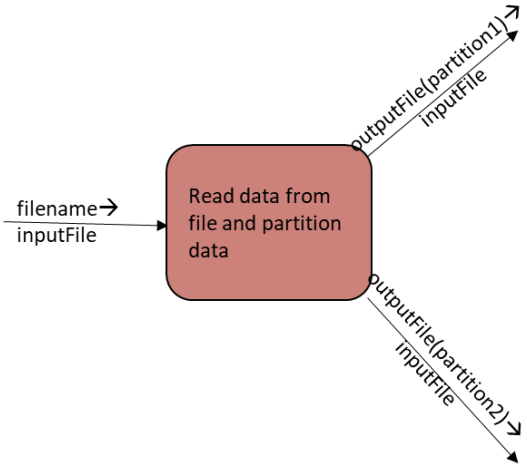


Figure 2 The module for reading in and partitioning data

This module is to read in a user's records and partition them into two or more parts. One part would include app-based data with their uncertainty radius larger than 100 meters, and the other part would include the rest of the records (radius no larger than 100 meters). Here, we partition records into two partitions based a threshold on the uncertainty radius *partition_threshold* (i.e., 100 meters). The unit of uncertainty radius is in meters, measuring the accuracy of the location record. A smaller uncertainty radius means higher location accuracy. The data is partitioned such that each partition will be passed independently to the next modules for extract stays.

Input:

The inputs are files where data are stored. An input file is in CSV format and contains records of users to be processed (Figure 3). Each row gives a record of a user containing seven fields, including *Timestamp (in Unix time), ID, ID_type, Latitude, Longitude, Uncertainty_radius, Human_time*. Here, *uncertainty_radius* measures the accuracy of the location record in meters.

Human_time is a string in the format of “year-month-date-hour-minute-second.” It is converted from *Timestamp* in Unix time for the convenience of following data processing. For example, a string ‘*190519183020’* is converted from Unix time *1558315820,* and it stands for 18:30:20 on May 19, 2019. The first line of an input file gives the names of the seven fields. Figure 4 gives a snapshot records in an input file.

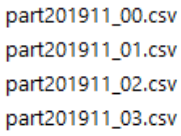


Figure 3 A snapshot of input files of the module for reading in and partitioning data

Timestamp	ID	ID_Type	Latitude	Longitude	Uncertainty	Human_time
1588778435	3dd3bc7f67	1	47.660252	-122.316416	60	200506082035
1588778495	3dd3bc7f67	1	47.660133	-122.316347	60	200506082135
1588778555	3dd3bc7f67	1	47.660046	-122.316384	60	200506082235
1588778615	3dd3bc7f67	1	47.659981	-122.316304	60	200506082335
1588778675	3dd3bc7f67	1	47.66006	-122.316604	60	200506082435
1588778735	3dd3bc7f67	1	47.659937	-122.316475	60	200506082535
1588778795	3dd3bc7f67	1	47.659995	-122.316288	60	200506082635
1588779515	3dd3bc7f67	1	47.659815	-122.314496	30	200506083835
1588779575	3dd3bc7f67	1	47.659797	-122.313198	30	200506083935
1588779635	3dd3bc7f67	1	47.659797	-122.313112	30	200506084035
1588779695	3dd3bc7f67	1	47.659797	-122.312066	30	200506084135
1588779755	3dd3bc7f67	1	47.659753	-122.310049	30	200506084235
1588779815	3dd3bc7f67	1	47.659565	-122.307173	30	200506084335
1588779875	3dd3bc7f67	1	47.658792	-122.307624	30	200506084435
1588779935	3dd3bc7f67	1	47.65747	-122.307946	30	200506084535
1588779995	3dd3bc7f67	1	47.656182	-122.309321	30	200506084635
1588780055	3dd3bc7f67	1	47.653401	-122.307242	30	200506084735
1588786220	3dd3bc7f67	1	47.651771	-122.30443	300	200506103020
1588786820	3dd3bc7f67	1	47.651885	-122.305015	300	200506104020
1588787420	3dd3bc7f67	1	47.651413	-122.304214	300	200506105020
1588788020	3dd3bc7f67	1	47.651988	-122.304237	300	200506110020
1588788620	3dd3bc7f67	1	47.651814	-122.304472	300	200506111020

Figure 4 A snapshot of records in an input file of the module for reading in and partitioning data

Parameters:

partition_threshold: The only parameter of the module is the threshold (in meters) of using uncertainty radius as the partition criteria. For example, given *partition_threshold=100* meters, one can partition the raw records of one user into two parts, one consisting of records whose uncertainty radii are not larger than 100 meters and the other containing the rest of records.

The parameter is a change point of the module, as it could be changed to a different threshold of uncertainty radius.

Output:

The outputs are the partitioned data, where each partition gives a part of input records. Given app-based data as input, the output would be two files, with one containing records with uncertainty radius larger than 100 meters (referred to as cellular data) (Figure 5A) and the other

containing the records with uncertainty radius no larger than 100 meters (referred to as GPS data) (Figure 5B).

Timestamp	ID	ID_Type	Latitude	Longitude	Uncertainty	Human_time
1588778435	3dd3bc7f67	1	47.660252	-122.316416	60	200506082035
1588778495	3dd3bc7f67	1	47.660133	-122.316347	60	200506082135
1588778555	3dd3bc7f67	1	47.660046	-122.316384	60	200506082235
1588778615	3dd3bc7f67	1	47.659981	-122.316304	60	200506082335
1588778675	3dd3bc7f67	1	47.66006	-122.316604	60	200506082435
1588778735	3dd3bc7f67	1	47.659937	-122.316475	60	200506082535
1588778795	3dd3bc7f67	1	47.659995	-122.316288	60	200506082635
1588779515	3dd3bc7f67	1	47.659815	-122.314496	30	200506083835
1588779575	3dd3bc7f67	1	47.659797	-122.313198	30	200506083935
1588779635	3dd3bc7f67	1	47.659797	-122.313112	30	200506084035
1588779695	3dd3bc7f67	1	47.659797	-122.312066	30	200506084135
1588779755	3dd3bc7f67	1	47.659753	-122.310049	30	200506084235
1588779815	3dd3bc7f67	1	47.659565	-122.307173	30	200506084335
1588779875	3dd3bc7f67	1	47.658792	-122.307624	30	200506084435
1588779935	3dd3bc7f67	1	47.65747	-122.307946	30	200506084535
1588779995	3dd3bc7f67	1	47.656182	-122.309321	30	200506084635
1588780055	3dd3bc7f67	1	47.653401	-122.307242	30	200506084735

(A) One partition of records of one user with uncertainty radius no larger than 100 meters

Timestamp	ID	ID_Type	Latitude	Longitude	Uncertainty	Human_time
1588786220	3dd3bc7f67	1	47.651771	-122.30443	300	200506103020
1588786820	3dd3bc7f67	1	47.651885	-122.305015	300	200506104020
1588787420	3dd3bc7f67	1	47.651413	-122.304214	300	200506105020
1588788020	3dd3bc7f67	1	47.651988	-122.304237	300	200506110020
1588788620	3dd3bc7f67	1	47.651814	-122.304472	300	200506111020

(B) One partition of records of one user with uncertainty radius larger than 100 meters

Figure 5 Snapshots of the outputs of the module for reading in and partitioning data

Module 2. Extract GPS stays

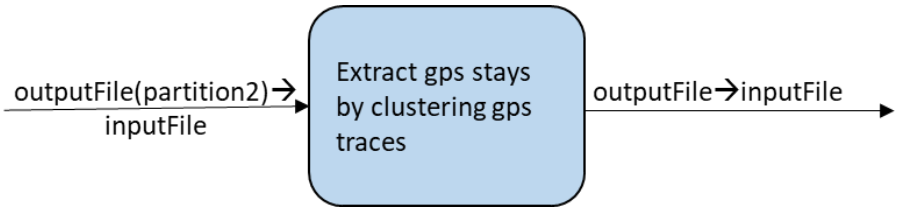


Figure 6 Module for extracting GPS stays

This module is to cluster GPS data of one user (one partition of the raw records with uncertainty radius smaller than 100 meters) into stays or identifying it as a transient point, using function *clusterGPS*. We show how this module works by introducing the functions of its three sub-modules in the following.

Input:

The input is a partition of the records, consisting of a part of the outputs from Module 1 *read and partition data*. Figure 5B gives a snapshot of example input. It contains gps records of one user ID (with uncertainty radius smaller than 100 meters). The input records are structured in a dictionary containing records of all days for one user. The key of the dictionary gives a date and

the value is a list containing time-ordered GPS records on the date. As the same as Module 2 *extract cellular stays*, a GPS record includes seven fields, including Timestamp, ID, ID_type, Latitude, Longitude, Uncertainty_radius, Human_time.

Parameters:

- *spatial_constraint_gps*: the spatial constraints for clustering GPS records to extract GPS stays in meters.
- *temporal_constraint*: the temporal constraints for clustering GPS records to extract GPS stays in seconds.

A GPS stay is a cluster of records satisfying both pre-specified spatial and temporal constraints, following the definition of a GPS stay. For example, a GPS stay can be defined as a cluster of records where the distance between any two records within the cluster is less than 200 meters, and the time duration of the cluster is longer than 300 seconds. More details on how the constraints work in clustering GPS records can be found in (Wang et al., 2019).

The two parameters could be changed according to how a GPS stay is defined.

Output:

The output gives processed GPS records of the input user ID. The data structure is similar as the input: a dictionary containing records of all days for one user with a key being a date and its value being a list of processed GPS records on the date.

As shown in Figure 7, five more fields are added to the original (input) data to store information of extracted GPS stays. For each record, if it is clustered and belongs to a stay, we add to the record five fields, including the latitude and longitude of the stay location (i.e., the centroid of the cluster), the stay duration (in seconds), the stay uncertainty radius (in meters) and a label of the stay (currently it serves as a placeholder and will be filled at Module 4). If a record is not clustered and does not belong to any stay, the four fields are filled with "-1", representing a "passing-by" record. Figure 7 gives a snapshot of an example output, showing an extracted GPS stay consisting of eight records and two "passing-by" records.

Timestamp	ID	ID_Type	Latitude	Longitude	Uncertainty	Stay_lat	Stay_long	Stay_unc	Stay_dur	Stay_label	Human_time
1588778435	3dd3bc7f67	1	47.660252	-122.316416	60	47.660496	-122.3157499	60	1080	-1	200506082035
1588778495	3dd3bc7f67	1	47.660133	-122.316347	60	47.660496	-122.3157499	60	1080	-1	200506082135
1588778555	3dd3bc7f67	1	47.660046	-122.316384	60	47.660496	-122.3157499	60	1080	-1	200506082235
1588778615	3dd3bc7f67	1	47.659981	-122.316304	60	47.660496	-122.3157499	60	1080	-1	200506082335
1588778675	3dd3bc7f67	1	47.66006	-122.316604	60	47.660496	-122.3157499	60	1080	-1	200506082435
1588778735	3dd3bc7f67	1	47.659937	-122.316475	60	47.660496	-122.3157499	60	1080	-1	200506082535
1588778795	3dd3bc7f67	1	47.659995	-122.316288	60	47.660496	-122.3157499	60	1080	-1	200506082635
1588779515	3dd3bc7f67	1	47.659815	-122.314496	30	47.660496	-122.3157499	60	1080	-1	200506083835
1588779575	3dd3bc7f67	1	47.659797	-122.313198	30	-1	-1	-1	-1	-1	200506083935
1588779635	3dd3bc7f67	1	47.659797	-122.313112	30	-1	-1	-1	-1	-1	200506084035

Figure 7 A snapshot of the output of the module for extracting GPS stays

As shown in Figure 8, Module 2 consists of three sub-modules, including trace-segmentation clustering, Incremental clustering, and update stay duration. In the following, we introduce the purpose, the input, parameters, and the output of each sub-module.

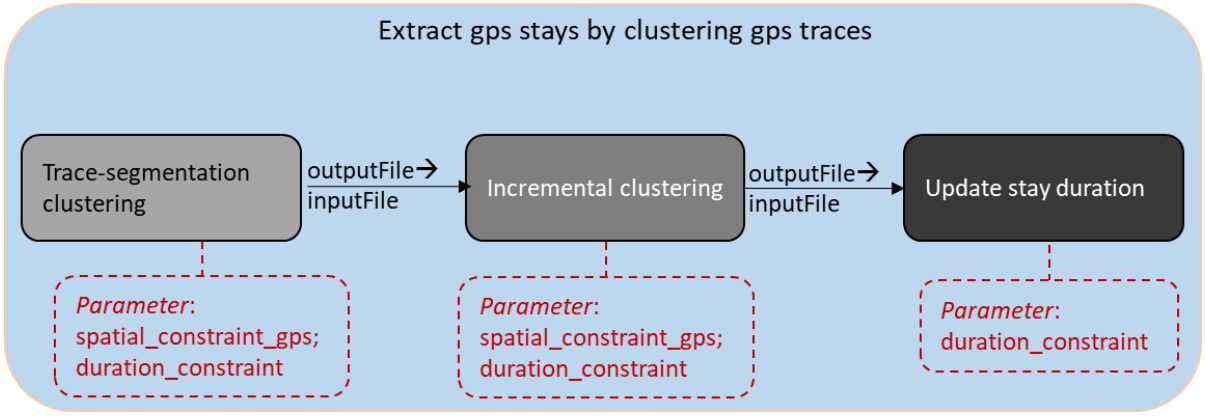


Figure 8 Sub-module for extracting GPS stays

Sub-module 2.1. Trace-segmentation clustering

This sub-module is to cluster raw GPS locations of one user to identify GPS stays, using function *cluster_traceSegmentation*. The function scans through the user's trajectories to identify where the user stayed for a while to do activities. A GPS stay is a cluster of records that satisfy both spatial and temporal constraints.

A commonly-used trace-segmentation method is applied to extract stays from the GPS set. For each trajectory of one user, we extract stays by scanning through the trajectory and segmenting it into multiple sequences of observations with two parameters: signal roaming distance *spatial_constraint_gps* and stay duration *duration_constraint*. A stay is extracted as a sequence of observations satisfying both thresholds: the distance between any two observations in the sequence should be shorter than *spatial_constraint_gps* and the duration (i.e., the time difference between the last and the first observation of this sequence) must be no less than *duration_constraint*. Intuitively, *spatial_constraint_gps* defines the maximum distance that the user could stray from a location; *duration_constraint* defines the minimum duration that the user has to be within the roaming distance to qualify as staying at that location. In our study, we use 200 meters and five minutes as *spatial_constraint_gps* and *duration_constraint*, respectively. The algorithm of trace-segmentation clustering can be found in Figure 16 of (Wang et al., 2019).

Note that following this module stays identified at different times are distinct in their latitude and longitude coordinates. This means a common location (e.g., home location) that is visited on different days is in different coordinates, preventing further mobility analyses such as identifying home and work locations. Therefore, after applying this module, Module *incremental clustering* will be applied (detailed in the next sub-module) to cluster locations of stays on multiple days, such that common stays can be identified.

Input:

The input is raw GPS records of one user ID, which are one partition of the raw records with their uncertainty radii no larger than 100 meters (Figure 5A). The input data is structured in a python dictionary containing records of multiple days for one user. A key and its value of the dictionary give a date and the raw GPS records on the date, respectively.

Parameters:

- *spatial_constraint_gps:* the spatial constraint of the clustering method in meters (e.g., 200 meters).
- *duration_constraint:* the temporal constraint of the clustering method in seconds (e.g., 300 seconds).

Both the two parameters can be changed according to the definition of a GPS stay. For example, if one defines a stay as a cluster of records with time duration no less than 600 seconds, then *duration_constraint* should be set as 600.

Output:

The output gives potential stays in GPS records of the input user ID. Similar to input data, the output data is structured in a python dictionary. A key gives a date, and its value gives a list of integrated, clustered records on the date.

Each record itself is a list of length 11, including the six original data fields and another five fields storing information of clustered GPS stays. For each record belonging to a stay, the additional five fields give the latitude and longitude of the stay location (i.e., the centroid of the cluster), the stay duration (in seconds), the stay uncertainty radius (in meters) and a placeholder of the stay label. If a record is not clustered and does not belong to any stay, the four fields are filled with "-1". Visits to the same location at different times are extracted as distinct stays with different latitudes and longitudes, and therefore the output here will be input to the next sub-module to identify common stays.

The map in Figure 9A shows raw GPS trajectories of two days of one user. After applying this module, two stays are extracted (Figure 9B) and the rest of the records do not belong to any cluster and thus passing-by records. The map shows that a location visited on two days are likely to be distinct stays.

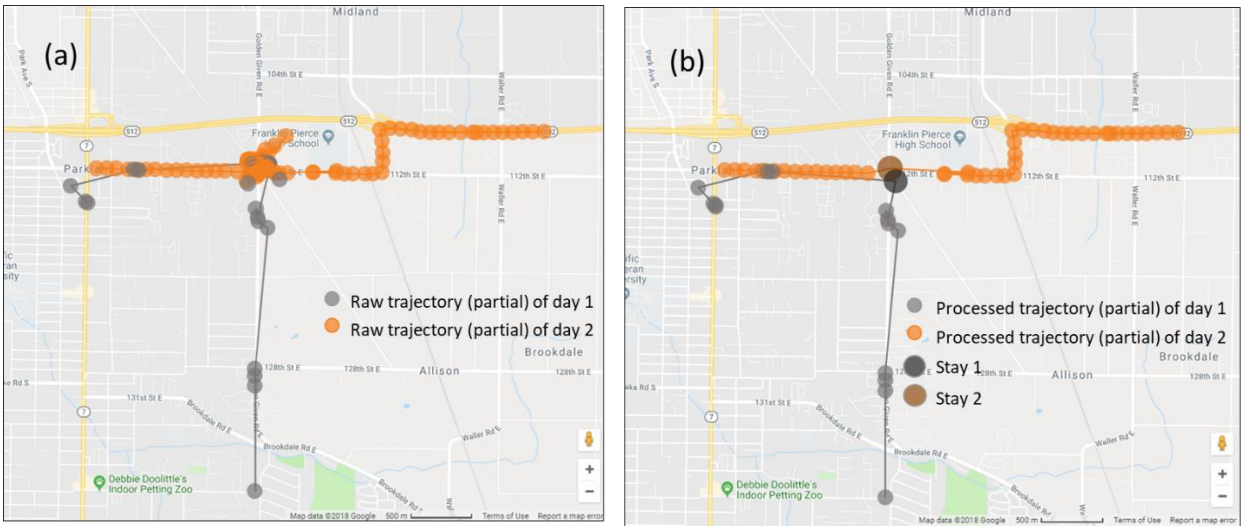


Figure 9 Illustration of identifying stays from the GPS data. (a) Raw GPS trajectories of two days before applying trace-segmentation method; (b) Processed trajectories with identified stays after applying trace-segmentation method.

Sub-module 2.2. Incremental clustering

This sub-module here is to cluster locations of GPS stays identified from sub-module *Trace-segmentation clustering*, so that *common stays* visited multiple times (e.g., home location) can be identified. It applies function *cluster_incremental*.

A common stay represents a place that is associated with multiple visits by one user at different times. Following the trace-segmentation clustering in the last module, extracted GPS stays at different times are unique in their longitude and latitude coordinates, as each stay is represented by the centroid of a unique cluster. To identify those common stays, we ignore the temporal scale of stays and apply another clustering method to aggregate stays close in space. Specifically, we put together all stays identified in one user's trajectories and aggregate those close in space into one cluster. Then, the location of a stay is modified using the centroid of the cluster where the stay belongs. In this workflow, the incremental clustering method is applied with the spatial constraint *spatial_constraint_gps*. A brief introduction of the incremental clustering is introduced in the following.

The incremental clustering gathers all cellular observations belonging to one user in a list. Then, the list is clustered without regarding their time ordering by repeating the following three steps:

1) Starting from a record d_1, one new cluster C_1 is created and d_1 is the center;
2) Each record that has not been clustered will be checked and the one within a distance *spatial_constraint_gps* to the center of C_1 is clustered into C_1 and the center of C_1 is correspondingly updated;
3) If no observation could be aggregated in the current cluster, one new cluster is created containing a non-clustered observation.

This clustering method returns a set of clusters, each of which contains observations that are close in space.

This module can be used for two purposes by specifying parameter *duration_constraint*, which is the definition of a stay (e.g., 300 seconds). First, by setting a positive value to *duration_constraint,* it clusters locations of stays on multiple days to identify common stays, which are needed for some mobility analysis, such as identifying home and work locations. Second, it clusters locations in raw cellular records to identify stays by setting *duration_constraint* to *None*. Here, to identify common stays visited on multiple days, we cluster locations of GPS stays and thus pursue the first purpose. Later, in Module 3 (*extracting cellular stays)*, the same sub-module is applied but for the second purpose.

Input:

The input here is the output of sub-module *Trace-segmentation clustering* (Figure 7), which gives clustered GPS records of one user ID. In these records, stays at different times are distinct in their latitude and longitude coordinates.

The input data is structured in a python dictionary containing records of multiple days for one user. A key and its value of the dictionary give a date and the raw GPS records on the date, respectively. Each record itself is a list of length 11. A record is either a transient point or belongs to a GPS stay.

Parameters:

There are two parameters:

- *spatial_constraint_gps:* the spatial constraint of the clustering method in kilometers (e.g., 0.2 kilometers)
- *duration_constraint:* the temporal constraint of the clustering method in seconds (e.g., 300 seconds).

The two parameters are specified based on how a common GPS stay is defined. For example, *spatial_constraint_gps* =0.2 means that a common GPS stay is a cluster of stays with the distance of any stay within the cluster to the cluster center is shorter than 0.2 kilometers. As noted above, to cluster locations of stays instead of locations of raw records, the temporal constraint "*duration_constraint*" is specified to overwrite its default value—*None*.

Both the two parameters can be changed according to the definition of a GPS stay. For example, if one defines a common GPS stay as a cluster of stays with the distance of any stay within the cluster to the cluster center is shorter than 0.5 kilometers, *spatial_constraint_gps* shall be set as 0.5 (kilometers).

Output:

The output is clustered GPS stays of the input user ID where common stays are identified. A common stay that is visited at different times now has the same latitude and longitude coordinates. The sub-module overwrites the input data by modifying some records directly, and therefore the output data structure is the same as input data.

Figure 10 illustrates that after applying the module, two distinct stays on two days are clustered and identified as one common stay.

Figure 10 Illustration of identifying common stays from the GPS data. (A): Processed trajectories with identified stays following sub-module trace-segmentation clustering; (B): Processed trajectories with a common stay being identified following sub-module incremental clustering.

Sub-module 2.3. Update stay duration

This sub-module is to update the duration of stays in the records of a user, using function *update_duration*. The sub-module is applied following any procedure involving modifying records. Note that the duration of a stay is defined as the time difference between the first

and the last record with in a cluster. After applying incremental clustering as the previous sub-module, it is possible that some clusters may be changed in terms of records in the clusters. Therefore, this module is implemented to re-calculate duration for stays, taking as input clustered GPS records of a user (i.e., the output of the previous sub-module *incremental clustering*).

Input:

The input is the output of the previous sub-module *incremental clustering*, containing clustered GPS records of a user. Each record is either a transient point or a stay. The input data is structured in a python dictionary. A key and its value of the dictionary give a date and the records on the date, respectively. Figure 7 shows a snapshot of example input.

Parameters:

The only parameter is the temporal constraint (*duration_constraint*) that defines a stay, as explained above.

Output:

The output contains records of the input user ID where durations of stays are updated. The sub-module directly overwrites the input data, and thus the data structure is the same as the input.

Module 3. Extract cellular stays

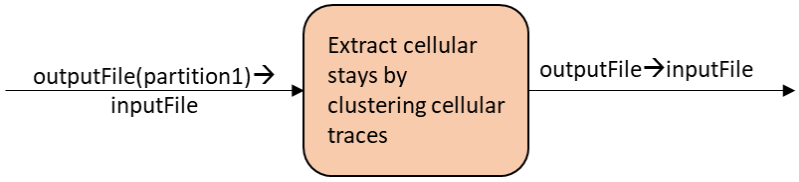


Figure 11 Module for extracting cellular stays

This module is to cluster cellular data of one user (containing records with uncertainty radius larger than 100 meters) into stays or identify it as a transient point, using function *clusterPhone*. This module contains a sub-flow with three sub-modules that are introduced in the following.

Input:

The input is a part of records resulting from Module 1 *read and partition data*, which contains cellular data of one user ID (with *uncertainty_radius* larger than 100 meters) (Figure 5B). The input records are structured in a dictionary containing records of all days for one user. The key of the dictionary gives a date, and the value is a list containing time-ordered cellular records on the date. A cellular record includes seven fields, including *Timestamp (in Unix time), ID, ID_type, Latitude, Longitude, Uncertainty_radius, Human_time*.

Parameters:

There are three parameters in the module, including *spatial_constraint_cell, duration_constraint* and *temporal_constraint*.

- *spatial_constraint_cell* and *duration_constraint* specify the spatial (in kilometers) and temporal constraints (in seconds) for clustering cellular records to extract stays. A stay is

a cluster of records satisfying the pre-specified spatial and temporal constraints. For example, by setting *spatial_constraint_cell* as 1 kilometer and *duration_constraint* as 300 seconds, a cellular stay is defined as a cluster of records, where the distance between any record in the cluster and the cluster center is less than 1 kilometer and the time duration of the cluster is longer than 300 seconds. More details on the role of the constraints in clustering cellular records can be found latter as we introduce the sub-modules. The two parameters can be changed according to the definition of a cellular stay. For example, if one defines a cellular stay as a cluster of records where the distance between any record in the cluster and the cluster center is less than 500 meters, *spatial_constraint_cell* shall be changed to 0.5 kilometers.

- *temporal_constraint* (in seconds) is used to identify and remove oscillation traces. The algorithm identifying and removing oscillation traces using time-window based method is introduced in (cite Wang and Chen 2018), where *temporal_constraint* serves as the length of the time window. This parameter can also be changed according to the criteria for identifying and removing oscillation traces. A larger value will impose stricter criteria, and consequently, more records are identified as oscillation traces and removed.

Output:

The output gives processed cellular records of the input user ID. The data structure is similar to the input: a dictionary containing records of all days for one user with a key being a date and its value being a list of processed cellular records on the date.

Similar to the output of Module 2 (extracting GPS stays), five more fields are added to the original (input) data to store information of extracted cellular stays (Figure 12). For each record, if it is clustered and belongs to a cellular stay, we add to the record the latitude and longitude of the stay location (i.e., the centroid of the cluster), the stay duration (in seconds), the stay uncertainty radius (in meters) and a label of the stay (currently it serves as a placeholder and will be filled at Module 4). If a record is not clustered and does not belong to any stay, the four fields are filled with "-1", representing a "passing-by" record. Figure 12 gives a snapshot of an example output, showing an extracted cellular stay consisting of five records and one "passing-by" record.

Timestamp	ID	ID_Type	Latitude	Longitude	Uncertainty	Stay_lat	Stay_long	Stay_unc	Stay_dur	Stay_label	Human_time
1588786220	3dd3bc7f67	1	47.651771	-122.30443	300	47.6524318	-122.3040553	397	2400	-1	200506103020
1588786820	3dd3bc7f67	1	47.651885	-122.305015	300	47.6524318	-122.3040553	397	2400	-1	200506104020
1588787420	3dd3bc7f67	1	47.651413	-122.304214	300	47.6524318	-122.3040553	397	2400	-1	200506105020
1588788020	3dd3bc7f67	1	47.651988	-122.304237	300	47.6524318	-122.3040553	397	2400	-1	200506110020
1588788620	3dd3bc7f67	1	47.651814	-122.304472	300	47.6524318	-122.3040553	397	2400	-1	200506111020
1588796845	3dd3bc7f67	1	47.65562	-122.28339	2225	-1	-1	-1	-1	-1	200506132725

Figure 12 A snapshot of the output of the module for extracting cellular stays

As shown in Figure 13, Module 3 implements four sub-modules. In the following, we introduce their purposes, inputs, parameters, and outputs.

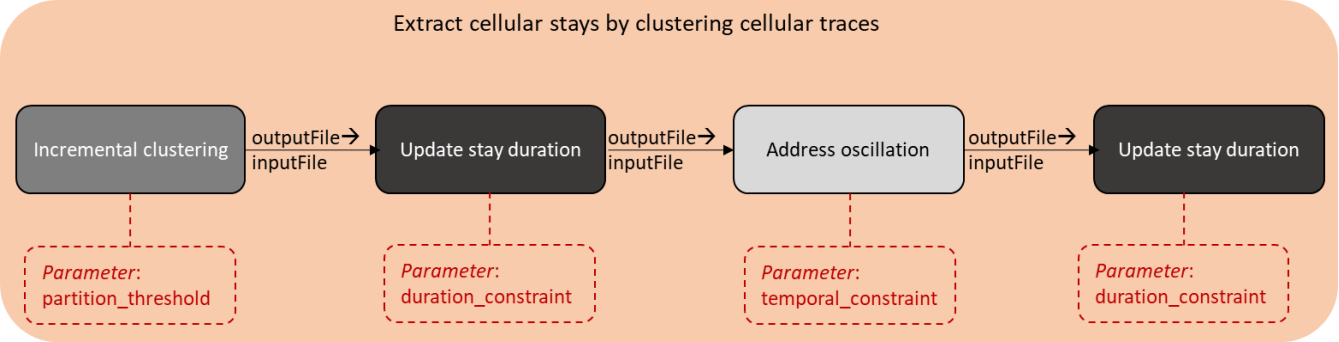


Figure 13 Sub-modules in the module for extracting cellular stays

Sub-module 3.1. Incremental clustering

This sub-module is to cluster raw cellular locations of one user on multiple days to identify cellular stays. The function used is *cluster_incremental*, which clusters locations on multiple days based on a specified spatial threshold such that each outputting cluster of locations represents a potential stay.

As mentioned earlier, this module can be used for two purposes by specifying parameter *duration_constraint.* Here, the module is used for the second purpose by setting *duration_constraint* as *None*.

Input:

The input is cellular records of one user ID, which are the partition of the raw records with uncertainty radius no less than 100 meters. The input data is structured in a python dictionary containing records of multiple days for one user. A key and its value of the dictionary give a date and the records on the date, respectively. Figure 5B shows a snapshot of example input.

Parameters:

There are two parameters:

- *spatial_constraint_gps:* the spatial constraint of the clustering method in kilometers (e.g., 0.2 kilometers)
- *duration_constraint:* the temporal constraint defining the duration of a stay.

As noted above, to use the first option (i.e., clustering locations in raw cellular records to identify stays), the temporal constraint *duration_constraint* is set as *None*. The parameter *spatial_constraint_cell* specifies how a cellular stay is defined. For example, *spatial_constraint_cell* =1 means that a cellular stay is a cluster of records where the distance between any record in the cluster and the cluster center is less than one kilometer.

The parameter *spatial_constraint_cell* can be changed according to the definition of a cellular stay.

Output:

The output gives potential stays in cellular records of the input user ID. Similar to input data, the output data is structured in a python dictionary. A key gives a date, and its value gives a list of integrated, clustered records on the date.

Each record itself is a list of length 11, including the six original data fields and another five fields storing information of clustered cellular stays. For each record belonging to a stay, the additional five fields give the latitude and longitude of the stay location (i.e., the centroid of the cluster), the stay duration (in seconds), the stay uncertainty radius (in meters) and a placeholder of the stay label. If a record is not clustered and does not belong to any stay, the four fields are filled with "-1".

The map in Figure 14 gives an example output, showing two cellular stays extracted from cellular records.

Figure 14 Map with extracted cellular stays. Hollow and filled dots represent raw records and extracted cellular stays, respectively.

Sub-module 3.2. Update stay duration

This sub-module is the same as the one in Module 2 (i.e., extracting GPS stays), using the same function *update_duration*. As noted earlier, the sub-module is applied following any procedure involving modifying records of stays. Therefore, here the sub-module takes clustered cellular records of a user (i.e., the output of the previous sub-module *incremental clustering*) to update the duration of cellular stays.

After applying incremental clustering as the previous sub-module, it is possible that some clusters may be changed in terms of records in the clusters. Therefore, this module is implemented to re-calculate duration for stays, taking as input clustered cellular records of a user (i.e., the output of the previous sub-module *incremental clustering*).

Input:

The input is the output of the previous sub-module *incremental clustering*, containing clustered cellular records of a user. The input data is structured in a python dictionary, where

each key and its value give a date and the records on the date, respectively. A record is either a transient point or a cellular stay. Figure 12 shows a snapshot of example input.

Parameters:

The only parameter is the temporal constraint *duration_constraint* that defines a stay, as explained above. The parameter can be changed according to the definition of a stay.

Output:

The output contains records of the input user ID where durations of stays have been updated if needed. The sub-module directly overwrites the input data, and thus the data structure is the same as the input.

Sub-module 3.3. Address oscillation

This sub-module is to address oscillation traces in one user's trajectory, using the function "*oscillation_h1_oscill*". In the following, we briefly introduce the oscillation phenomenon and how the module identifies oscillation records.

Oscillation traces lead to false trips rather than reflecting users' actual movements and thus need to be removed. For example, some devices may be observed switching between two (or more) faraway locations with high frequencies, and the switching speed (if calculated) could be incredibly high (e.g., several hundreds of miles per hour). In order to avoid these false trips, the module implements a time-window-based method (Wang and Chen, 2018) to detect and remove observations that are generated from the occurrence of the signaling noises. It scans trajectories with a short time window (five minutes), which always starts at the last observation of a stay and returns a sequence of observations. Among the sequences returned, the one containing at least one circular event is considered as a noise sequence, and trips in the noise sequence are removed. Here, a circular event refers to a tour when one device is initially observed at one location, later jumps to other distinct locations, and then comes back to the initial location. Since it is less likely for any user to take a tour within a short time window, the noise sequences detected may contain observations generated from signaling noises and therefore are removed. Readers are referred to Wang and Chen (2018) for more details on the time-window-based method.

Input:

The input is the clustered records of one user, where a record is either a transient point or a stay. The input data is structured in a python dictionary. A key and its value of the dictionary give a date and the records on the date, respectively.

Parameters:

The only parameter is a temporal constraint *temporal_constraint*. It is set as 300 seconds and used as the length of the time window for identifying oscillation traces. The algorithm identifying and removing oscillation traces using time-window based method is introduced in (Wang and Chen 2018).

The parameter can be changed. A larger value will impose stricter criteria, and more records are identified as oscillation traces and removed.

Output:

The output contains records of the input user ID with oscillation traces removed. The data structure is the same as the input.

The maps in Figure 15 show processed an example trajectory before and after removing false trips resulting from oscillation traces. Before removing oscillation traces in Figure 15A, there is a false stay s_3 identified in Portage Bay area (marked in red dot), and frequent jumps can be observed between stay s_2 and s_3. After removing oscillation traces, stay s_3 no longer exists and the trajectory is corrected (Figure 15B).

Figure 15 Maps showing before (A) and after (B) removing false trips resulting from oscillation traces. Hollow dots are the raw records, filled dots are the extracted stays, and the red filled dot (s3 in (A)) represents the false stay resulting from oscillation traces.

Module 4. Combining extracted stays from two data sources

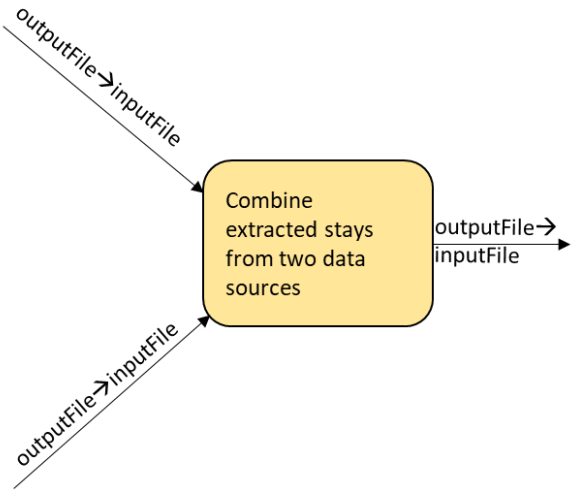


Figure 16 Module for combining extracted stays from two data sources

This module is to combine processed cellular records (output of module *extract cellular stays*) and processed GPS records (output of module *extract GPS stays*) of one user ID, using function *combineGPSandPhoneStops*. The module implements an algorithm proposed by (Wang et al., 2019), which is briefly introduced in the following.

The algorithm is designed based on concepts of space-time relationship analyses in Geographic Information System (GIS). Geometrics in multiple layers/data sources can be spatially and temporally related in different ways, including spatially/temporally contained and adjacent. These geometrics are joint by determining what type of relations they represent. For example, counts of two adjacent polygons in the same period can be integrated. Similarly, the algorithm proposed treats each data set as a layer and identify stays as features in the layer. The time and location information (i.e., centroid and radius) of each stay act as the temporal and spatial attributes, respectively. Then, features (i.e., stays) from multiple layers (i.e., data sets) are combined by measuring their spatiotemporal relationship based on their temporal and spatial attributes. The temporal relationship is defined in three categories, including temporally separate, temporally contained, and temporally intersected. For the spatial relationship, two stays are defined spatially contiguous if the difference of their stay radii is greater than their spatial distance. For the app-based data, we use the predominant GPS data set (representing about 85% of the app-based data) as the basis. Then, for each of the cellular stay in a cellular trajectory of one user, we check its spatiotemporal relationship with the processed GPS trajectory of the user (observations of the same user on the same day) and decide how to combine it into the GPS trajectory. For example, if a cellular stay is temporally separate and not spatially contiguous with GPS stays, it is added to the GPS trajectory as a new stay. Examples of spatiotemporal relationships and more details of how the integration algorithm works can be found in (Wang et al., 2019).

Input:

The input includes two parts of the processed records of one user. One is from the output of Module 2 (*extract GPS stays*), containing GPS stays of the user (Figure 7). The other part is

from the output of Module 3 (*extract cellular stays*), containing cellular stays of the user (Figure 12). Both parts of data are structured in python dictionaries, with a key and its value providing a date and the records on the date, respectively. A record is either a transient point or a stay.

Parameters:

- *partition_threshold:* the threshold used for partitioning the raw records based on attribute uncertainty radius (e.g., 100 meters) in module *read and partition data*.
- *temporal_constraint:* minimum time duration (e.g., 300 seconds) that defines a stay.
- *spatial_constraint_gps:* the spatial constraint that defines a GPS stay (e.g., 0.2 kilometers).

The parameters could be changed, and they should be consistent with the values of corresponding parameters in the previous modules. For example, if parameter *spatial_constraint_gps* is 0.2 kilometers in Module 2 (*extract GPS stays*), it should be 0.2 kilometers here as well.

Output:

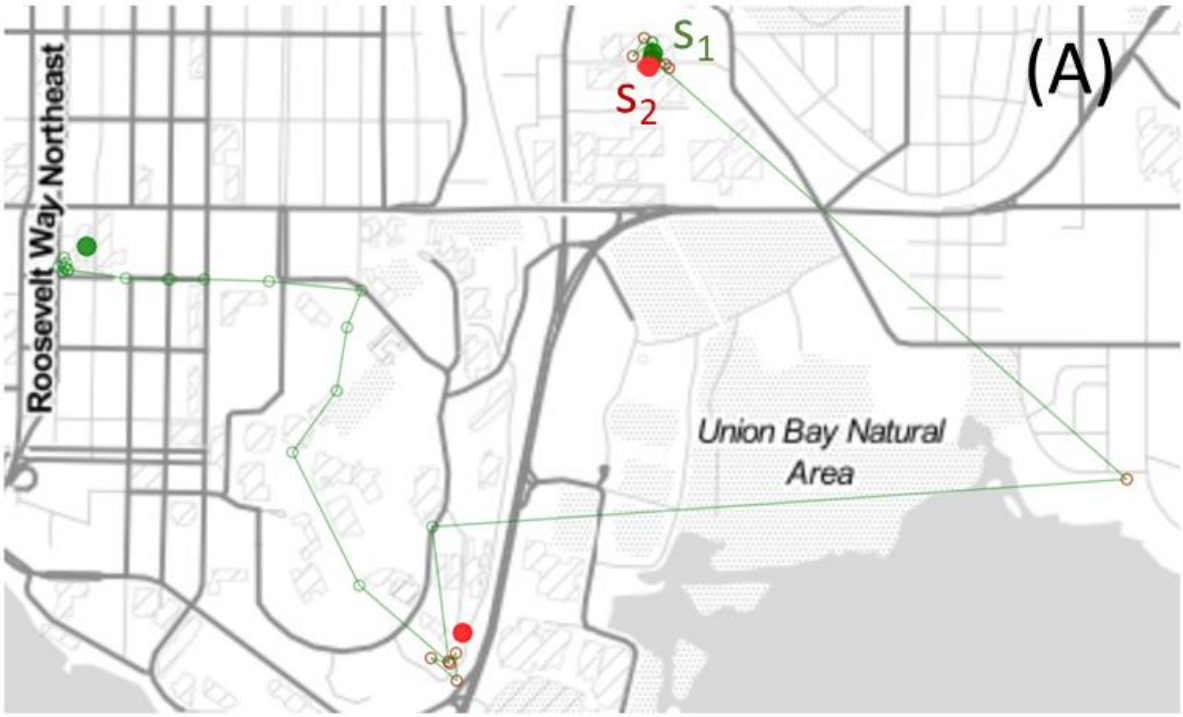
The output gives combined records of the input user ID. As the same as the input data, the output data is structured in a python dictionary. A key gives a date, and its value gives a list of records on the date. Each record has 12 fields, where stay information such as latitude, longitude, duration, uncertainty radius are included if the record is a stay.

Figure 17 gives a snapshot of an example output showing two stays with one from GPS stays and one from cellular stay. Each processed record in the output file could represent one of the two: 1) a stay (either from cellular stays or GPS stays before the integration) with latitude, longitude, uncertainty radius, and positive stay duration; 2) a transient point (a passing-by record) which is the same as a record in the original raw input file with stay duration marked as -1.

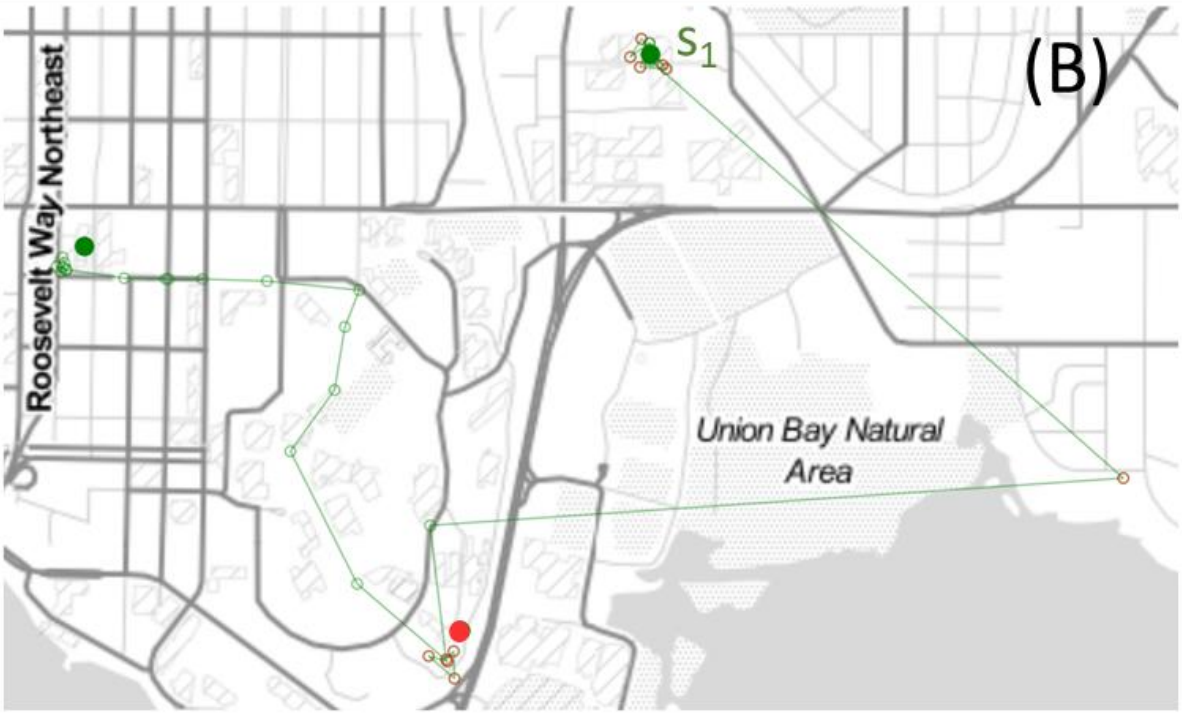
Figure 18 shows an example trajectory before and after combining GPS and cellular stays. Before implementing Module 4, there are two stays from GPS stays and another two from cellular stays. After implementing Module 4, GPS stay s_1 and cellular stay s_2 are combined as one, as they represent the same visit to a place. The resultant trajectory shows only one cellular stay.

Unix_start_t	ID	ID_Type	Orig_lat	Orig_long	Orig_unc	Stay_lat	Stay_long	Stay_unc	Stay_dur	Stay_label	Human_start_t
1588778435	3dd3bc7f67	1	47.660252	-122.316416	60	47.660496	-122.3157499	60	1080	stay3	200506082035
1588779575	3dd3bc7f67	1	47.659797	-122.313198	30	-1	-1	-1	-1	-1	200506083935
1588779635	3dd3bc7f67	1	47.659797	-122.313112	30	-1	-1	-1	-1	-1	200506084035
1588779695	3dd3bc7f67	1	47.659797	-122.312066	30	-1	-1	-1	-1	-1	200506084135
1588779755	3dd3bc7f67	1	47.659753	-122.310049	30	-1	-1	-1	-1	-1	200506084235
1588779815	3dd3bc7f67	1	47.659565	-122.307173	30	-1	-1	-1	-1	-1	200506084335
1588779875	3dd3bc7f67	1	47.658792	-122.307624	30	-1	-1	-1	-1	-1	200506084435
1588779935	3dd3bc7f67	1	47.65747	-122.307946	30	-1	-1	-1	-1	-1	200506084535
1588779995	3dd3bc7f67	1	47.656182	-122.309321	30	-1	-1	-1	-1	-1	200506084635
1588780055	3dd3bc7f67	1	47.653401	-122.307242	30	-1	-1	-1	-1	-1	200506084735
1588786220	3dd3bc7f67	1	47.651771	-122.30443	300	47.6524318	-122.3040553	397	2400	stay0	200506103020

Figure 17 A snapshot of the output of the module for combining extracted stays from two data sources



(A) before combining GPS and cellular stays



(B) after combining GPS and cellular stays

Figure 18 An example trajectory before and after implementing Module 4 to combine GPS and cellular stays. Hollow dots are the raw records, and filled dots are the extracted stays with GPS

stays in green and cellular stays in red. After implementing Module 4, GPS stay s_1 and cellular stay s_2 are combined.

Module 5. Visualization

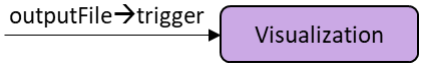


Figure 19 Module for visualization

This module is for visualization purposes. Either an intermediate result or the final processed records can be taken as the input of visualization. Potential visualizations include maps visualizing users' trajectories or some statistics measuring specific properties of records of users.

Input:

The input could be either an intermediate result or the final processed records of users. For example, the input could be a sequence of records of a randomly selected user.

Parameters:

An example parameter would be the number of trajectories to be visualized, which properties of users' records to be visualized.

The parameters can be changed to indicate what properties to visualize and how.

Output:

The output could be maps of trajectories of randomly selected users, distributions plots of properties of users' records. Figure 20 shows a map of a random user's trajectory.

Figure 20 Map of an example trajectory.

Module 6. Writing processed data to file

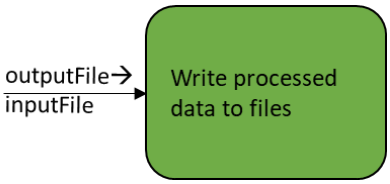


Figure 21 Module for writing processed data to file

This module is to write the processed records of one user to file. One could specify which file and the file type of file to write.

Input:

The input is the output of Module 4 (*combine extracted stays from two data sources*), containing the processed records of one user (Figure 17). The input data is structured in a python dictionary, where a key gives a date, and its value gives a list of records on the date.

Parameters:

The parameters include the name and type of a file where the data should be written. A common file type is a CSV file. The parameters can be changed. For example, one could specify the name and type of the output file.

Output:

An output file contains processed records of the input user. If it is a CSV file, each row would be one record of the user. When specifying the file name where to write, if the file name already exists, the records of the user are appended to the end of the file, resulting in more than one user in the file.

trip_identified_part201911_00.csv
trip_identified_part201911_01.csv
trip_identified_part201911_02.csv
trip_identified_part201911_03.csv

Figure 22 A snapshot of output files of the module for writing processed data to file