Westfälische Wilhelms-Universität Münster - Institut für Geoinformatik (ifgi)
University of Münster - Institute for Geoinformatics (ifgi)

Bachelorarbeit - Bachelor Thesis

Meaningful aggregation of trajectory data in R

Roland Harhoff

15.1.2015

Advisor:
Prof. Edzer Pebesma
Dr. Benedikt Gräler

Contents

List of Tables

List of Figures

Abstract

Background

Nowadays an increasing amount of trajectory data with observed attribute values attached to the spatial positions of moving entities is generated. Data aggregation is an useful and proposed technique to explore and analyse such data. The R software environment provides solutions to handle and analyse spatial and spatio-temporal data. The recently implemented R package **trajectories** supports the representation and analysis of trajectory data. It contains methods for a *pure* spatial aggregation of trajectory point attribute data without respecting the trajectories' temporal characteristic.

Aims

This work aims to provide methods for meaningful overlay and (weighted) aggregation of trajectory data in R that respect the spatial and temporal domain. The methods are supposed to be applicable to objects of classes defined in **trajectories** and to aggregate trajectory point attribute data over spatial and spatio-temporal grouping predicates represented by objects of classes defined in the packages **sp** and **spacetime**. Another aim is to provide methods for coercion of objects of classes that represent trajectories and that are defined in other R packages.

Methods

The implementation of the newly-created methods is guided by the implementation of methods for spatial and spatio-temporal overlay and aggregation from the packages **sp** and **spacetime**. In particular this regards the methods' arguments, that are also extended, as well as the data structures of the returned objects.

Results

Methods for meaningful spatial and spatio-temporal overlay and aggregation of **trajectories** objects are implemented by S4 generic functions and provided by the R package **trajaggr**. The methods respect the spatial and temporal domain of the trajectory data and may be used with spatial or spatio-temporal grouping predicates. Weighted aggregation based on temporal or spatial characteristics of the trajectories is supported. Also methods for bidirectional coercion between objects representing trajectory data as well as for *counting* of trajectories over spatial and spatio-temporal regions are provided.

Conclusions

The implemented aggregation methods provide a first basic approach to fill the gap in the lack of suitable software solutions for (weighted) aggregation of trajectory data in R that respect the data's spatial and temporal domain in a meaningful way.

1. Introduction

Currently there is lack of suitable software solutions in R regarding meaningful (weighted) aggregation of trajectory data with respect to the trajectories' spatial and temporal characteristics.

Nowadays trajectory data with observed attribute values attached to the spatial positions of moving entities may be easily obtained, due to the further development of position tracking and parameter measuring sensors. Thus an increasing amount of such trajectory data is available. Due to the large amount of data techniques to explore, reduce and analyse these data are needed.

Data aggregation in general is considered as a meaningful process for grouping large data sets (Goldstein and Roth, 1994), in which spatial and temporal grouping predicates are common (Fredrikson et al., 1999). Meratnia and de By (2002) already proposed raster-based spatial and spatio-temporal aggregation for trajectory data. And today it is generally accepted that aggregation is a useful process for exploration, reduction and analysis of trajectory data (Andrienko et al., 2003; Giannotti et al., 2007; Andrienko and Andrienko, 2011).

The R software for statistical computing (R Development Core Team, 2014) as an open source and widely used software environment provides various packages that support handling and analysing of spatial and spatio-temporal data. In particular the recently implemented package **trajectories** (Pebesma and Klus, 2014) supports the representation and analysis of trajectory data with the ability to distinguish between (sets of) trajectories of particular entities. Regarding to aggregation the package **trajectories** provides methods performing a *pure* spatial aggregation of trajectory point attribute data without respecting the temporal characteristic of the trajectories.

This thesis aims for providing methods for meaningful overlay and aggregation of trajectory data in R that respect the spatial and temporal characteristics of the trajectories. The methods for overlay constitute the basis of the aggregation methods. All methods are supposed to respect the characteristic of the sampling rate as well as the overall duration (or distance) based on the intersection of the trajectories with the geometries of the grouping predicate, if several trajectories are involved. In case of aggregation of one trajectory characterized by an irregular sampling rate a weighted aggregation approach is supported that weights the observed values according to the sampling rate. In case of aggregation of several trajectories this approach may be extended by assigning weights to each particular trajectory, in which each weight corresponds to one particular geometry of the grouping predicate. These weights are based on the duration (or distance) of that trajectory corresponding to its temporal (or spatial) intersection with the geometries of the grouping predicate.

The implemented methods are supposed to be applicable to objects of classes defined in the package **trajectories** (`Track`, `Tracks`, `TracksCollection`) and to aggregate trajectory point attribute data over spatial and spatio-temporal grouping predicates represented by objects of classes defined in the packages **sp** (Pebesma and Bivand, 2005) and **spacetime** (Pebesma, 2012). In particular the accepted spatial grouping predicates are supposed to be objects of the classes `SpatialPolygons`, `SpatialPixels` and `SpatialGrid` as well as their `data.frame` counterparts from the package **sp**. For spatio-temporal grouping predicates objects of the classes `STF` and `STFDF` from the package **spacetime** are accepted.

Besides the implementation of methods for bidirectional coercion between objects of classes defined in the packages **move** (Kranstauber and Smolla, 2014) and **adehabitatLT** (Calenge, 2006) and objects of classes defined in the package **trajectories** are provided. The classes from **move** and **adehabitatLT** are defined to represent animal trajectory data. Due to the coercion methods the aggregation functionality is provided to trajectory data stored in objects of the classes defined in the mentioned packages. And vice versa data stored in **trajectories** objects may be analysed by methods provided by these packages. Moreover the coercion provides access to a large amount of data stored in the movebank database (Wikelski and Kays, 2011). This data is provided by objects whose classes are defined in the package **move**.

Finally, an aim is to provide methods for counting the number of trajectories over spatial and spatio-temporal grouping predicates. Such methods may be used to analyse the spatial or spatio-temporal distribution of trajectories and are especially suitable for trajectories without attributes or for trajectories whose attributes may not be aggregated in a meaningful way.

As pointed out the focus of this thesis is about aggregation over area-measured spatial or spatio-temporal grouping predicates. It aims not for providing aggregation methods using temporal or cyclic temporal grouping predicates. Also the aggregation of trajectory data by attributes is not part of this work. Moreover an aggregation in terms of aggregating a set of trajectory points to a smaller set of points, what may be known as generalisation, is neither part of this work. And finally no methods to aggregate trajectories connections, which are bounding in each case two consecutive trajectory points, are implemented.

Generally the implemented methods for overlay and aggregation are guided by the implementation of the methods for spatial and spatio-temporal overlay and aggregation defined in the packages **sp** and **spacetime**. Moreover the implemented methods are as far as possible consistent with the mentioned methods from **sp** and **spacetime** regarding to the methods' arguments and the data structure of the returned objects. The spatio-temporal overlay constitutes the basis of the aggregation methods.

Regarding to the implementation itself all methods are defined by S4 generic functions and the entirety of the implemented methods as well as some example data from movebank are integrated in the R package **trajaggr**. The package is developed with the current R version 3.1.2 (2014-10-31) (R Development Core Team, 2014). Even regarding to the coercion the methods are additionally defined by S4 generic functions, which enables an appropriate documentation as well as a comfortable usage.

This thesis is structured as follows: In Chapter 2 definitions for 'aggregation' and 'trajectory' are given and a review about the relevant literature and R packages is presented. In Chapter 3 theoretical issues and considerations related to trajectories and its meaningful aggregation are introduced and the theoretical basics about the chosen realization of the aggregation is presented. In Chapter 4 the example data sets, that are provided with the created R package **trajaggr** and that are used for illustration purposes are introduced. In Chapter 5 the implemented coercion methods are presented and illustrated. In Chapter 6 the implemented methods are introduced and illustrated by applying them to the example data sets. The method `over` may be used for spatial and spatio-temporal overlay with `Track` objects. The method `count` may be used to count the number of trajectories, and the method `aggregate` performs spatial and spatio-temporal aggregations of objects of the classes `Track`, `Tracks` and `TracksCollection`. In Chapter 7 the strengths and weaknesses of the implemented methods are discussed. In Chapter 8 the results are summarized and evaluated, and the Chapter 9 presents an outlook regarding to the weaknesses of the implementation and the remaining challenges related to meaningful aggregation of trajectory data.

2. State of the Art

2.1. Definitions

Aggregation

Data aggregation is a process deriving new data references, in which groups considered as wholes are created out of multiple original references (Andrienko and Andrienko, 2006). The authors point out that there are numerous different techniques for data aggregation.

Stasch et al. (2014) define spatio-temporal aggregation as a process of grouping observations relative to spatial regions and/or temporal intervals, based on a spatial and/or temporal grouping predicate, with a subsequent calculation of statistical measures on the observed values in these groups of observations by applying an aggregation function. The spatial regions and/or temporal intervals on which the groups are built represent the support of the aggregated values and indicate the change of support, which is typical for data aggregation.

Trajectories

The term 'Trajectory' is used in various different fields and disciplines and thus it is hard to give an overall general definition.

Roduit (2009) defines trajectories as a continuous functions from R to R^n, where n represents the number of spatial dimensions. In conjunction with movement data in the two dimensional space this function becomes $f(t) = [x, y]$, where t corresponds to time and x and y are the coordinates of the point locations.

Similarly but extended by attribute data trajectories are defined by Stasch et al. (2014) as 'mappings from discrete objects and times to spatial locations', in which attribute data may be attached to the spatial locations.

2.2. Literature review - Aggregation of trajectory data

One of the basic works about data aggregation comes from Fredrikson et al. (1999). They introduce spatial and temporal aggregation as well as aggregation by attributes.

Nowadays it is generally accepted that data aggregation is a meaningful process for grouping large data sets (Goldstein and Roth, 1994). Common aggregation approaches are using spatial and temporal grouping predicates (Fredrikson et al., 1999), and raster-based spatial and spatio-temporal aggregation are proposed for trajectories by Meratnia and de By (2002). For instance with GPS sensors a large amount of spatio-temporal data is generated and aggregation is a useful approach for exploration and reduction of such data respectively of trajectory data (Andrienko et al., 2003; Giannotti et al., 2007; Andrienko and Andrienko, 2011).

The authors N. Andrienko and G. Andrienko contributed a lot to the research about analysis of spatial and temporal data. They wrote a book about the exploratory analysis of such data (Andrienko and Andrienko, 2006). Furthermore and in particular they worked on the visualisation of spatio-temporal (movement) data (Andrienko et al., 2003; Andrienko and Andrienko, 2012) and also used aggregation techniques to support the visual exploration of such data (Andrienko and Andrienko, 2008, 2010, 2011). Moreover they worked on basic concepts of movement data and techniques to analyse it (Andrienko et al., 2008, 2011).

There is just a very little further literature that explicitly examines the meaningfulness of spatio-temporal aggregation of trajectory data. A current work is that from Stasch et al. (2014), that examines meaningful spatial prediction and aggregation in general but also gives some statements about meaningful aggregation of trajectory data. See Section 3.1 for further details.

As examples of studies whose analyses are based on the aggregation of trajectory data one my refer to the studies from D'Hondt et al. (2013) and from Elen et al. (2012). In the context of the latter study the authors used an appropriate measurement plan containing repeated measurements to guarantee a meaningful aggregation result. Moreover they stated that it is challenging to aggregate data in a meaningful way when the data is collected in an unstructured way.

2.3. R packages

The R software for statistical computing (R Development Core Team, 2014) as an open source and widely used software environment provides rich packages that support handling and aggregation of spatial or temporal data (**sp** (Pebesma and Bivand, 2005), **raster** (Hijmans, 2014), **zoo** (Zeileis and Grothendieck, 2005) and **xts** (Ryan and Ulrich, 2014)). Additionally the package **spacetime** (Pebesma, 2012) provides a first approach to fill the gap in the lack of suitable software solutions to analyse spatio-temporal data (Schabenberger and Gotway (2004) in Pebesma (2012)).

The Comprehensive R Archive Network provides several packages that deal with trajectory data (Pebesma, 2014). The packages **move** (Kranstauber and Smolla, 2014), **adehabitatLT** (Calenge, 2006) and **trip** (Sumner, 2013) focus on animal trajectories. The package **spacetime** provides amongst others the class `STTDF` which is inspired by **adehabitatLT** (Bivand et al., 2008) and designed for trajectory data in general.

The **move** package provides functions to access, analyse and download animal track data from the movebank database (Wikelski and Kays, 2011). Three S4 classes (`Move`, MoveBurst and MoveStack) are defined to store movement data. These classes extend the class SpatialPointsDataFrame.

The package **adehabitatLT** provides a S3 class named ltraj that is defined as a list of data.frame objects each containing the data of one trajectory. The spatial and temporal information is *simply* saved in columns of the data.frame objects.

The **trip** package provides functions to access and manipulate animal tracking data that is stored in a S4 class called trip. The class trip as well extends the class SpatialPointsDataFrame.

However only the package **spacetime** even supports spatio-temporal data handling and analysis including aggregation that respects the spatio-temporal characteristics of trajectory data.

The package **spacetime** provides various S4 classes and methods for spatio-temporal data including generic aggregation functions that support spatial, temporal and spatio-temporal aggregation (Pebesma, 2012). The above mentioned class STTDF, designed for trajectory data, has amongst others a slot named traj that contains a list of STI objects each representing one trajectory. The classes STI and its data.frame counterpart STIDF represent spatio-temporal irregular data, in which time and space points (of observed values) do not have an obvious organisation. Each value is stored in conjunction with its timestamp and its spatial feature (STIDF). Two further classes and its data.frame counterparts are defined in **spacetime**: STF and STS. The class STF represents a spatio-temporal full grid. For each spatial feature the same temporal sequence is stored and analogous data is sampled in the case of its data.frame counterpart STFDF. The class STS respectively the class STSDF represents a spatio-temporal sparse grid which has the same general layout as the class STF, yet not the full grid is stored but only observations with non-missing values.

Methods for spatial or spatio-temporal overlays as defined in the packages **sp** (Pebesma and Bivand, 2005) and **spacetime** (Pebesma, 2012) are the basis for aggregation methods for spatial or spatio-temporal data. In **sp** the overlay of spatial features and/or grids are defined to combine numerically two maps in which the over method retrieves indices or attributes from one spatial feature at the locations of another spatial feature (Bivand et al., 2008). In **spacetime** the method over is defined analogous to the definition in sp but relative to spatio-temporal data.

Recently the author of **spacetime** provided the package **trajectories** (Pebesma and Klus, 2014) that fits well into the special requirements of trajectory data. The package aims to fill the gap of missing generic data structures and methods for analysing trajectories with respect to space and time (Klus and Pebesma, 2014). The package provides a S4 class called Track that extends the class STIDF from **spacetime**. Additionally a slot named connections is part of the class definition to provide attribute data of the segments that connect two consecutive trajectory points. Two further S4 classes are implemented which represent a set of Track objects of one individual (Tracks) and a set of Tracks objects of different individuals (TracksCollection). The **trajectories** classes are enhanced versions of the class `STTDF` from **spacetime**. Relating to aggregation there is just a spatial aggregation method implemented in **trajectories** yet which aggregates the trajectory points' data.

Relative to the domains of space and time the classes representing trajectory data defined in the package **trajectories** are more *powerful* than the classes defined packages **move**, **trip** and **adehabitatLT** as a result of extending the class `STIDF` from **spacetime**.

Moreover **trajectories** classes are more *powerful* in comparison to the class `STTDF` defined in **spacetime** because it provides the `connections` slot and the three classes introduced above. Improved are the representation and distinction of points and connections, as well as the ability to combine and to distinguish different trajectories of one individual (`Tracks`) and distinguish sets of trajectories of different individuals (`TracksCollection`). As yet just one aggregation method has been implemented which aggregates the trajectory points over space.

3. Meaningfulness related to aggregation of trajectory data

This Chapter is divided into two sections. The first section explains which criteria are relevant related to a meaningful aggregation of trajectory data. The second section describes how some challenges derived from the criteria presented in the first Section are addressed.

3.1. Issues about meaningful aggregation of trajectories

The meaningfulness of aggregation of trajectory data depends on several criteria. The following list gives an overview and in the subsequent paragraphs these items are explained and discussed in detail. Important criteria are:

- the domains (space and/or time) represented by the grouping predicate,
- the target features whose data should be aggregated (points or segments)
- the measurement scales of the attributes to be aggregated,
- the choice of the aggregation function to be applied,
- the sampling rate of the trajectory data and its variability,
- the handling of irregular sampling rates (within one track)
- the approach to aggregate data (of several tracks) based on different sampling rates
- the units of the attribute data (if having a spatial or temporal reference),
- the variability of the attribute data,

Grouping predicates

Generally due to the spatio-temporal characteristic of trajectory data an aggregation process needs to respect the spatial as well as the temporal domain. In a meaningful aggregation one needs to keep the temporal and spatial information of the trajectory data. Due to the grouping predicate the temporal and/or spatial support of the (aggregated) data changes. If the grouping predicate is based on spatio-temporal geometries the spatio-temporal support of the aggregated values is equivalent to these spatio-temporal geometries.

If the grouping predicate represents for instance just the spatial domain, the spatial support of the aggregated values is again equivalent to the spatial geometries of the grouping predicate, but the temporal support of the data does not change, if one does not want to drop the temporal information completely. This unchanged temporal information should be kept, because in a meaningful aggregation it is (normally) desired to keep all the available information of the data to be aggregated. This results in a changed data structure that characterized by a spatial support equivalent to the spatial grouping predicate, whereas the temporal support keeps being unchanged and is equivalent to the track point timestamps. Thus the resulting data structure may be characterized by one time series possibly of length zero for each spatial geometry from the spatial grouping predicate, in which the timestamps of the time series correspond to the track point timestamps.

Moreover also other types of grouping predicates may be used. For instance temporal or cyclic temporal grouping predicates. Because these are not part of this work they are not further discussed in this context.

Target features

In aggregation of trajectory data one needs to distinguish between the aggregation of point data corresponding to the trajectory points and *connection* data corresponding to the segments connecting two consecutive track points. If one aims to aggregate track point data there are no further issues or restrictions to be respected aside from the criteria represented in this section.

In the case of aggregation of *connection* data there are some further issues which need to be considered. If the grouping predicate contains more than one spatio-temporal geometry covering the whole spatio-temporal *extent* of the trajectories whose data should be aggregated, some of the connections may intersect the spatial and/or the temporal borders of the spatio-temporal geometries. Thus an adequate approach is needed to assign the data values corresponding to these connections to the spatio-temporal geometries intersected by these connections.

Another special situation appears, if the spatial domain of the grouping predicate is represented by *complex* polygons and the aggregation of *connection* data is desired. In such cases those connections, which intersect the spatial borders of spatio-temporal geometries, may intersect some of these geometries twice or several times, which need to be respected when assigning the data values corresponding to these connections to the spatio-temporal geometries.

Measurement scales and data types

The ability to aggregate attribute data depends on the data types respectively of the measurement scales of the data. Stevens (1946) classified the measurement scales and permissible statistics, which may be applied in a meaningful manner to data corresponding to the defined scales. For instance the functions *count* and *mode* may be applied to all measurement scales. The *median* may be applied to the ordinal and higher scales, and the functions *mean* and *standard deviation* may be applied to the interval scale as well as to the ratio scale (Stevens, 1946). Stasch et al. (2014) interpret the question of meaningfulness in a more specific way and introduce an approach in which a statistical operation on data is classified as meaningful, if the operation is 'interpretable in the context in which the data was generated'.

Besides the scales introduced by Stevens (1946) there are further scales like for instance the cyclic scale corresponding to a direction of movement given in degree. But such a scale may be transformed to the nominal scale from Stevens. Thus Stevens measurement scales are not complete but well accepted.

Aggregation functions

As stated in conjunction with the definition of aggregation in Section 2.1 applying an aggregation function to a set of measurement values is the basis of an aggregation process. Especially the usage of the function *sum* in aggregation processes is problematic, which is known from the database community where it is referred to as the summarizability problem (Lenz and Shoshani, 1997; Mazón et al., 2009).

Stasch et al. (2014) pointed out that the function *sum* can just be applied as an aggregation function in a meaningful way if the 'complete knowledge about the extent over which values are summed' is given. That is not the case for trajectory data because we just have information about the parameter values at the track point locations where the data is measured, but not along the segments connecting the measurement points. Moreover if one aims to aggregate data of a trajectory there may be a lack of information about other entities which may move at the same time in the same spatial extent and which provide measurements of the same property as well.

For completeness it should be mentioned that situations might occur in which the function *sum* may be useful. For instance if one aims for counting of events or of a specific characteristic of a boolean variable a meaningful application of that function might be possible.

Other (*simple*) functions like the minimum or maximum of an attribute may be used as aggregation functions without any restrictions. The usage of functions calculating a summary value considering several measurement values like the function *mean* is possibly not purely meaningful in all situations depending on the number of involved tracks and the characteristics of the sampling rates of the involved tracks as it is described in the following paragraphs.

Sampling rate frequencies

Generally it is desirable that the temporal (or spatial) frequency of the sampling rate should be high enough such that the time intervals (or distances) between two consecutive measurement points are smaller than the time intervals (or spatial extents) characterizing the spatio-temporal geometries of the grouping predicate. That guarantees (at least for the temporal domain) that each spatio-temporal geometry intersected by the path of the track is (temporally) matched by at least one measurement point.

The aim of data aggregation is to calculate one representative value for an attribute out of a set of values of that attribute for each unit of the grouping predicate. An important point is the variability of the frequency of the sampling rate especially if an aggregation function like *mean*, that calculates a new value considering several measurement values, is used. A track with a sampling rate with constant time intervals between all pairs of two consecutive measurements may be called a *regular* trajectory and contrary a trajectory with a varying frequency of the sampling rate may be called an *irregular* trajectory, in which meaningful analyses of the latter are more challenging (Calenge et al., 2009).

The measured data of each track point typically correspond to a particular time instance of the temporal domain in conjunction with a particular point of the spatial domain, and thus these measurement points itself do not have any duration or distance. But due to the movement of the tracked object and the permanent progression of time one may calculate duration and distance for each segment connecting two consecutive track points. For a regular trajectory with a sampling rate based for instance on time the durations corresponding to the segments are all equal. So there is information about the attribute data related to the track points and information about duration and distance related to the connecting segments. To combine these information one need to assign information about duration and distance to the measurements in a meaningful way.

In case of a (temporal) regular trajectory the information about duration, if assigned to the measurement points, is equal for all points, and thus point data of such a regular trajectory may be aggregated without any restriction respectively without considering the duration assigned to the points. The same applies to the aggregation of data from several regular trajectories if their sampling rate frequencies are equal (and their speed of movement is approximately similar).

Assuming a (temporal) irregular trajectory in conjunction with the assignment of the varying duration from the sampling rate to the measurement points as described above one may need to respect this varying duration when calculating the representative aggregated value of an attribute. For instance the values, that got a long duration assigned, should have a higher impact to the representative aggregated value than corresponding values that got a short duration assigned. Thus the sampling rate may need to be respected when aggregating data of an irregular trajectory.

The same applies to to the aggregation of several trajectories, independent if they are all regular with different sampling rates or if they are irregular. In both cases it is useful to respect the duration and/or distance. For instance in case of aggregation of several regular trajectories with different sampling rates with no respect of the duration those attribute values from trajectories with higher sampling rates would be overrepresented in the resulting aggregated value. To clarify that one may assume two trajectories with different sampling rates following the same path and thus intersecting the same spatio-temporal geometries in the same way. But due to the higher sampling rate of one trajectory more track points of that trajectory would intersect the spatio-temporal geometries, and when calculating the aggregated values for the spatio-temporal geometries more points of the high frequent sampled trajectory would contribute to the resulting value. This would not be meaningful because the two trajectories should contribute equally to a aggregated attribute value instead of assigning a higher importance to the high frequent sampled trajectory.

Thinking further it may be desirable (in most cases) to have the ability to aggregate one or several trajectories with respecting the duration *and* the distance assigned to the track points. If such an aggregation would be useful depends on the data to be aggregated and on the questions to be answered by the aggregation process. An example of such a situation, in which it would be highly desirable to respect the duration as well as the distance, is the aggregation of several (temporal) irregular trajectories, at which the individuals (or entities) are moving with highly different speed, for instance a mixture of trajectories from pedestrians and cars, whereas the measured data is independent from the speed itself. Due to the temporal irregular sampling rate it would be desirable to respect the duration, and due to the highly different speed it would be desirable to respect the distance as well, because the fast individuals are passing the intersected spatio-temporal geometries faster and hence have on average less track points intersecting a particular spatio-temporal geometries than the slow individuals.

The decision if it is useful to respect merely the information about duration or distance in the case of aggregation of at least one irregular trajectory may depend on several factors like the underlying question to be answered and the data to be aggregated and may be decided independently in every particular case. But there are cases in which it is useful to respect merely duration or distance as described in the following paragraphs.

Spatial or temporal referenced data

In cases where the units of the data to be aggregated have a temporal (or spatial) reference the additional information relative to duration (or distance) should be doubtlessly respected. A meaningful aggregation approach needs to respect the duration if the data units have a temporal reference or analogous the distance needs to be respected if the data units have a spatial reference.

To illustrate that one may assume a trajectory of a motorized vehicle in conjunction with the measurement of the fuel consumption. The fuel consumption may be measured with reference to the driven distance for instance in litre per kilometre as well as with reference to the trip duration for instance in litre per hour. Generally, when aggregating data, the resulting aggregated values have the same units as the input values but with a changed spatial and temporal support. To meaningfully aggregate the consumption data given in litre per hour one need to respect the duration, that may be assigned to each measurement point like described above. The aggregated values with measurement units of litre per hour correspond to data units and to the durations of the track passing each spatio-temporal geometry. For instance a certain measurement point got assigned the half of the time duration of the overall duration corresponding to the measurement points intersecting a particular spatio-temporal geometry. Then the value corresponding to that certain point needs to contribute to 50 percent to the resulting aggregated value, independently of the number of further measurement points intersecting that particular spatio-temporal geometry. When aggregating attribute data with a spatial reference the distance assigned to the measurement points needs to be respected analogously.

Data variability

Moreover the variability of the data to be aggregated has an impact on the meaningfulness of aggregation in general. An aggregation result of a highly variable attribute need to be interpreted with caution as long as the frequency of the sampling rate is not high enough to catch the variability of the attribute.

3.2. Meaningful realization of aggregation

This section describes how the challenges mainly derived from the criteria related to a trajectories sampling rate presented in Section 3.1 are generally addressed by the approach of meaningful aggregation presented in this work. The aggregation approach is based on a core assumption related to the trajectories sampling rates which is also explained in this Section.

Assigning duration and distance to track points

The aggregation approach presented in this work uses the idea of assigning information about duration and distance of segments to track points, as it is mentioned in Section 3.1 in the paragraphs about the sampling rate frequencies. This allows the combination of information about duration and distance corresponding to segments with the track points and consequential also with the measured data, and thus it allows a weighted aggregation of track point attribute data based on duration or distance assigned to these track points.

Generally track points get the sum of the half of the duration and distance of their prior and subsequent segment assigned. But we have to take a closer look to track points whose prior and/or subsequent segment intersects the spatial or temporal border of spatio-temporal geometries of the grouping predicate, because such segments intersect (at least) two spatio-temporal geometries, and thus assigning information about duration and distance to that track points is not as simple as described above. In particular one may want to calculate the exact duration and distance of the segment parts that intersect the two spatio-temporal geometries, which are intersected by the two track points which define that segment, to obtain an adequate aggregation result in case of a applied weighted aggregation based on the assigned duration or distance.

A general assumption is made to avoid the calculation of the exact duration and distance of the parts of segments intersected by a spatial or temporal border of the spatio-temporal geometries. It is assumed that the sampling rate of trajectories, which may be based on time or distance, is relatively high related to the time intervals or spatial extents of the grouping predicate. For instance a track sampling rate of one measurement per hour and a spatio-temporal grouping predicate with time intervals of 10 minutes would not meet that assumption. As a general rule related to time based sampling rates it is assumed that the sampling rate time intervals should at least be smaller than the time intervals defined in the spatio-temporal grouping predicate.

This assumption limits the maximal possible *length* of the segments, in which length may be understood as spatial or temporal length. Moreover it guarantees that those two consecutive points, which define a segment that intersects a spatial or temporal border of a spatio-temporal geometry of the grouping predicate, are intersecting two *neighbouring* spatio-temporal geometries, in which *neighbouring* means that these two spatio-temporal geometries have at least one *corner point* in common.

Consequential the assumption allows an adequate estimation of the duration and distance for segment parts by simply assigning the half of the segments duration and distance, because the possible error is limited by the controlled segments' *length.* And basically the situation is as follows: The higher the frequency of the sampling rate the smaller is the possible error related to the estimation of duration and distance. Finally these estimated values about duration and distance of the segments parts may be used to calculate the values of duration and distance which will be assigned to the corresponding track points.

Usage of assigned spatio-temporal information

In the above paragraphs it is explained how information about duration and distance of segments is assigned in a meaningful manner to trajectory points to combine these spatio-temporal information with data corresponding to these trajectory points. This assigned spatio-temporal information is used in several ways.

By providing the information about duration and distance corresponding to trajectory points a weighted aggregation of one trajectory over a spatial or spatio-temporal feature may be applied, in which the track point data may be weighted on basis of the information about duration and distance. This weighted approach may also be used when applying a map overlay of a spatio-temporal features and a trajectory. In any case of aggregation (or map overlay using an aggregation function) the information about duration and distance is used to calculate metadata about the approximate total duration and distance of track points respectively track *parts* for each spatio-temporal geometry of the grouping predicate.

Furthermore the calculated metadata may be used to perform a weighted aggregation of several trajectories, in which the track *parts*, that intersect the same spatio-temporal geometry, are (additionally) weighted according to their duration or distance corresponding to that spatio-temporal geometry.

Other issues

Generally this work just contains aggregation approaches which aggregate point data. Regarding to the grouping predicate the presented approaches keep the temporal and spatial information of the trajectories in any case. For further details see Section 6.1. An aggregation using the aggregation function *sum*, which is typically not meaningful relative to trajectory aggregation, as it is described in Section 3.1, will be attended with a warning message.

4. Example Data

There are several example data sets provided with this package. Three data sets contain **move** objects from the movebank database (Wikelski and Kays, 2011), and one data set contains an object of class `ltraj` defined in the package **adehabitatLT**. Additionally some artificial toy data containing objects whose classes are defined in **trajectories** as well as other spatial and spatio-temporal objects are provided (but not documented).

Moreover a function is provided which creates `SpatialPolygons`, `SpatialPixels` or a `SpatialGrid` object covering the extent of a `SpatialPoints` object that is passed to that function as the main argument. This function and the toy data set are motivated and further introduced in the following paragraphs. The data sets from movebank and **adehabitatLT** are introduced in the subsequent sections.

Creation of `Spatial` grouping predicates

The implemented methods expect as main arguments at least a **trajectories** object and an object inheriting from class `Spatial` or of class `STF` as the required grouping predicate. To illustrate the functionality of the implemented methods they are called with **trajectories** objects (coerced) from the example data sets introduced in this chapter. But adequate objects acting as grouping predicates of class `Spatial` or `STF` need to be created additionally. To simplify that creation process a function is implemented that creates area-measured `Spatial` objects from `SpatialPoints`. These created objects cover the whole extent of the passed `SpatialPoints`. This function is called `createSpatialArealObjFromPoints`. The desired dimension of the *larger side* of the returned object as well as its desired class may be passed as optional arguments to the function.

Trajectories toy data

The creation of the toy data was inspired by the implementation of `example("Track")` from the **trajectories** package (Klus and Pebesma, 2014), but it is built up of even more simple time and attribute data. The data set contains several objects of classes defined in **trajectories** as well as other spatial and spatio-temporal objects which may be used as grouping predicates for the implemented methods. Objects from the toy dataset are used to illustrate the functionality of the method `over` in this vignette and to test the implemented methods related to the correctness of aggregated values using the package **testthat**. Moreover the data is used in some examples provided with the documentation. For a detailed inspection of the two objects from the toy data set that are used in the calls of the method `over` see Section 6.2. The toy data set may be loaded by:

```
> load(system.file("extdata","trajaggr_TestData.RData",
+                  package = "trajaggr"), verbose = TRUE)
```

4.1. Example data from movebank

Vultures data set

The vultures data comes from a study about search efficiency of vultures foraging on spatio-temporally unpredictable carcasses in the Etosha National Park in Namibia (Spiegel et al., 2013).

The `MoveStack` object `vulture_moveStack` represents a subset of the original data set downloaded from movebank and contains trajectory data of three vulture individuals, named X1 (*Gyps africanus*), X2 (*Torgos tracheliotus*) and X3 (*Gyps africanus*).

The vultures were tracked in Namibia, in which this data subset covers a 14 days period. The daily tracking of the vultures was limited to the time interval from around 6 o'clock in the morning to 6 o'clock in the evening. The sampling rate is not regular and the time intervals between the relocations mainly vary between around 2 and 5 minutes. The data set contains 9639 tracked point locations and eleven attributes with values corresponding to the point locations, as it is shown by the following commands. The data set is loaded by `data(vulture_moveStack)`:

```
> library(trajaggr)
> data(vulture_moveStack)
> class(vulture_moveStack)[1]

[1] "MoveStack"

> levels(vulture_moveStack@trackId)

[1] "X1" "X2" "X3"

> length(vulture_moveStack)

[1] 9639

> names(vulture_moveStack@data)

 [1] "ground_speed"           "heading"
 [3] "height_above_ellipsoid" "height_raw"
 [5] "location_lat"           "location_long"
 [7] "timestamp"              "visible"
 [9] "sensor_type_id"         "deployment_id"
[11] "event_id"
```

Due to the number of individuals and the characteristic of the data sampling this data sets needs some further preparation to be coerced to a `TracksCollection` in a suitable manner. This preparation is motivated and presented in Appendix A.1.

The permission to use the data was obtained from the owners of the data (Orr Spiegel). Further information about the data set may be obtained by `?vulture_moveStack`.

Pigeons data sets

The pigeons data come from a project, that studied the leadership-based flock structures of homing pigeons (*Columba livia*). In particular the repeatability of leadership-based flock structures was studied within a flight and across multiple flights conducted with the same animals (Santos et al., 2014).

The pigeons data consist of overall four `MoveStack` objects. The objects represent already restructured subsets of the original data downloaded from movebank. Each of the `MoveStack` objects `pigeon_R_moveStack` and `pigeon_S_moveStack` contains two trajectories of one pigeon individual. In each case the trajectories of the two pigeon individuals are sampled regularly and synchronously and are following almost the same course. Each of the four trajectories corresponding to the mentioned objects contains 480 point locations tracked over a period of two minutes and thus the regular sampling rates have a very high frequency characterized by four samples per second. Nine data attributes whose values are corresponding to the tracked point locations are available for each of the trajectories.

Additionally to these two introduced `MoveStack` objects equivalent but further limited objects are provided. Each trajectory represented by these objects is reduced to 60 point locations which corresponds to a sampling period of 15 seconds. These further reduced objects are named `pigeon_R_moveSt_sub` and `pigeon_S_moveSt_sub` and are used for illustration in Subsection 6.4.2 due to the reduced required computation time according to the reduced number of point locations.

For instance the data may be loaded by `data(pigeon_R_moveStack)`. The following commands give a short and exemplary inspecting of the `MoveStack` object `pigeon_R_moveStack`:

```
> # library(trajaggr)
> data(pigeon_R_moveStack)
> class(pigeon_R_moveStack)[1]

[1] "MoveStack"

> levels(pigeon_R_moveStack@trackId)

[1] "R_17923536" "R_17923512"

> length(pigeon_R_moveStack)

[1] 960

> names(pigeon_R_moveStack@data)

[1] "comments"               "ground_speed"
[3] "height_above_ellipsoid" "location_lat"
[5] "location_long"          "timestamp"
[7] "sensor_type_id"         "deployment_id"
[9] "event_id"
```

The permission to use the data was obtained from the owners of the data (Carlos David Santos). Further information about the data set may be obtained by the commands `?pigeon_R_moveStack` respectively `?pigeon_S_moveStack`. In the following Section 4.2 the example data from the package **adehabitatLT** is introduced.

4.2. Example data from the R package adehabitatLT

The package **trajaggr** contains one example data set of class `ltraj`, that is defined in the package **adehabitatLT**. The data set is stored in an object called `wildboars_4Ind_ltraj`.

This data set is a modified version of the trajectory data stored in the object `puechabonsp`, which is provided by the **adehabitatLT** (Calenge, 2006). The data represent the results of the monitoring of 4 wild boars in 1993 at Puechabon (Mediterranean habitat, South of France) and thus the original data contains four trajectories. The data set provided by the package **trajaggr** is modified in the way, that the trajectory of one individual is split (*bursted*) into two trajectories respectively bursts. This provides a more complex structure relative to the number of trajectories corresponding to the individuals. Thus the provided data contains overall five trajectories of overall four individuals.

The trajectories are characterized by a low frequent sampling rate with less than one tracked point location per day, as one may realize by inspecting the output of `summary` in the following block of commands. By `data(wildboars_4Ind_ltraj)` the data set may be loaded.

```
> # library(trajaggr)
> data(wildboars_4Ind_ltraj)
> class(wildboars_4Ind_ltraj)

[1] "ltraj" "list"

> # library(adehabitatLT)
> adehabitatLT::summary.ltraj(wildboars_4Ind_ltraj)

     id   burst nb.reloc NAs date.begin   date.end
1 Brock Brock.1       30   0 1993-07-01 1993-08-31
2 Calou Calou.1       19   0 1993-07-03 1993-08-31
3  Chou  Chou.1       16   0 1992-07-29 1992-08-28
4  Chou  Chou.2       24   0 1993-07-02 1993-08-30
5  Jean  Jean.1       30   0 1993-07-01 1993-08-31
```

The `wildboars_4Ind_ltraj` data is just used for illustration of the methods that are implemented for coercion of objects of class `ltraj` to objects defined in **trajectories**. The package **adehabitatLT** provides a lot of further example data sets.

In the following Chapter 5 the implemented coercion methods are introduced and illustrated by being applied to the example data presented in this chapter.

5. Coercion - Design and Implementation

Methods are implemented that provide (mostly) bidirectional coercion between objects defined in the packages **move** and **adehabitatLT** and objects of the classes `Track`, `Tracks` or `TracksCollection` from the package **trajectories**.

Generally coercion of an object `obj` may be applied by calling for instance `as(obj, "Track")`. For all coercion methods generic coercion methods like `as.Track(obj)` are implemented as well. These generic methods enable easy usage of the coercion methods as well as an easy and appropriate documentation due to the fact that coercion methods are usually documented in conjunction with the documentation of the classes which are involved in the coercion.

In the cases of coercion of `Tracks` or `TracksCollection` objects the `data` slots of the involved `Track` objects are checked for consistency. If there are attributes which exist just in some `data` slots the slots are harmonized by using the function `rbind.fill` from the package **plyr**. Thus for such cases the package **plyr** is required. This functionality is implemented because there is no guarantee for identical structured `data` slots in **trajectories** objects (yet). The coercion related to objects defined in the package **move** is introduced in the Section 5.1. The coercion related to `ltraj` objects defined in **adehabitatLT** is presented in Section 5.2.

5.1. Coercion related to objects defined in move

A general overview of possible coercion options between objects defined in the packages **move** and **trajectories** is given by Table 5.1. A coercion of `Track` objects to `MoveBurst` objects is obviously not possible because there is no information available on which the `Track` might be split into bursts of relocations.

By processing one of the coercion options shown in Table 5.1 the data from the `data` slot from e.g. a **move** object is passed to the `data` slot of the desired **trajectories** object and vice versa in the case of coercion in the contrary direction. Furthermore each **move** object has a slot named `idData`. In cases of coercion to `Tracks` or `TracksCollection` objects the data from `idData` is passed to the `tracksData` or `tracksCollectionData` slot and as well vice versa in the case of coercion in the contrary direction. Further details about the coercion related to **move** objects will be presented in the following subsections separated by the direction of coercion.

Class	Track	Tracks	TracksCollection
Move	bidirectional	-	-
MoveBurst	Track	bidirectional	-
MoveStack	-	bidirectional	bidirectional

Table 5.1.: Possible coercion of objects defined in the packages **move** and **trajectories**. If not 'bidirectional' the possible direction of coercion is indicated by the class of the resulting object.

Coercion of objects defined in move to trajectories objects

`Move` objects will be coerced to `Track` objects and `MoveStack` objects may be coerced to `Tracks` or `TracksCollection` objects depending on the number of individuals involved. It is the responsibility of the user to choose the right coercion, because there is no way to definitely decide if the trajectories stored in a `MoveStack` object belong to a single individual or to several individuals. `MoveBurst` objects may be coerced to `Track` or `Tracks` objects.

If a `Move` object is coerced to a `Track` object the information stored in the `idData` slot is lost. Moreover there are several other slots whose data is not passed to the desired **trajectories** object anyway. However this data may be classified as unimportant relating to trajectory data. For instance there is a slot `dateCreation` filled with an object of class `POSIXct` indicating the date of the data set creation.

A special case is the coercion of `MoveBurst` objects to `Tracks`. A `MoveBurst` object is similar to an object of class `Move` but is characterized by an additional slot named `burstId`. In this slot information is stored which is used to classify the segments connecting the consecutive points of an animal track. For instance the classification might be about the animals' behaviour, whereat each behavioural category could be specified and represented by an assigned particular character string. For each segment of the animal track one of these characterizing strings would be stored in the `burstId` slot. Thus the `burstId` slot contains a vector whose length equals the number of track point minus one and whose elements define the classification of the segments (similar to the `connections` slot of a `Track` object).

When coercing a `MoveBurst` to a `Tracks` object each set of consecutive segments belonging to the same classification category will be transformed to a particular `Track` object. To realize that those points with different `burstId` specifications corresponding to their prior and subsequent segments are duplicated, so that they may act as the last point of a `Track` representing a specific `burstId` category and at the same time as the first point of the next `Track` representing the following different `burstId` category. Thus the resulting `Tracks` object contains in total more track points than the `MoveBurst`. This difference in the number of track points equals the number of sets of consecutive segments with a common `burstId` category minus one. Due to the fact that the category of a set of consecutive segments do not have to be unique the resulting `Tracks` object may possess as well more `Track` objects than the number of classifying categories of the `MoveBurst`.

The coercion of **move** objects to objects defined in **trajectories** is illustrated by the following commands which perform coercions of an object of class `MoveStack` to `Tracks` and `TracksCollection` objects as well as of an object of class `MoveBurst` to `Tracks`. Further validation of these coercions by comparing the relevant data (`data` and `idData` slots, coordinates and timestamps) may be found in Appendix A.2.

```
> ### Coercion of MoveStack to Tracks
> # library(trajaggr)
> class(vulture_moveStack)[1]

[1] "MoveStack"

> v_Tracks <- as(vulture_moveStack, "Tracks")
> class(v_Tracks)[1]

[1] "Tracks"

> ## Compare as-method and generic method
> v_Tracks_gen <- as.Tracks(vulture_moveStack)
> identical(v_Tracks, v_Tracks_gen)

[1] TRUE

> ### Coercion of MoveStack to TracksCollection
> v_TrColl <- as.TracksCollection(vulture_moveStack)
> class(v_TrColl)[1]

[1] "TracksCollection"

> ### Coercion of MoveBurst to Tracks
> # Create a MoveBurst object from the first individuals' first day track
> # First subset the Move object to the first tracked day
> v_X1_Move <- vulture_moveStack[[1]]
> day1 <- which(as.Date(v_X1_Move@timestamps) ==
+                  as.Date(v_X1_Move@timestamps[1]))
> v_X1_1_Move <- v_X1_Move[day1]
> length(v_X1_1_Move) # number of locations

[1] 189

> # Create MoveBurst object with bursts specifying the type of
> # locomotion (on.ground or flying) based on the vulture's speed
> behav <- rep("on.ground", length(day1))
> behav[which(v_X1_Move@data$ground_speed[day1] > 5)] <- "flying"
> v_X1_1_mb <- move::burst(v_X1_1_Move, f = behav[1:length(behav) - 1])
> length(v_X1_1_mb@burstId) # number of bursts corresponding to segments
```

```
[1] 188

> # Coerce MoveBurst to Tracks ...
> v_X1_1_mbTracks <- as.Tracks(v_X1_1_mb)
> v_X1_1_mbTracks@tracksData[c("n", "tmin", "tmax", "medspeed")]

                  n                tmin                tmax
Burst1_on.ground 90 2008-05-02 06:03:59 2008-05-02 12:02:02
Burst2_flying    21 2008-05-02 12:02:02 2008-05-02 12:42:02
Burst3_on.ground 80 2008-05-02 12:42:02 2008-05-02 17:58:01
                    medspeed
Burst1_on.ground  0.03061205
Burst2_flying    15.95943986
Burst3_on.ground  0.01666610
```

Related to the coercion of `MoveBurst` to `Tracks` objects we see from the commands above that the *bursted* `Move` object contains 189 point locations. By its coercion a `Tracks` object is created whose overall sum of `Track` points is 191. This information may be obtained by the returned `tracksData` which contains the attribute `n` specifying the number of point locations for each `Track` object. Moreover we see that the end time, indicated by `tmax`, for instance of the first `Track` and the start time of the second `Track` are identical. The increased number of `Track` points and the identical timestamps indicate that two of the points of the original **move** object have been duplicated due to the implemented coercion method for coercing `MoveBurst` to `Tracks` as it is described above in detail.

We also see that the number of `Track` objects does not accord the number of `burst` categories. There are two `burst` categories, in particular ”on.ground” and ”flying”, and the resulting `Tracks` object contains three `Track` objects (see `tracksData` output). That is the case because the `burstId` ”on.ground” is used for two independent sets of consecutive segments which are transformed to two particular `Track` objects as described above. The Subsection 5.1 presents the contrary coercion of **trajectories** objects to **move** objects.

Coercion of objects defined in trajectories to objects defined in move

A general overview of the options to coerce objects of class `Track`, `Tracks` and `TracksCollection` to objects of class `Move`, `MoveBurst` and `MoveStack` is given by Table 5.1. The method `move` from package **move** is used to create `Move` objects, which in turn optionally may be passed to the methods `move::burst` or `move::moveStack` to create objects of class `MoveBurst` or `MoveStack`.

The `move` method has some specific characteristics which affect the coercion process. The method expects besides the coordinates of the point locations, the information about the coordinates reference, the temporal information and the `data` corresponding to the locations also optionally two character strings, one to specify the used `sensor` and another representing the individual ID or name (`animal`), which are specified as 'unknown' and 'unnamed' by default.

When creating a `Move` object using method `move` the passed character strings mentioned above are additionally stored in the `data` slot of the resulting object. Thus the `data` slot of a newly created `Move` object contains these two additional attributes not contained in the `data.frame` passed to the `data` argument of the `move` method. This also applies in all implemented coercion methods of **trajectories** objects to **move** objects, and thus the `data` slots of the `Track` object and of the `Move` object are never identical due to the contained attribute columns.

Moreover when creating a `Move` object by the method `move` attributes from the `data.frame` passed to the `data` argument, whose parameter values are characterized by an overall identical singular value, are automatically stored in the `idData` slot of the resulting `Move` object instead of being stored in the `data` slot. This characteristic is classified to be inadequate related to the coercion of **trajectories** objects to **move** objects, because on the one hand that data is related to the point locations and not to the individual even if the values are equal for all point locations, and on the other hand the `data` slot of the resulting object should contain (at least) all the attributes, which are contained by the `data` slot of the input object. To realize an adequate coercion this characteristic behaviour of the `move` method is adjusted by the implementation of the coercion methods to get a desired adequate resulting object, whose `data` slot contains in any case all the attributes of the `data` slot of the input object.

The coercion of `Tracks` to `MoveBurst` objects is a special case just as the contrary coercion described in the previous Subsection 5.1. Usually all point locations from the `Tracks` object are preserved and the points of the particular `Track` objects are concatenated to one (internal) `Move` object, which then is *bursted* on basis of the particular `Track` objects respectively of their names. To realize that additional segments are needed to connect the sets of points of the particular `Track` objects. These additional connections are not belonging clearly to one of the connected `Track` objects, and thus as well they are not belonging clearly to one of the `burst` categories, but however they need to get a `burst` category assigned. Regarding to that problem an additional `burst` category named 'undefined' is assigned to these additional connecting segments. Thus the resulting `MoveBurst` contains one more `burst` category than the input `Tracks` object contains `Track` objects. Furthermore there is still one limitation related to this approach. In the input `Tracks` objects all `Track` objects have to be stored in chronological order.

In special cases the `Tracks` object to be coerced may contain duplicated points (with identical timestamps) as end and start points of consecutive `Track` objects like it could be the case for instance if the object was created by coercion of a `MoveStack` object as it is described in the Subsection 5.1. Under these special conditions in each case of duplicated points one of these points will be deleted from the set of points which are be the basis for the creation of the `MoveStack` object. Thus consecutive `Track` objects have theoretically one common point which acts as the end point for one `Track` object and concurrently acts as the start point of the subsequent `Track` object. In this case the additional `burst` category 'undefined' is neither needed nor used.

The coercion of **trajectories** objects to **move** objects is illustrated by the following commands coercing `Tracks` to `MoveStack` and `MoveBurst` objects as well as coercing a `TracksCollection` to a `MoveStack`. As input the **trajectories** objects created in Subsection 5.1 are used. The coercion is further validated in Appendix A.3 by comparing the newly created **move** objects with the original **move** objects used as input for the coercion presented in Subsection 5.1.

```
> ### Coercion of Tracks to MoveStack
> # library(trajaggr)
> class(v_Tracks)[1]

[1] "Tracks"

> library(move)
> v_moveSt <- as.MoveStack(v_Tracks)
> class(v_moveSt)[1]

[1] "MoveStack"

> ### Coercion of TracksCollection to MoveStack
> class(v_TrColl)[1]

[1] "TracksCollection"

> v_moveSt <- as.MoveStack(v_TrColl)
> class(v_moveSt)[1]

[1] "MoveStack"

> ### Coerce Tracks to MoveBurst
> class(v_X1_1_mbTracks)[1]

[1] "Tracks"

> v_X1_1_newMB <- as.MoveBurst(v_X1_1_mbTracks)
> class(v_X1_1_newMB)[1]

[1] "MoveBurst"
```

The commands above in conjunction with the validation in Appendix A.3 verify the consistent coercion of `Tracks` and `TracksCollection` objects to objects defined in **move**. In the Section 5.2 the coercion related to `ltraj` objects defined in the package `adehabitatLT` is presented.

5.2. Coercion related to `ltraj` objects defined in adehabitatLT

Similar to the coercion of objects from **move** objects of class `ltraj` from **adehabitatLT** which will be coerced with respect to the number of IDs and bursts which are represented by the `ltraj` object. Due to the definitions of class `Track` in **trajectories** and of `burst` in **adehabitatLT** all coercion methods are built upon the determination that one `Track` corresponds to one `burst` and vice versa.

The main part of the `ltraj` data corresponds to steps connecting two consecutive relocations and hence this data is stored in the `connections` slot(s) of the created objects defined in **trajectories**. The only exception is the attribute `R2n` which corresponds to the relocations and hence is stored accordingly. Moreover additional information related to the relocation may be stored in an object of class `ltraj`. In particular this data is stored in a `data.frame` saved as an `attribute` named `infolocs` for each burst of relocations. In case of coercion this data is stored in the `data` slot(s) of the returned **trajectories** object. All this described specifications are valid as well in the case of a vice versa coercion.

Due to the fact that objects of class `ltraj` do not store any information about the coordinate reference system of the stored coordinates of the relocations, this information, if desired, needs to be set manually or gets lost dependent of the direction of the coercion.

Coercion of objects defined in adehabitatLT to trajectories objects

Generally just `ltraj` objects with temporal information (indicated by the `attribute typeII`) may be coerced to **trajectories** objects. If `typeII == FALSE` the `ltraj` object does not have any temporal information related to the timestamps of the relocations. Another limitation regards the attributes `id` and `burst` in cases of coercion to `Track` or `Tracks` objects. In the former case neither the `id` nor the `burst` can be stored in the `Track` object. And in the latter case the `id` of the individual can not be stored in the `Tracks` object, whereas the `burst` character strings are stored as names for the `Track` objects being part of the `Tracks` object. If a called coercion method is inadequate due to the desired class of the resulting object and the number of IDs and bursts of the passed object the evaluation of the method call is stopped. Examples of such cases are presented further down.

Objects of class `ltraj` tolerate NA values in the `data.frame` columns containing the spatial and temporal information. To realize a desired coercion the trajectory points with such missing information need to be dropped, because a `Track` object does not tolerate such missing values. In such cases the `ltraj` data corresponding to the steps between relocations needs to be recalculated, which requires an installation of the package **adehabitatLT**. In all other cases **adehabitatLT** is not required.

By the following commands the coercion of `ltraj` objects is illustrated by applying the coercion methods to the `ltraj` example data set `wildboars_4Ind_ltraj`. In Section

4.2 it is evaluated that the object `wildboars_4Ind_ltraj` is of class `ltraj` and contains altogether five busts of four individuals.

```
> # library(trajaggr)
> class(wildboars_4Ind_ltraj[1])

[1] "ltraj" "list"

> # Coerce ltraj track of first individual (first burst) to Track object
> wb_1_Track <- as(wildboars_4Ind_ltraj[1], "Track")
> class(wb_1_Track)[1]

[1] "Track"

> # Coercion of third and fourth burst which belong to the same individual
> wb_Tracks <- as(wildboars_4Ind_ltraj[3:4], "Tracks")
> class(wb_Tracks)[1]

[1] "Tracks"

> dim(wb_Tracks)

    tracks geometries
         2         40

> # Coercion of whole ltraj object (4 ind., 5 bursts) to TracksCollection
> wb_TracksColl <- as(wildboars_4Ind_ltraj, "TracksCollection")
> wb_TracksColl@tracksCollectionData

      n   xmin   xmax    ymin    ymax       tmin       tmax
Brock 1 698626 700387 3160768 3161559 1993-07-01 1993-08-31
Calou 1 699656 700419 3160553 3161678 1993-07-03 1993-08-31
Chou  2 699131 701410 3157848 3159572 1992-07-29 1993-08-30
Jean  1 699294 700306 3158012 3161450 1993-07-01 1993-08-31
```

Further validation of the applied coercion is presented in Appendix A.4. The last output shows that the whole object `wildboars_4Ind_ltraj` may be coerced to a `TracksCollection` object. That object itself is composed of four `Tracks` objects whereas the third `Tracks` object contains two `Track` objects corresponding to the third and fourth burst from `wildboars_4Ind_ltraj` belonging to the individual *Chou* (compare Section 4.2). By the following commands two examples of impossible coercion are shown.

```
> # Coercion of one burst to Tracks or TracksCollection is inadequate
> wb_1_Tracks <- as(wildboars_4Ind_ltraj[1], "Tracks")

Error in asMethod(object) : length(from) > 1 is not TRUE!
```

```
> # Coercion of bursts of several individuals to Tracks is inadequate
> wb_Tracks <- as(wildboars_4Ind_ltraj, "Tracks")

Error in asMethod(object) : length(unique(idVec)) < 2 is not TRUE!
```

In the following subsection the coercion of **trajectories** objects to objects of class `ltraj` is introduced and shortly illustrated by *re-coercing* the above created objects back to `ltraj` objects.

Coercion of trajectories objects to objects defined in adehabitatLT

Generally there are no further limitations in coercing **trajectories** objects to `ltraj` objects, but the package **adehabitatLT** is required in all cases. The following commands illustrate the coercion to `ltraj` objects by coercing the above created **trajectories** objects back to objects of class `ltraj`. Further validation of the coercion may be found in Appendix A.5.

```
> # Coercion of Track to ltraj
> # library(trajaggr)
> wb_1_ltraj <- as(wb_1_Track, "ltraj")
> class(wb_1_ltraj)

[1] "ltraj" "list"

> # Coercion of Tracks to ltraj
> wb_3and4_ltraj <- as(wb_Tracks, "ltraj")
> class(wb_3and4_ltraj)

[1] "ltraj" "list"

> # Coercion of TracksCollection to ltraj
> wb_ltraj <- as(wb_TracksColl, "ltraj")
> class(wb_ltraj)

[1] "ltraj" "list"
```

Aside from the mentioned limitations the output of the last command and the validation presented in Appendix A.5 show that the bidirectional coercion between `ltraj` objects and objects defined in **trajectories** preserves any relevant information. In the following Chapter 6 the implemented methods `over`, `count` and `aggregate` are introduced and illustrated.

6. Overlay and Aggregation - Design and Implementation

This Chapter is about (map) overlay, counting of trajectories and aggregation of trajectory data represented by classes defined in the R package **trajectories**. The Chapter starts with a description of general aspects related to the implemented methods `over`, `count` and `aggregate`. In the second Section the method `over` performing map overlays with `Track` objects is introduced in detail and applied to `Track` objects from the artificial toy data set introduced in Chapter 4. In the Section 6.3 the method `count` is presented and applied to **trajectories** objects created by coercion from the example `MoveStack` object `vulture_moveStack`. In the last Section of this Chapter the method `aggregate` is introduced and illustrated. For illustration the method is applied to the example data objects `vulture_moveStack`, `pigeon_R_moveSt_sub` and `pigeon_R_moveSt_sub`.

6.1. General aspects

This Section is about general aspects related to the implemented methods `over`, `count` and `aggregate`. Generally the arguments which may be passed to the implemented methods are explained in conjunction with the illustration of these methods in the corresponding sections. Besides the arguments already known from the methods `over` and `aggregate` implemented for objects of class `Spatial` and of classes defined in **spacetime** some additional arguments may be used when calling the methods `over` and `aggregate` for objects defined in **trajectories**. These additional arguments, that may be used amongst others for attribute data selection and to perform weighted aggregations, are introduced in this Section.

In conjunction with the arguments used to perform a weighted aggregation the general approach of the weighted aggregation is explained. This Section contains moreover the description of the metadata provided by the results of any aggregation performed either by the method `over` or by the method `aggregate` and a short introduction about the basic implementation approaches according to the class of the methods' grouping predicate. Assumptions, that are (further) made related to the **trajectories** objects passed to the implemented methods, are introduced in the following paragraph.

Assumptions

The implemented aggregation functionality is based on some core assumptions. We act on the assumption that the sampling rate of the **trajectories** objects is relatively high related to the time intervals or spatial extents of the grouping predicate. This assumption is introduced and motivated in detail in Section 3.2.

Moreover it is generally assumed that the time reference of all `Track` objects that are themselves part of a `Tracks` or `TracksCollection` object is based on the same time zone. This assumption is not (yet) a requirement related to the classes defined in **trajectories**.

Selection of attribute data

The argument `use.data` may be used in the methods `over` and `aggregate`. When calling the method `over` `use.data` indicates whether the returned object contains (aggregated) data from the data slot of the `Track` object or indices of the `Track` points matching the spatio-temporal geometries passed to the `x` argument of the `over` method. If the default `use.data = FALSE` is used indices are returned. If `use.data = TRUE` (aggregated) data are returned.

To subset the attributes to be returned `over` may be called with passing an index `vector` or a character `vector` with the names of the desired columns to the `use.data` argument. The former selects the attribute columns in the `data` slot by the given indices whereas the latter selects attribute columns by the given names.

For calls of the method `aggregate` the argument `use.data` needs to be *positive*. Its default is `use.data = TRUE`, but analogous to the options defined for the method `over` particular attributes from the `data` slot of the **trajectories** objects may be selected by passing an index or a character `vector` to the `use.data` argument.

Metadata related to the aggregation result

In conjunction with any aggregation either performed by the method `over` or by the method `aggregate` metadata for each geometry of the grouping predicate is provided. The calculation of these metadata is (partly) based on the chosen approach to assign segments' durations and distances to the trajectory points as described in Section 3.2. This metadata provides information about the approximate overall duration (`approx_duration`) and distance (`approx_distance`) assigned to the trajectory points respectively trajectory *parts* spatio-temporally intersecting the particular geometries of the grouping predicate. Moreover the number of `Track` points intersecting each geometry (`nlocs`) as well as the number of `Track` objects (`ntraj`), if more than one `Track` is involved, are provided. The metadata is stored as additional attributes in the `data.frame` respectively in the `data` slot of the object returned by called method.

Weighted aggregation

Besides the known arguments and functionality of `over` and `aggregate` methods from **sp** and **spacetime** the implemented methods for **trajectories** objects are extended with the possibility to pass a weighted aggregation function to the aggregation function argument `fn` respectively `FUN` to perform a weighted aggregation of **trajectories** object. Regarding the weighted aggregation function every built-in or user-defined function may be used, which expects a `vector` of weights as its second argument whereas the first argument specifies the data to be aggregated. For instance the function `weighted.mean` from the package **stats** as well as the function `weightedMedian` from the package **matrixStats** would be suitable.

The weighting of `Track` point attributes in the aggregation process may be performed using weights based on the duration or distance associated with the corresponding `Track` point. The approach to assign values about duration and distance to trajectory points is described in Section 3.2. The additionally provided argument `weight.points` is used to specify if the weights correspond to the assigned duration, specified by passing the `character` string 'byTime', or if the weights correspond to the assigned distance, specified by passing the `character` string 'byDist' to the `weight.points` argument.

Besides the weighted aggregation of `Track` objects, where each `Track` point got a weight assigned, there is also the possibility to perform a weighted aggregation of `Tracks` or `TracksCollection` objects, in which furthermore each `Track` object may need to get a weight assigned. These weights are derived from the metadata containing information about the overall duration and distance of a `Track` object corresponding to each spatio-temporal geometry of the grouping predicate.

Weights for the `Track` objects are specified by the additional argument `weight.tracks`, which expects just as the argument `weight.points` one of the character strings 'byTime' or 'byDist' to specify if the weights assigned to the `Track` objects are based on the overall duration or distance of that `Track` object corresponding to the spatio-temporal geometries of the grouping predicate. Typically both arguments should be specified equally.

Thus for a weighted aggregation of a `Tracks` or `TracksCollection` object the arguments `weight.points` and `weight.tracks` both need to be specified, whereas in a first step a weighted aggregation of each particular `Track` specified by the `weight.points` argument is performed including the calculation of metadata about the overall duration and distance for each `Track`. In the proximate step a weighted aggregation of the aggregation results from the first step is performed, in which the weights are specified by the argument `weight.tracks` and derived from the metadata calculated in the first step.

To allow moreover a weighted aggregation of `Tracks` or `TracksCollection` objects in which weighting is limited to either just passing weights related to `Track` points or just passing weights related to `Track` *parts* also the `character` string 'equal' may be passed to the arguments `weight.points` and `weight.tracks`. The string 'equal' indicates no weighting.

Applying aggregation functions individually

When applying the methods `count` and `aggregate` to objects of class `TracksCollection` an additional argument named `byID` may be specified. This argument indicates if the called method is applied to the whole of the trajectories contained by the `TracksCollection`, which is the default case (`byID = FALSE`), or if the method is applied individually to the `Tracks` objects of the `TracksCollection`, which is the case if the method is called with `byID = TRUE`. In the latter case the returned object contains for each (selected) attribute of the `TracksCollection` the individually aggregated attribute data for each

Tracks object of the TracksCollection respectively for each tracked individual or entity. Just as the attribute data also the metadata is calculated individually for each Tracks object. To distinguish the returned data related to the particular individuals the names of the trajectory attributes as well as of the metadata attributes are extended by prefixing the names of the Tracks objects.

Approaches according to the methods' grouping predicate

These paragraphs give a short introduction about the methods' implementation approaches according to the class of the methods' grouping predicate, whereat grouping predicate is meant to be the x argument of the over method or the by argument of the methods count and aggregate. The implemented methods accept two *types* of grouping predicates. On the one hand objects of class STF or its data.frame counterpart and on the other hand SpatialPolygons, SpatialPixels and SpatialGrid objects as well as their data.frame counterparts are accepted.

In the former case respectively if the grouping predicate is of class STF or its data.frame counterpart the returned object is of class STFDF, in which the spatio-temporal geometries correspond to the spatio-temporal geometries of the grouping predicate. That applies to all implemented methods. The resulting attribute values, stored in the data slot of the STFDF object, are calculated for each geometry of the grouping predicate on the basis of those trajectories points spatio-temporally intersecting a particular geometry.

In the latter case respectively if the grouping predicate is an object inheriting from class Spatial the situation is more complex. As explained in Section 3.1 for instance in the case of aggregation one would expect something like one time series, for instance an xts object, possibly of length zero, for each spatial geometry from the Spatial grouping predicate, in which the timestamps of the time series correspond to the trajectory points' time instances. But it is not possible to store a time series in a Spatial object in a comfortable and user-friendly way. However as stated as well in Section 3.1 one wants to keep the temporal information definitely, and to realize that a STFDF object is chosen as the object to be returned if the method aggregate is called with a Spatial object grouping predicate.

The sp slot of the returned STFDF object corresponds to the Spatial grouping predicate and the time slot is built out of the time instances of (the subset of) those trajectory points which are spatially intersecting the geometries of the grouping predicate. Consequential in case of aggregation of a Track object each Track point intersecting the Spatial grouping predicate corresponds to its particular spatio-temporal geometry of the returned STFDF object.

Thus this approach is specially suitable to aggregate synchronously sampled trajectories, related to the aggregation of Tracks or TracksCollection objects, because in the case of non-synchronously sampled trajectories for each trajectory point a particular spatio-temporal geometry based on the timestamp of that point is created. The same approach applies to the method over, in which the returned object is not of class STFDF but of one the typical classes returned by over depending on the method's arguments.

For the method `count` the implementation approach is kept simple and in case of an object of class `Spatial` passed to the `by` argument an object of class `Spatial*DataFrame` is returned in which the temporal information of the **trajectories** object passed to `count` is ignored.

6.2. `over` - Overlay with `Track` objects

The aim of the `over` method is to numerically combine two spatial or spatio-temporal features whereas indices or (aggregated) data of one feature corresponding to the (spatio-temporal) geometries of another feature are returned. An overview of the available `over` methods related to **trajectories** classes could be obtained by the command `showMethods` whereas the required packages need to be loaded first, if not done yet:

```
> library(trajectories)
> library(trajaggr)
> showMethods(over, classes = c("Track", "Tracks", "TracksCollection"))

Function: over (package sp)
x="SpatialGrid", y="Track"
x="SpatialPixels", y="Track"
x="SpatialPolygons", y="Track"
x="STF", y="Track"
x="TracksCollection", y="Spatial"
x="Track", y="Spatial"
x="Tracks", y="Spatial"
```

The output shows all available `over` methods related to the **trajectories** classes whereas the exported `over` methods from **trajaggr** are those whose signatures are characterized by `y="Track"`. Overlays for objects of class `STF` and for objects inheriting from class `Spatial` (`SpatialPolygons`, `SpatialGrid` and `SpatialPixels`) with objects of class `Track` are implemented.

Generally the implemented `over` methods for spatial and spatio-temporal overlay are geared to the `over` methods implemented in **sp** and **spacetime** related to the arguments and the returned data structures. The methods respect the spatial and temporal domain of the `Track` object which will cause a consideration of the time domain even if an overlay of a `Spatial` object with a `Track` object is performed.

Besides the arguments used by `over` implemented in **sp** and **spacetime** the `over` methods applying to `Track` objects are using the two further arguments `use.data` and `weight.points` which are introduced in Section 6.1.

In general over(x, y) returns an object whose length equals `length(x)` if `x` is of or inherits from class `STF` or whose length equals `length(x) * length(y)` if `x` is of one of the classes mentioned above which inherit from class `Spatial` and all points from `y` are intersecting `x`.

For illustration of the over methods the small artificial toy data set is used, which easily allows to understand and verify the results of the implemented methods. For more details about that data set see Chapter 4. One may load that data set by:

```
> load(system.file("extdata","trajaggr_TestData.RData",
+                  package = "trajaggr"), verbose = FALSE)
```

From the toy data set a Track object named Track_A1 and a STF object named stf_Polys_4t will (mainly) be used as input for the over methods. Before applying over to the toy data objects we will inspect the properties of these objects including their dimensions and their temporal data to get a first rough idea of the expected results:

```
> class(Track_A1)[1]

[1] "Track"

> dim(as(Track_A1, "STIDF"))

    space      time variables
        6         6         1

> library(spacetime)
> index(Track_A1@time) # Track point timestamps

[1] "2012-12-20 01:00:00 CET" "2012-12-20 01:04:00 CET"
[3] "2012-12-20 01:16:00 CET" "2012-12-20 01:20:00 CET"
[5] "2012-12-20 01:24:00 CET" "2012-12-20 01:36:00 CET"

> class(stf_Polys_4t)[1]

[1] "STF"

> dim(stf_Polys_4t)

space  time
    4     4

> class(stf_Polys_4t@sp)[1] # class of object in the sp slot

[1] "SpatialPolygons"

> index(stf_Polys_4t@time) # STF timestamps

[1] "2012-12-20 01:00:00 CET" "2012-12-20 01:21:00 CET"
[3] "2012-12-20 01:42:00 CET" "2012-12-20 02:03:00 CET"

> stf_Polys_4t@endTime # STF end times

[1] "2012-12-20 01:21:00 CET" "2012-12-20 01:42:00 CET"
[3] "2012-12-20 02:03:00 CET" "2012-12-20 02:24:00 CET"
```

We see that the `Track` object contains six points and the `STF` object is built out of four `SpatialPolygons`. From the timestamps we may derive the information that the time of the `STF` object is considered to reflect time intervals, because the `endTime` values differ from the `time` values, and that all `Track` point time instances match one of the first two time intervals. Moreover one may notice that the sampling rate of the `Track` object is irregular, because the time differences between the `Track` point timestamps vary.

Figure 6.1 shows a visual (pure) spatial 'overlay' of the `Track` points and the `SpatialPolygons` from the `sp` slot of the `STF` object which further helps to understand the results of the up-coming `over` calls and which shows that the `SpatialPolygons` cover the whole spatial extent of the `Track` object.

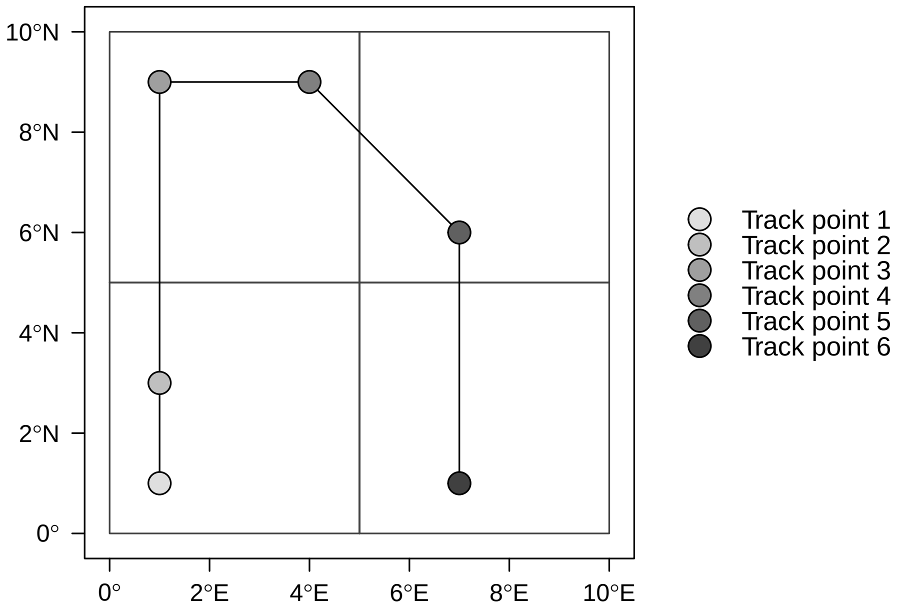

Figure 6.1.: Visual spatial 'overlay' of the example `Track` (`Track_A1`) and the four `SpatialPolygons` (squares) from the `sp` slot of the `STF` object (`stf_Polys_4t`) that are passed to the `over` example calls.

In the Subsection 6.2.1 the usage and the results of the `over` method for spatio-temporal overlay of `STF` and `Track` objects are presented using the introduced toy data objects as input.

6.2.1. Overlay of STF and Track objects

The spatio-temporal overlay of `STF` and `Track` objects performed by the method `over` returns `Track` point indices or (aggregated) `Track` attribute data for each spatio-temporal geometry of the `STF` object. Calling the method `over` with the argument `returnList = FALSE` returns a `vector` of length `length(x)` with `Track` point indices:

```
> over(x = stf_Polys_4t, y = Track_A1, returnList = FALSE)

 [1]  3 NA  1 NA NA  5 NA  6 NA NA NA NA NA NA NA NA
```

Each `vector` element corresponds to one spatio-temporal geometry of the `STF` object and represents the index of the (first) corresponding `Track` point due to spatio-temporal intersection to that geometry. Due to the convention that spatial features are cycled first in `STF` objects (Pebesma, 2012), the first four `vector` elements represent the (first) indices of points that spatio-temporally intersect the four polygons during the first time interval of the `STF` object. For instance the `Track` point number one (third `vector` element) intersects the third polygon during the first time interval, and the point number five (sixth `vector` element) intersects the second polygon during the second time interval. Moreover because there are no temporal matches between the `Track` point timestamps and the third and fourth time interval of the `STF` object, as we have seen above, the last eight `vector` elements just contain `NA` values.

In the next command we change `returnList` to `TRUE` and a `list` of length `length(x)` is returned whereas each `list` element corresponds to one spatio-temporal geometry and contains the indices of all `Track` points corresponding to that geometry. Note that the printed output of the returned `list` is limited to a subset of the first three elements:

```
> over_indexL <- over(x = stf_Polys_4t, y = Track_A1, returnList = TRUE)
> identical(length(stf_Polys_4t), length(over_indexL))

[1] TRUE

> over_indexL[1:3]

[[1]]
[1] 3 4

[[2]]
integer(0)

[[3]]
[1] 1 2
```

The output shows that for instance the first polygon during the first time interval is intersected by the third and fourth `Track` point, which verifies and extends the information from Figure 6.1 that the `Track` points three and four spatially intersect with the first, respectively the upper left, polygon.

The `use.data` argument, whose default is `FALSE`, selects between returning indices, as it was shown above, and returning (aggregated) data. In the following command `use.data` and `returnList` are set to `TRUE` which causes the return of a `list` of `data.frame` objects whereas each `list` element represents again one spatio-temporal geometry of the `STF` object. The printed output is limited to the first three `list` elements again:

```
> over_data <- over(x = stf_Polys_4t, y = Track_A1, returnList = TRUE,
+                   use.data = TRUE)
> over_data[1:3]

[[1]]
  co2
3  12
4   4

[[2]]
[1] co2
<0 rows> (or 0-length row.names)

[[3]]
  co2
1   8
2   4
```

The printed part of the returned list shows the data values of the `Track` points intersecting the first three polygons and at the same time matching the first time interval of the `STF` object. In the case of no matches a `data.frame` with zero rows is returned as it is for the second `list` element.

Relating to the `use.data` argument an integer or character `vector` may be passed to `use.data` as well to indicate that data should be considered and to select specific data columns at once, which is applied in the following two commands further down. If concurrently to any *positive* `use.data` argument `returnList` is set to `FALSE` a `data.frame` with `length(x)` rows will be returned. If the `fn` argument, whose default is `NULL`, keeps being unset each row of the returned `data.frame` contains the data of the first `Track` point that intersects the corresponding spatio-temporal geometry or `NA`, if there is no match. Moreover in the matching case the timestamp and the `timeIndex` of the (first) `Track` point is added to the `data.frame`. An example of such a call of the `over` method is presented in Subsection 6.2.2.

With the `fn` argument one may pass a built-in or user defined function to aggregate data by calculating summary statistics of data subsets. In this case each row of the returned `data.frame` contains the aggregated data values of one spatio-temporal geometry instead of the data of the first matching `Track` point as described above.

Additionally, if the `fn` argument is set, metadata about the number of relocations, the approximate duration and the approximate distance of the `Track` object are calculated for each spatio-temporal geometry. The estimated approximate values for duration and distance are calculated by summing the estimated values for each `Track` point intersecting the corresponding spatio-temporal geometry. The estimated point values are calculated by summing the half of the correspondent values of the `connections` which border on the corresponding `Track` point. See Section 3.2 for further details.

By adding `fn = mean` like in the following call of the `over` method mean attribute values are calculated. The returned `data.frame` contains the additional metadata represented by the variables `nlocs`, `approx_duration` [sec] and `approx_distance` [m]. The metadata are given in the same units as `duration` and `distance` in the `connections` slot of the `Track` object:

```
> over(x = stf_Polys_4t, y= Track_A1, returnList = FALSE,
+      fn = mean, use.data = c(1))

   co2 nlocs approx_duration approx_distance
1    8     2             720        895236.9
2   NA    NA              NA              NA
3    6     2             600        551981.3
4   NA    NA              NA              NA
5   NA    NA              NA              NA
6   12     1             480        510138.7
7   NA    NA              NA              NA
8    4     1             360        275980.0
9   NA    NA              NA              NA
10  NA    NA              NA              NA
11  NA    NA              NA              NA
12  NA    NA              NA              NA
13  NA    NA              NA              NA
14  NA    NA              NA              NA
15  NA    NA              NA              NA
16  NA    NA              NA              NA
```

Compared with the first and third `list` element returned by the presented command `over(x = stf_Polys_4t, y = Track_A1, returnList = TRUE, use.data = TRUE)` one may recognize the accurate calculated mean values of the attribute `co2` in the first and third row of the recently returned `data.frame`. Because there are no temporal matches between the `Track` point timestamps and the third and fourth time interval of the `STF` object, as we have seen above already, the last eight rows of the `data.frame` just contain `NA` values.

In the beginning of this Section we have seen that the sampling rate of the `Track` object `Track_A1` is irregular. In such a case it might be useful to expand the calculation of summary statistics with a weighting procedure as it was explained in detail in Chapter 3. The chosen approach is to weight the `Track` point attribute values according to the duration (or spatial distance) which may be assigned to each particular `Track` point. This weighting approach may be realized by passing a weighted aggregation function to the argument `fn` and concurrently passing an adequate `character` string to the `weight.points` argument, whose default is `NULL`. Currently the `weight.points` argument accepts the self-explanatory character strings 'byTime', 'byDist' and 'equal'.

The following call of the method `over` illustrates this weighting approach by calculating means of the attribute value `co2` weighted by duration ('byTime'). Because we found out already that there are no matches between the `Track` points and the third and fourth time interval of the `STF` object, a subset of the `STF` object covering just the first and second time interval is passed to the `x` argument of the `over` method:

```
> over(x = stf_Polys_4t[ , 1:2], y = Track_A1, returnList = FALSE,
+      fn = weighted.mean, use.data = "co2", weight.points = "byTime")

        co2 nlocs approx_duration approx_distance
1  9.333333     2             720        895236.9
2        NA    NA              NA              NA
3  4.800000     2             600        551981.3
4        NA    NA              NA              NA
5        NA    NA              NA              NA
6 12.000000     1             480        510138.7
7        NA    NA              NA              NA
8  4.000000     1             360        275980.0
```

Due to the subset of the `STF` object passed to `over` the returned `data.frame` just contains eight instead of 16 rows. We realize that the averaged `co2` values for the first and third spatio-temporal geometry of the `STF object`, respectively the first and third row of the returned `data.frame`, changed compared with the result of the prior unweighted call of the method `over`.

The time differences between the second and the third as well as between the fifth and the sixth `Track` point timestamps are three times larger then the other time differences. This causes a higher contribution (in this particular case a doubled weight) of the attribute values of the four mentioned points to the weighted means calculated for the spatio-temporal geometries. For instance the weighted mean for the first spatio-temporal geometry is calculated by

```
> (8 * 12 +   # contribution of the third Track point value (12), weight is 8
+  4 * 4) /   # contribution of the fourth Track point value (4), weight is 4
+  12         # sum of weights

[1] 9.333333
```

Correspondingly the weighted mean for the third spatio-temporal geometry, which is intersected by the first two `Track` points, is smaller than the unweighted mean, because the smaller `co2` value of the second `Track` point gets the higher weight. There are no effects on the values for the sixth and eighth spatio-temporal geometry due to the fact that in each case just one `Track` point intersects these geometries, and as well there are no effects on the returned metadata, which is independent from the weighting procedure.

6.2.2. Overlay of `Spatial` and `Track` objects

Due to the implementation approach of the method `over` for objects inheriting from class `Spatial` and `Track` objects, whose general aspects are explained in Section 6.1 in the paragraphs about the methods' grouping predicate, each element (or possibly row) of the returned data structure (`vector`, `list` or `data.frame`) contains the index or data of at most one `Track` point.

Thus for all cases of spatial intersection of the `Track` points with the `Spatial` object calls of the method `over` with `returnList = TRUE` return the *same* data (indices or attribute values) as calls with `returnList = FALSE`, whereas in the former case the data is returned as a `list` (possibly of `data.frame` objects) instead of a `vector` or `data.frame`.

Moreover the `length` of the returned object depends on the number of `Track` points intersecting the `Spatial` object. If all points intersect the `Spatial` object the length of the returned object is `length(x) * length(y)`.

These described characteristics will be illustrated by the following commands applying the `over` method with `use.data = FALSE`, which will cause the return of `Track` point indices:

```
> ### over, in which all Track points intersect the Spatial object
>
> # returnList = FALSE:
> over_sp <- over(x = stf_Polys_4t@sp, y = Track_A1, returnList = FALSE)
> over_sp

 [1] NA NA  1 NA NA NA  2 NA  3 NA NA NA  4 NA NA NA NA  5
[19] NA NA NA NA NA  6

> # Checking object length:
> identical(length(over_sp), length(stf_Polys_4t@sp) * length(Track_A1))

[1] TRUE

> # returnList = TRUE:
> over_sp_list <- over(x = stf_Polys_4t@sp, y = Track_A1, returnList = TRUE)
> # Checking if data is identical:
> bool_indices_1 <- !is.na(over_sp)
> bool_indices_2 <- sapply(over_sp_list, function(x) length(x) > 0)
> identical(over_sp[bool_indices_1], unlist(over_sp_list[bool_indices_2]))

[1] TRUE
```

The result of the above first `over` call with `returnList = FALSE` need to be interpreted as follows: The returned `vector` of indices has the length `length(x) * length(y)` and corresponds to a theoretical non-existent `STF` object which would be built up out of the `Spatial` object passed to the `x` argument and out of the time instances of the `Track` points spatially intersecting the `Spatial` object, whereas in this particular case all `Track` points, respectively their time instances, would be considered. This implies that the first four elements of the returned `vector` correspond to the four polygons at the time instance of the first `Track` point and that this `Track` point intersects the third polygon. Analogous we may realize that the second `Track` point as well intersects the third polygon, but at the time instance of the second `Track` point, and the third `Track` point intersects the second polygon at the time instance of the third `Track` point, and so on.

With the next commands it is shown analogous to the above commands what happens if just a subset of the `Track` points intersect the `Spatial` object. Moreover in the second call of `over` with `returnList = TRUE` an object of class `SpatialPixels` named `spPix`, which is conform to the `SpatialPolygons` object, is passed to the `x` argument to illustrate that these methods operate on objects of that class as well:

```
> ### over, whereas Track points partly intersect the Spatial object
>
> # returnList = FALSE:
> over_sp_partlyIntersec <- over(x = stf_Polys_4t@sp[1:2], y = Track_A1,
+                                returnList = FALSE)
> over_sp_partlyIntersec

[1]  3 NA  4 NA NA  5

> # Checking object length:
> intersecPoints <- over_sp_partlyIntersec[(!is.na(over_sp_partlyIntersec))]
> identical(length(over_sp_partlyIntersec),
+           length(stf_Polys_4t@sp[1:2]) * length(Track_A1[intersecPoints]))

[1] TRUE

> # returnList = TRUE, with SpatialPixels:
> class(spPix)[1]

[1] "SpatialPixels"

> over_spPix_partlyIntersec_list <- over(x = spPix[1:2], y = Track_A1,
+                                        returnList = TRUE)
> # Checking if data is identical:
> bool_indices_partly_1 <- !is.na(over_sp_partlyIntersec)
> bool_indices_partly_2 <- sapply(over_spPix_partlyIntersec_list,
+                           function(x) length(x) > 0)
> identical(over_sp_partlyIntersec[bool_indices_partly_1],
+           unlist(over_spPix_partlyIntersec_list[bool_indices_partly_2]))

[1] TRUE
```

We realize from the commands above that the length of the resulting object depends on the number of `Track` points intersecting with the `Spatial` object. The length is equivalent to `length(x)` multiplied with the number of intersecting `Track` points.

The resulting `vector` of the `over` call with `returnList = FALSE` indicates analogous to the explanations above that for instance the third `Track` point intersects the first polygon at the time instance of that third `Track` point (first `vector` element) and that the fifth `Track` point intersects the second polygon at the time instance of that fifth `Track` point (last `vector` element).

Besides we see that there are no differences in the results if either `SpatialPolygons` or `SpatialPixels` are passed to the `x` argument of the `over` method. The same applies if `over` is called with a `SpatialGrid` object passed to the `x` argument.

In the following commands the method `over` is called with a `SpatialGrid` object named `spGrid_ul` for argument `x` and `use.data = TRUE`. The `SpatialGrid` object contains just one cell and is *conform* to the upper left polygon from the Section above, and thus it does not cover the whole spatial extent of the `Track` object. These commands illustrate that for the `over` methods with a `Spatial` object given to the `x` argument passing an aggregation function to the argument `fn` has on the one hand no effect on the returned attribute data values (`co2`) but on the other hand affects the additional provided information in the returned `data.frame`:

```
> # Inspecting the SpatialGrid object
> class(spGrid_ul)[1]

[1] "SpatialGrid"

> spGrid_ul@grid

                   X1  X2
cellcentre.offset 2.5 7.5
cellsize          5.0 5.0
cells.dim         1.0 1.0

> # over with fn == NULL (default)
> over(x = spGrid_ul, y = Track_A1, returnList = FALSE, use.data = TRUE)

  co2                time timeIndex
1  12 2012-12-20 01:16:00         3
2   4 2012-12-20 01:20:00         4

> # over with fn != NULL
> over(x = spGrid_ul, y = Track_A1, returnList = FALSE, use.data = TRUE,
+      fn = mean)

  co2 nlocs approx_duration approx_distance
1  12     1             480        496141.6
2   4     1             240        399095.3
```

The R output of the above command related to the SpatialGrid object verifies that the Grid has a dimension of 1 x 1, respectively contains one cell. As Figure 6.1 illustrates the upper left polygon is spatially intersected by the third and fourth Track point, whose attribute data (co2) are part of the returned data.frame objects. As expected the number of rows of the returned data.frame objects is two, which equals length(x) multiplied by the number of intersecting Track points.

The first call of over without specifying the argument fn returns in addition to the Track point attribute data the timestamp and the timeIndex of the correspondent Track point. This applies in all calls of the method over with a *positive* use.data argument, returnList = FALSE, and an unspecified fn argument (fn = NULL). The variable timeIndex reflects the number of the Track points related to the consecutive character of the Track.

The second call of over with specifying the argument fn (in any case) returns in addition to the Track point attribute data the metadata about the number of Track points corresponding to each spatio-temporal geometry, as well as the approximate duration and distance assigned to the part of the Track (in this particular case just singular Track points) intersecting the spatio-temporal geometries. This also applies in all calls of the method over with a *positive* use.data argument, returnList = FALSE, and a specified fn argument.

If the Spatial object passed to the argument x is of length one, like in the last presented commands, a time series object, in particular an xts object, which reflects the Track points intersecting the Spatial geometry as time instances may easily be created by the following command block:

```
> over_sp_1cell <- over(x = spGrid_ul, y = Track_A1, returnList = FALSE,
+                        use.data = TRUE)
> # Creating the xts object, package xts is required
> # library(xts)
> xts_obj <- xts::xts(x = over_sp_1cell["co2"], order.by = over_sp_1cell$time,
+                      tzone = attr(Track_A1@time, "tzone"))
> class(xts_obj)[1]

[1] "xts"

> xts_obj

                    co2
2012-12-20 01:16:00  12
2012-12-20 01:20:00   4
```

6.2.3. Overlay of Spatial and Track objects ignoring time domain

If one is interested in the result of a *pure* spatial overlay ignoring the temporal information of the Track points, this may easily be obtained by applying the over method described in Section 6.2.1 with a STF object containing just one time interval which covers the whole duration of the Track of interest:

```
> # Creating STF object with one time interval
> stf <- STF(stf_Polys_4t@sp, time = Track_A1@time[1],
+            endTime = index(Track_A1@time[length(Track_A1@time)]) + 1)
> dim(stf)

space  time
    4     1

> # over returning list of indices
> over(x = stf, y = Track_A1, returnList = TRUE)

[[1]]
[1] 3 4

[[2]]
[1] 5

[[3]]
[1] 1 2

[[4]]
[1] 6

> # over returning (weighted aggregated) data
> over(x = stf, y = Track_A1, returnList = FALSE, fn = weighted.mean,
+      use.data = TRUE, weight.points = "byTime")

        co2 nlocs approx_duration approx_distance
1  9.333333     2             720        895236.9
2 12.000000     1             480        510138.7
3  4.800000     2             600        551981.3
4  4.000000     1             360        275980.0
```

The results from the `over` calls are consistent with former presented results: The returned `Track` point indices from the first `over` call verify the visual spatial 'overlay' presented in Figure 6.1. And the weighted mean attribute values calculated by the second `over` call are consistent with the weighted mean attribute values from the relevant rows from the `data.frame` returned by the correspondent call of `over` from Subsection 6.2.1.

If one aims to consider the temporal information in the resulting data structure of a *pure* spatial overlay a list of `xts` time series objects, whereas each list element corresponds to one spatial geometry from the `Spatial` object passed to the `x` argument, may be easily obtained by the following commands using the above created `STF` object named `stf`:

```
> # over returning a list of indices (again)
> indexL <- over(x = stf, y = Track_A1, returnList = TRUE, use.data = FALSE)
> # Creating a list of xts objects
> # library(xts)
```

```
> xts_list <- lapply(seq_along(indexL), function(z) {
+   xts::xts(x = Track_A1@data[indexL[[z]], , drop = FALSE],
+            order.by = index(Track_A1@time[indexL[[z]]]),
+            tzone = attr(Track_A1@time, "tzone")) })
> # Inspecting the returned object
> class(xts_list[[1]])
```

```
[1] "xts" "zoo"
```

```
> xts_list[[1]]
```

```
                    co2
2012-12-20 01:16:00  12
2012-12-20 01:20:00   4
```

The returned result, respectively the `xts` object of the first `list` element, is identical to the `xts` object created at the end of the Subsection 6.2.2. The above commands represent a further approach for a consistent spatial overlay of objects inheriting from class `Spatial` and `Track` objects without loosing any spatial or temporal information. The weak point of this approach is that the resulting data structure (`xts` objects) may hardly or at least just uncomfortable be stored in a `data` slot of a `Spatial` object.

Finally it should be mentioned that an internal `over` method is implemented which performs an overlay of an object inheriting from class `Spatial` with a `Track` object whereas the temporal domain is completely ignored. This method is internally used by the `over` methods presented in Subsection 6.2.2 and by the `count` methods as well if the count of **trajectories** objects over spatial geometries is desired (Subsection 6.3.1). In the following Section the `count` methods, which may be used to count the number of **trajectories** objects over spatial or spatio-temporal geometries, is introduced.

6.3. `count` - Counting of trajectories

The method `count` performs a counting of `Track` objects (even as part of `Tracks` or `TracksCollection` objects) over spatial or spatio-temporal geometries represented by `Spatial` or `STF` objects. The counting process is based on the presence of at least one `Track` point intersecting a particular geometry. The returned objects are `data.frame` counterparts of the input geometries inheriting from class `Spatial` or `STF`. An overview of the available `count` methods related to **trajectories** classes can be obtained by the command `showMethods`:

```
> showMethods(count, classes = c("Track", "Tracks", "TracksCollection"))
```

```
Function: count (package trajaggr)
x="TracksCollection", by="SpatialGrid"
x="TracksCollection", by="SpatialPixels"
x="TracksCollection", by="SpatialPolygons"
```

```
x="TracksCollection", by="STF"
x="Track", by="SpatialGrid"
x="Track", by="SpatialPixels"
x="Track", by="SpatialPolygons"
x="Tracks", by="SpatialGrid"
x="Tracks", by="SpatialPixels"
x="Tracks", by="SpatialPolygons"
x="Tracks", by="STF"
x="Track", by="STF"
```

As the output of `showMethods` indicates a call of `count` expects at least the two arguments `x`, which needs to be of class `Track`, `Tracks` or `TracksCollection`, and `by`, which needs to be an object inheriting from class `Spatial` or a `STF` object. The method `count` works also for the `data.frame` counterparts of the classes specified as possible `by` arguments in the output of `showMethods`. If `x` is a `TracksCollection` object one may specify the additional argument `byID`, which may be `TRUE` or `FALSE` (default), to indicate whether the number of intersecting `Track` objects should be counted separated by individuals (`Tracks`) or for the entirety of the `TracksCollection`.

The count result is stored as an attribute called `ntraj` in the `data` slot of the returned object and represents the number of `Track` objects intersecting the spatial or spatio-temporal geometries. In the cases of no intersection of `Track` points with a geometry the corresponding value of `ntraj` in the `data` slot of the returned object is set to `NA`. If a `TracksCollection` is passed to the argument `x` and `byID` is set to `TRUE` the calculated attributes are named by a concatenation of the character string 'ntraj' with the names of the corresponding `Tracks` objects from the `TracksCollection`.

The illustration of the `count` methods is based on the `vulture_moveStack` data set. For further information about that data set consult Chapter 4 or run `?vulture_moveStack`. Before the `vulture_moveStack` object may be coerced to a suitable structured `TracksCollection` object some preparation is needed which is motivated and illustrated in conjunction with the coercion in Appendix A.1. The results of the example calls of the method `count` are illustrated by plots.

The Subsection 6.3.1 presents the application of the method `count` with objects inheriting from class `Spatial` passed to the `by` argument. The Subsection 6.3.2 presents the counting over `STF` objects.

6.3.1. Counting trajectories over `Spatial` objects

In this subsection the method `count` with objects inheriting from class `Spatial` passed to the `by` argument is presented. To create such objects we use a simple function, that is introduced in Chapter 4. For the `x` argument objects of class `Tracks` and `TracksCollection` coerced from the `vulture_moveStack` data set, whose preparation and coercion is illustrated in Appendix A.1, are used.

The method `count` applied to `Spatial` objects as the `by` argument returns an object whose class is a `data.frame` counterpart of the class of the object passed to the `by` argument if that is not yet an object of class `Spatial*DataFrame` (See also Section 6.1.).

By the following commands a `SpatialGrid` object is created and the method `count` is called to count the number of `Track` objects from `Tracks_X1` over the `SpatialGrid`. The `Tracks` object `Tracks_X1` was created by the commands above and contains the 14 `Track` objects of the first vulture individual.

```
> spG_X1_dim25 <- createSpatialArealObjFromPoints(
+   as(Tracks_X1, "SpatialPointsDataFrame"),
+   desDim = 25, out = "SpatialGrid")
> class(spG_X1_dim25)[1]

[1] "SpatialGrid"

> count_Tracks_X1 <- count(Tracks_X1, spG_X1_dim25)
> class(count_Tracks_X1)[1]

[1] "SpatialGridDataFrame"

> str(count_Tracks_X1@data)

'data.frame':        325 obs. of  1 variable:
 $ ntraj: num  NA NA NA NA NA NA NA 2 3 6 ...
```

By the above call of `count` a `SpatialGridDataFrame` is returned. The `data` slot contains the attribute `ntraj` representing the result of the counting. The Figure 6.2 shows the `count` result and `Tracks_X1` as `SpatialLines` together in one graphic generated by `spplot`. One may identify two *hot-spot* regions where the vulture individual passed up to six of the 14 tracked days.

The following commands perform a count of tracks from the `TracksCollection` `vulture_TrC` over a `SpatialGrid`, which covers the whole spatial extent of the first two `Tracks` objects from that `TracksCollection`. The argument `byID` is set to `TRUE` which results in separated counts for each individual.

```
> spG_X1X2_dim15 <- createSpatialArealObjFromPoints(
+   as(vulture_TrC[1:2], "SpatialPointsDataFrame"),
+   desDim = 15, out = "SpatialGrid")
> count_vultureTrC_byID <- count(vulture_TrC, spG_X1X2_dim15, byID = TRUE)
> class(count_vultureTrC_byID)[1]

[1] "SpatialGridDataFrame"

> str(count_vultureTrC_byID@data)
```

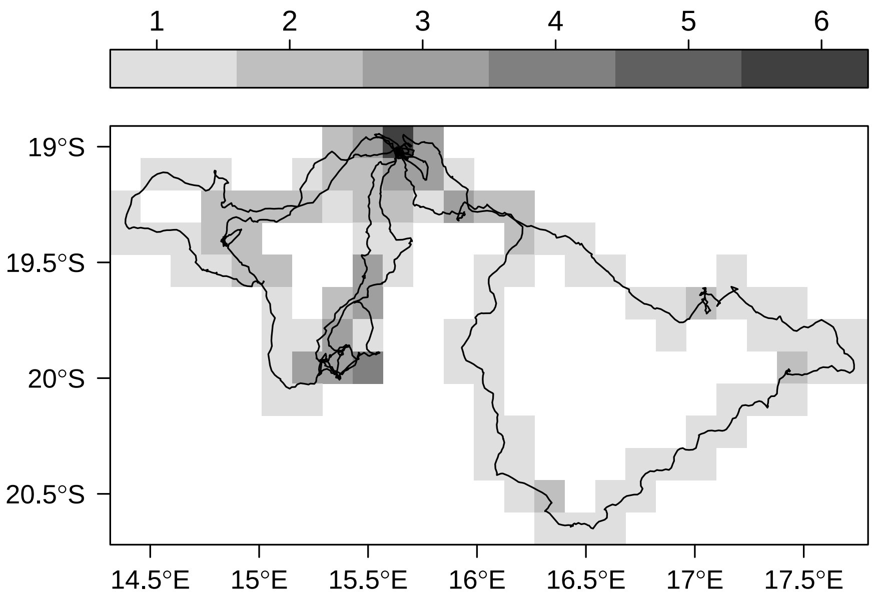

Figure 6.2.: Number of `Track` objects from the vulture individual X1 (*Gyps africanus*) counted for each grid cell. The tracks (`Tracks_X1`) are represented by `SpatialLines` and plotted above the `SpatialGrid`.

```
'data.frame':        135 obs. of  3 variables:
 $ ntraj_Tracks_X1: num  NA NA NA NA NA NA NA NA NA NA ...
 $ ntraj_Tracks_X2: num  NA NA NA 1 11 12 1 NA NA NA ...
 $ ntraj_Tracks_X3: num  NA NA NA NA NA NA 8 2 NA NA ...
```

The `data` slot of the resulting `SpatialGridDataFrame` from the recent `count` call contains three attributes, each representing the number of `Track` objects per grid cell of one individual. The names of the attributes reflect the corresponding individuals respectively the `Tracks` objects of the corresponding individuals. The results are graphically shown in Figure 6.3, whereas for a better design just the counts of the first two individuals are plotted.

In the same way as the method `count` was applied in this subsection it may also be applied with `by` arguments of class `STF`, which is presented in the Subsection 6.3.2.

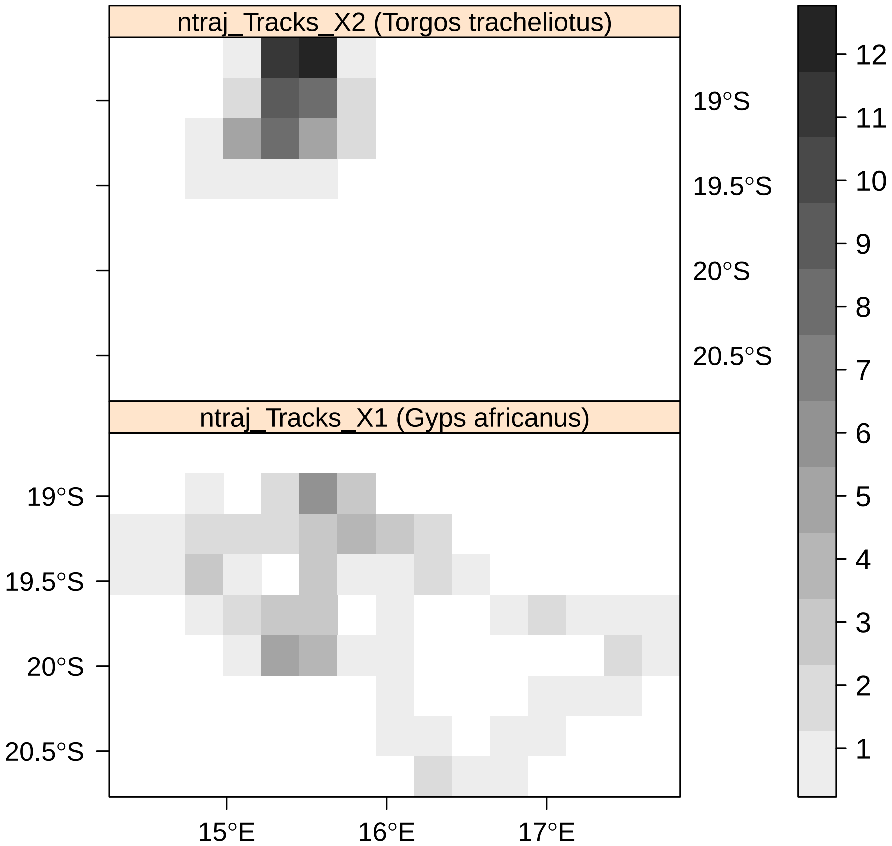

Figure 6.3.: Individual number of `Track` objects of two vulture individuals counted for each grid cell. Result from a call of `count` with a `TracksCollection` and `byID = TRUE`.

6.3.2. Counting trajectories over `STF` objects

For `by` arguments of class `STF` the method `count` works analogous to the above presented functionality whereas the returned object is of class `STFDF`. For each spatio-temporal geometry the number of intersecting `Track` objects is counted. For illustration we need a `STF` object which may be passed to the `by` argument. Such an object will be created by first creating an object of class `Spatial` as described above and then creating an object of class `STF` out of the `Spatial` object adding the desired temporal data to the slots `time` and (optional) `endTime`.

The functionality of counting over a `STF` object will be presented on basis of the `Tracks` object `Tracks_X3`. With the following commands first an object of class `SpatialPixels` is created which covers the extent of the `Tracks` object `Tracks_X3`. Subsequent a `STF` object is created out of the `SpatialPixels` containing four time intervals with an equal length of 3 days covering the first 12 of the 14 tracked days from `Tracks_X3`. With the last command the method count is called with the Tracks object Tracks_X3 and the recently created STF object.

```
> spPix_X3_dim15 <- createSpatialArealObjFromPoints(
+   as(Tracks_X3, "SpatialPointsDataFrame"),
+   desDim = 15, out = "SpatialPixels")
> stf_spPx_X3_4t <- STF(spPix_X3_dim15,
+                         time = Tracks_X3[c(1,4,7,10)]@tracksData$tmin,
+                         endTime = Tracks_X3[c(3,6,9,12)]@tracksData$tmax)
> count_Tracks_X3_stf <- count(Tracks_X3, stf_spPx_X3_4t)
```

The result of the above call of count is presented in Figure 6.4. The Figure shows the number of Track entities from the Tracks object Tracks_X3 (*Gyps africanus*) counted for each spatio-temporal geometry of the STF object built up out of SpatialPixels and four time intervals. Each of the four panels represents a three days time interval whose starting dates are given by the panel titles. One may identify some kind of a route with a north-south orientation which is passed by the vulture individual in three of the four time intervals on two tracks (Track objects). Due to the fact that one Track covers the vulture movement of one day we may interpret that the vulture passed that route at six of the twelve days of the total STF object time interval at least one time per day. At the both *endpoints* of that north-south orientated route one may identify areas which were visited on two or three days of nearly each three day interval. Moreover one may identify some kind of day trips to the surrounding environment of that north-south orientated route especially during the first three time intervals.

We see that with applying the method count one may retrieve basic information from trajectory data about the presence of one or more individuals related to spatial or spatio-temporal geometries. In the next Section (Section 6.4) the functionality of the method aggregate is presented which may be used to retrieve summary statistics of attribute data measured for Track point entities as well as to get further trajectory (meta-)data about the number of relocations and the approximate duration and distance corresponding to each spatio-temporal geometry.

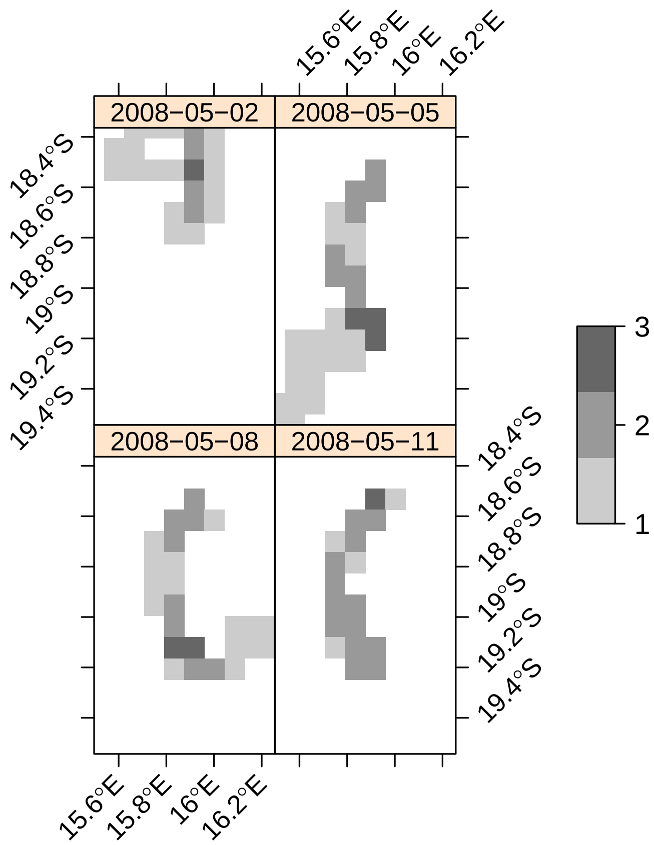

Figure 6.4.: Number of `Track` objects of the vulture individual X3 (*Gyps africanus*) counted over spatio-temporal geometries. Each panel represents a three days time interval whose starting dates are given by the panel titles.

6.4. `aggregate` - Aggregation of trajectories objects

The aim of the implemented `aggregate` methods related to **trajectories** objects is to aggregate attribute values observed in conjunction with trajectory points' records on basis of a given grouping predicate, that is represented by spatial or spatio-temporal geometries. An overview of the available S4 style `aggregate` methods for classes defined in **trajectories** could be obtained by the command `showMethods` again:

```
> showMethods(aggregate, classes = c("Track", "Tracks", "TracksCollection"))

Function: aggregate (package stats)
x="Track"
x="Tracks"
x="TracksCollection"
```

Due to the facts that the *original* `aggregate` method is a S3 style method (package **stats**) and that the implementation of `aggregate` in the package **trajaggr** uses internal methods to dispatch on the `by` argument, that represents the grouping predicate, the output of `showMethods` just shows the `x` argument of the `aggregate` methods.

The `aggregate` methods accept analogous to the methods `over` and `count` an object inheriting from class `Spatial`, in particular a `SpatialPolygons`, `SpatialPixels` or `SpatialGrid` object, or a `STF` object as its grouping predicate. The methods respect the spatial and temporal domain of the passed `trajectories` object which causes a consideration of the temporal domain even if a `Spatial` object is passed to the `by` argument. The returned object is always of class `STFDF`.

Generally the implemented `aggregate` methods are geared to the `aggregate` methods implemented in **sp** and **spacetime** related to the arguments, that may be specified. Arguments known from the mentioned methods are `x` for the object whose data should be aggregated, `by` for the grouping predicate, `FUN` for the aggregation function and `simplify`. If `simplify` is specified as `TRUE` (default) it affects the return of the `aggregate` methods if the dimension of the temporal or spatial domain of the grouping predicate equals one. In such cases the returned object is *simplified* by dropping the domain whose dimension equals one and in case of aggregation of **trajectories** objects the returned *simplified* objects may be of class `Spatial`, if the temporal domain is dropped, or of class `xts`, if the spatial domain is dropped.

Besides these arguments mentioned above the `aggregate` methods for `trajectories` objects are using further arguments depending of the class of the object passed to the `x` argument. In particular these further arguments are `use.data`, `weight.points`, `weight.tracks` and `byID` which are introduced in detail in Section 6.1.

In cases where `x` is of class `Tracks` or `TracksCollection` the `data` slots of the `Track` objects are checked for consistency related to the existence of attributes by comparing the `data.frame` column `names`. This is performed because it is not a requirement related to the classes defined in **trajectories**, that all `Track` objects' `data` slots are *identical*. In the case of disparity of the `data` slots missing columns are added to the relevant `data.frame` objects and filled with `NA` values. But note that if `aggregate` is called with a `TracksCollection` and `byID = TRUE` this consistency check of the `data.frame` objects is just performed individually for each `Tracks` object, and it is not ensured that the different `Tracks` objects have *identical* `data` slots.

Spatio-temporal aggregation of **trajectories** objects over `STF` objects is illustrated in Subsection 6.4.1 by applying the `aggregation` method to the **trajectories** objects coerced from the `vulture_moveStack` example data. The Subsection 6.4.2 the aggregation with a `Spatial` grouping predicate applied to **trajectories** objects coerced from the pigeons' example data is presented.

6.4.1. Spatio-temporal aggregation of trajectories objects

The spatio-temporal aggregation of **trajectories** objects over an object of class `STF` performed by the method `aggregate` returns an objects of class `STFDF` with the same spatial and temporal component as the passed grouping predicate. The `STFDF` object's `data` slot contains the aggregated attribute values corresponding to each particular spatio-temporal geometry of the grouping predicate.

As mentioned in the introduction of this chapter the **trajectories** objects coerced from the `vulture_moveStack` example data are used to illustrate the aggregation. Before the `vulture_moveStack` object may be coerced to suitable structured `Tracks` and `TracksCollection` objects some preparation is needed which is motivated and illustrated in conjunction with the coercion in Appendix A.1.

As an first example we will calculate the minimal altitude of a vulture individual (*Gyps africanus*) over spatio-temporal regions. Due to the aggregation function calculating the minimum in conjunction with the selected attribute representing the altitude there is neither the need nor the option of using a weighted aggregation approach to obtain a meaningful aggregation result.

By the following commands first a `STF` is created. That object has a `sp` slot with `SpatialPixels` covering the spatial extent of the `Tracks` object `Tracks_X1`, that represents 14 day-tracks of a vulture individual. Related to the time domain the `STF` object has four time intervals each covering three days of the 14 days the vulture was tracked. Subsequently the attribute `height_raw` of the `Tracks_X1` object, selected by the argument `use.data`, is (unweighted) aggregated over the spatio-temporal geometries of the created `STF` object. As aggregation function the function `min` is passed to the argument `FUN`, which determines the minimal altitude of the vulture individual for each spatio-temporal geometry.

```
> # Create SpatialPixels covering the extent of the Tracks_X1 object
> spPix_X1_dim15 <- createSpatialArealObjFromPoints(
+   as(Tracks_X1, "SpatialPointsDataFrame"),
+   desDim = 15, out = "SpatialPixels")
> # Create a STF object with four time intervals
> stf_px_X1_4t <- STF(spPix_X1_dim15,
+                      time = Tracks_X1[c(1,4,7,10)]@tracksData$tmin,
+                      endTime = Tracks_X1[c(3,6,9,12)]@tracksData$tmax)
> # Aggregate the attribute 'height_raw' using function 'min'
> agg_X1_stf_minHght <- aggregate(x = Tracks_X1, by = stf_px_X1_4t, FUN = min,
+                                 na.rm = TRUE, use.data = "height_raw")
> # Class of returned object
> class(agg_X1_stf_minHght)[1]

[1] "STFDF"

> # Some example rows from the data slot of the resulting STFDF object
> agg_X1_stf_minHght@data[474:477, ]
```

```
    height_raw nlocs approx_duration approx_distance ntraj
474        8.6    68         17343.0        13314.09     1
475       22.5    43          7440.0        37409.59     1
476      707.5     2           240.5         3480.54     1
477         NA    NA              NA              NA    NA
```

The output shows that the object returned from the method `aggregate` is of class `STFDF` and that the `data` slot of the returned `STFDF` object contains the aggregated values of the selected attribute `height_raw` as well as the calculated metadata for each spatio-temporal geometry. The metadata is being composed of the variables `nlocs`, representing the number of trajectory points, `approx_duration`, representing the overall sum of duration assigned to the trajectory points, `approx_distance`, representing the overall sum of distance assigned to the trajectory points, and `ntraj`, representing the number of trajectories intersecting the particular geometries.

The Figure 6.5 gives a visual representation of the aggregated values of the attribute `height_raw` indicating the minimal altitude of the vulture for each spatio-temporal geometry. The graphic is divided into four panels each representing a three days time interval, whose starting date is indicated by the panel title. From the figure one may for instance derive the information in which spatio-temporal regions the vulture individual stayed on the one hand at least temporarily on or near to the ground and on the other hand which regions were overflown with an enormous altitude.

In the following example the individual average speed of three vulture individuals, whose trajectories are stored in an object of class `TracksCollection`, over spatio-temporal regions is calculated. Due to the temporal reference of the attribute speed in conjunction with the aim to calculate averaged values a aggregation weighted by time is suitable. In particular this includes a weighted aggregation of the measurements corresponding to the trajectory points for each `Track` due to a slightly irregular sampling rate as well as a weighted aggregation of the aggregated values corresponding to each `Track` due to the temporal reference of the attribute speed.

The described approach of the aggregation of the vultures' speed is realized by the commands following further down. For the grouping predicate the `STF` object from the last `aggregate` call is used, which in fact does not cover the whole spatial extent of the `TracksCollection`, but for visual presentation of the aggregation results just the aggregated values of the first individual (`Tracks_X1`) will be plotted.

To realize the weighted aggregation of the `TracksCollection` object `vulture_TrC`, which is created by coercion from the `vulture_moveStack` (Appendix A.1), a weighted aggregation function needs to be passed to the argument `FUN` and further arguments need to be specified. The function `weighted.mean` is passed to the `FUN` argument, and the `character` string 'byTime' is passed to the arguments `weight.points` and `weight.tracks` to indicate that the used weights should be based on the duration assigned to the `Track` points and `Track` *parts* intersecting the particular spatio-temporal geometries as described in Section 3.2. Moreover the argument `byID` is set to `TRUE`

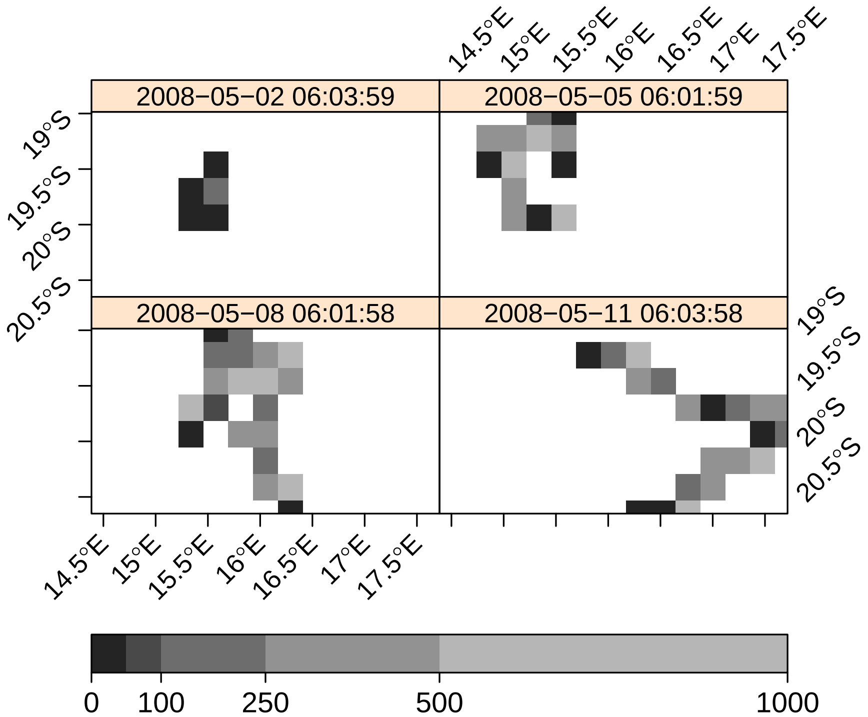

Figure 6.5.: Minimal altitude of the vulture individual X1 (*Gyps africanus*) over spatial regions and four consecutive 3-days time intervals. Result from the method `aggregate` applied to a `Tracks` object and a spatio-temporal grouping predicate. Each panel represents one of the 3-days time intervals whose starting dates are given by the panel titles. The altitude is given in metres.

to indicate that the aggregated values are calculated individually for each vulture respectively for each `Tracks` object. By the way an aggregation of a `TracksCollection` with calculating aggregated values over all `Tracks` objects (`byID = FALSE`) is performed in Subsection 6.4.2.

The ability of creating missing attribute columns in particular `Track` objects in cases where the `data` slots of the `Track` objects for instance of a `Tracks` object are inconsistent related to the existing columns is described in the introduction of this section. To illustrate that ability the attribute `ground_speed` is delete from 13 of the 14 `Track` objects from the second vulture individual respectively from the second `Tracks` object of the `TracksCollection`, as one may realize from the following commands.

```
> # Identify the index of the attribute ground_speed
```

```
> w_spd <- which(names(vulture_TrC[2][2]@data) == "ground_speed")
> # To illustrate the creation of missing attributes the attribute
> # ground_speed is deleted from the data of the second vulture
> for (i in 2:length(vulture_TrC@tracksCollection[[2]]@tracks)) {
+   vulture_TrC@tracksCollection[[2]]@tracks[[i]]@data <-
+     vulture_TrC@tracksCollection[[2]]@tracks[[i]]@data[ , -w_spd]
+ }
> # Attribute still available in e.g. second Track...?
> "ground_speed" %in% names(vulture_TrC[2][2]@data)

[1] FALSE

> # Weighted aggregation of the TracksCollection separated by individuals
> agg_vTrC_wMeanSpd <- aggregate(x = vulture_TrC, by = stf_px_X1_4t,
+                                FUN = weighted.mean, na.rm = TRUE,
+                                use.data = "ground_speed",
+                                weight.points = "byTime",
+                                weight.tracks = "byTime", byID = TRUE)
> # Overview of individual ground_speed
> summary(agg_vTrC_wMeanSpd@data[["Tracks_X1.ground_speed"]])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  1.108   6.703  15.200  12.560  16.370  22.430     422

> summary(agg_vTrC_wMeanSpd@data[["Tracks_X2.ground_speed"]])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  1.700   2.519   3.339   3.339   4.158   4.977     478

> summary(agg_vTrC_wMeanSpd@data[["Tracks_X3.ground_speed"]])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
 0.5988  2.8910  8.4630  8.4710 13.4300 15.4700     467
```

The output of the above commands shows that it is no problem related to the aggregation process if some attributes are missing in some `Track` objects of a `Tracks`. The aggregated values of `ground_speed` of the second `Tracks` object (`Tracks_X2.ground_speed`) returned by the `summary` command are relatively small compared to the other `Tracks` objects' aggregated values. This is explained by the deletion of the speed attribute values of 13 of the 14 `Track` objects of the second `Tracks` object, even if that is not directly proved by the above commands.

The Figure 6.6 visually presents the aggregated values of `ground_speed` indicating the weighted averaged speed of the vulture individual for each spatio-temporal geometry. The graphic is again divided into four panels each representing a three days time interval, whose starting date is indicated by the panel title. From the figure one may derive the information in which spatio-temporal regions the vulture individual was moving fast which may indicate (continuous) flying over long periods. For spatio-temporal regions where the averaged speed is lower one may derive that the vulture at least stayed a certain time on the ground or its movement was characterized by an alternation of flying (shorter distances) and continuing at a certain position.

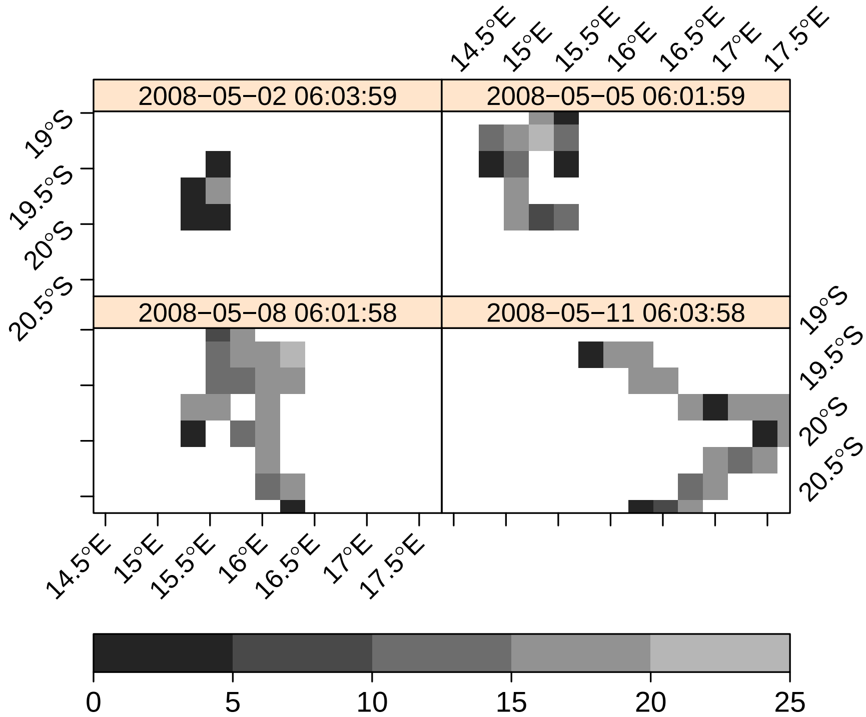

Figure 6.6.: Weighted average speed of the vulture individual X1 (*Gyps africanus*) over spatial regions and four consecutive 3-days time intervals. Result from the method `aggregate` applied to a `TracksCollection` object and a spatio-temporal grouping predicate in conjunction with `byID = TRUE`. Each panel represents one of the 3-days time intervals whose starting dates are given by the panel titles. The speed is given in metres per second.

One may also easily compare Figure 6.5 and Figure 6.6 due to the identical grouping predicate that was used in the `aggregate` calls. There is a relation between the minimal altitude of the vulture (Figure 6.5) and the weighted averaged speed (Figure 6.6), especially in the extremely low values of altitude and speed, as one also would expect.

In the following Subsection 6.4.2 the method `aggregate` for **`trajectories`** objects with a grouping predicate of class `Spatial` passed to the `by` argument is presented, and for its illustration the pigeon example data sets introduced in Section 4.1 are used.

6.4.2. Spatial aggregation of trajectories objects

This subsection presents the meaningful aggregation of trajectory objects defined in the package **trajectories** over a `Spatial` grouping predicate. If the method `aggregate` is applied with a `Spatial` grouping predicate the returned object is of class `STFDF`, that is characterized by a `sp` slot containing an object equivalent to the grouping predicate. The temporal information of the `STFDF` object is characterized by time instances, that are derived from the unique timestamps of those trajectory points, that are spatially intersecting the geometries of the grouping predicate. This approach respects and preserves the temporal information of the trajectories related to the returned aggregation result.

For illustration of the aggregation the pigeons example data sets, introduced in Section 4.1, are used. In particular these are two `MoveStack` objects each containing two (subsets of) trajectories of a pigeon individual. In each case the trajectories of the different individuals are sampled synchronously and their sampling rates are characterized by a high frequency with four samples per second. Due to the aggregation approach the call of `aggregate` with a `Spatial` object passed to the `by` argument needs a huge amount of computation time. To avoid that in the following example a minimal subset of the pigeons example data and a `Spatial` object with just a few geometries are created and passed to the `aggregate` method.

The `MoveStack` objects `pigeon_R_moveSt_sub` and `pigeon_S_moveSt_sub`, which represent the minimal subsets of the pigeon example data sets, are coerced to `Tracks` objects. Afterwards in each case the `Tracks` are reduced to a subset containing just the first `Track` object and these reduced `Tracks` objects are combined to a `TracksCollection` object, which is shown and validated by the following commands.

```
> # Load the small subsets of the example data sets
> data(pigeon_R_moveSt_sub, pigeon_S_moveSt_sub)
> # Validate the class
> is(pigeon_R_moveSt_sub, "MoveStack") || is(pigeon_S_moveSt_sub, "MoveStack")

[1] TRUE

> # Coerce MoveStacks to Tracks objects and validate the returned object
> pigeon_R_Tr <- as.Tracks(pigeon_R_moveSt_sub)
> pigeon_S_Tr <- as.Tracks(pigeon_S_moveSt_sub)
> is(pigeon_R_Tr, "Tracks") || is(pigeon_S_Tr, "Tracks")

[1] TRUE

> # Inspect the high frequent and synchronious sampling rate
> options(digits.secs=2)
> index(pigeon_R_Tr[1]@time[1:5])
```

```
[1] "2012-06-19 11:36:56.00 UTC"
[2] "2012-06-19 11:36:56.25 UTC"
[3] "2012-06-19 11:36:56.50 UTC"
[4] "2012-06-19 11:36:56.75 UTC"
[5] "2012-06-19 11:36:57.00 UTC"

> options(digits.secs=0)
> length(pigeon_R_Tr[1]@time)

[1] 60

> identical(index(pigeon_R_Tr[1]@time), index(pigeon_S_Tr[1]@time))

[1] TRUE

> # Create a TracksCollection with one Track object od each individual
> pigeons_TrColl <- TracksCollection(
+   list(Tracks(list(pigeon_R_Tr[1])), Tracks(list(pigeon_S_Tr[1]))))
> # Inspect the tracksCollectionData
> pigeons_TrColl@tracksCollectionData[ , -c(6, 7)]

        n     xmin     xmax     ymin     ymax
Tracks1 1 8.936704 8.939813 47.52063 47.52178
Tracks2 1 8.936752 8.939868 47.52062 47.52177
```

The output validates the statements from the two paragraphs above especially about the high frequent and synchronous sampling rates and shows that both `Track` objects from the `TracksCollection` contain 60 geometries respectively (measurement) points.

It is important to know, that the trajectories of the two pigeon individuals are following almost the same course, which may be explained due to the fact that the trajectory data comes from a project, that studied the leadership-based flock structures of homing pigeons. The `tracksCollectionData` confirms that due to almost similar minimal and maximal x and y coordinates of the two `Tracks` objects.

As an example for the method `aggregate` with a `Spatial` object passed to the `by` argument of the `aggregate` method the attribute `ground_speed` from the pigeons' trajectories is aggregated by the commands further down using `weighted.mean` as the aggregation function that is passed to the argument `FUN`. Due to the regular sampling rate there is no need to weight the particular `Track` point measurements, but due to the temporal reference of the attribute `ground_speed` weighting of the particular `Track` object *parts*, that intersect a particular spatial geometry of the grouping predicate, according to the assigned duration, is appropriate. For details about the assignment of duration to `Track` points and *parts* see Section 3.2. Consequential the arguments related to the weighting process are specified as `weight.points = "equal"` to prevent the weighting of `Track` point measurements and `weight.tracks = "byTime"` to enable the weighting of `Track` *parts* according to their assigned duration.

By the following commands a `Spatial` object covering the whole spatial extent of the `TracksCollection` `pigeons_TrColl` is created using a simple provided function that is introduced in Chapter 4. Afterwards the attribute `ground_speed` is aggregated over this `Spatial` object as described in the previous paragraph.

```
> # Create SpatialPixels covering the extent of the TracksCollection
> spPix_pTrC_dim4 <- createSpatialArealObjFromPoints(
+   as(pigeons_TrColl, "SpatialPointsDataFrame"),
+   desDim = 4, out = "SpatialPixels")
> # Dimensions of the SpatialPixels
> spPix_pTrC_dim4@grid@cells.dim

[1] 4 2

> # Aggregate the attribute 'ground_speed' using function 'weighted.mean'
> agg_pTrC_sp_wMeanSpd <- aggregate(x = pigeons_TrColl, by = spPix_pTrC_dim4,
+                                   FUN = weighted.mean, na.rm = TRUE,
+                                   use.data = "ground_speed",
+                                   weight.points = "equal",
+                                   weight.tracks = "byTime", byID = FALSE)
> # Class of returned object
> class(agg_pTrC_sp_wMeanSpd)[1]

[1] "STFDF"

> # Dimensions of the returned STFDF object
> dim(agg_pTrC_sp_wMeanSpd)

    space      time variables 
        8        60         5 

> # Summary and example rows of the data slot of the resulting STFDF object
> summary(agg_pTrC_sp_wMeanSpd@data[ , -c(3, 4)])

  ground_speed       nlocs           ntraj      
 Min.   :17.65   Min.   :1.000   Min.   :1.000  
 1st Qu.:18.65   1st Qu.:2.000   1st Qu.:2.000  
 Median :18.96   Median :2.000   Median :2.000  
 Mean   :19.00   Mean   :1.967   Mean   :1.967  
 3rd Qu.:19.33   3rd Qu.:2.000   3rd Qu.:2.000  
 Max.   :19.98   Max.   :2.000   Max.   :2.000  
 NA's   :419     NA's   :419     NA's   :419    

> agg_pTrC_sp_wMeanSpd@data[c(209:212, 217:219), -4]

    ground_speed nlocs approx_duration ntraj
209           NA    NA              NA    NA
210     19.93889     1            0.25     1
```

```
211    18.70833  1           0.25   1
212          NA NA             NA  NA
217          NA NA             NA  NA
218    19.17917  2           0.50   2
219          NA NA             NA  NA
```

The output of the commands above shows that the dimensions of the returned `STFDF` object correspond to the spatial dimensions of the spatial grouping predicate in conjunction with the (unique) temporal dimensions respectively timestamps of the `pigeons_TrColl` representing the synchronous sampling rate. The number of the synchronous timestamps was shown in the last but one block of `R` commands.

To further interpret the output of the `R` commands it is useful to remember, that the trajectories of the two pigeons are following almost the same course. Thus two synchronously sampled `Track` points mostly intersect the same spatial geometry of the grouping predicate.

The output of the `summary` command and the subset of the `data` slot show that the majority of the spatio-temporal geometries which at least correspond to one `Track` point (thus those geometries whose values are not `NA`) have two corresponding `Track` points, one of each `Tracks` object. Cases in which just one `Track` point corresponds to a spatio-temporal geometry indicate that the temporally corresponding `Track` point from the other `Tracks` object intersects another geometry of the spatial grouping predicate. In such rare cases this second `Track` point intersects in the result of this particular aggregation (mostly) a *neighbouring* spatial geometry of the grouping predicate. This is indicated by the two consecutive rows of the subset of the `data` slot, that both correspond to one `Track` point. The rows 210 and 218 correspond to the same spatial geometry of the grouping predicate due to its `length`, which is eight. Thus the row 211 corresponds to a *neighbouring* geometry of the geometry corresponding to row 210 due to the fact that spatial features are cycled before proceeding in the temporal domain.

In Figure 6.7 the aggregation results are visually presented by time series of the weighted average speed individually for each spatial geometry of the grouping predicate.

In the following subsection the ability of performing a spatial aggregation of **trajectories** objects without respecting the temporal information of these objects is shortly introduced and illustrated. The aggregation is performed using the pigeons example data sets which may further help to clarify its characteristics.

6.4.3. Spatial aggregation of trajectories objects ignoring time

Results of a *pure* spatial aggregation ignoring and loosing the temporal information of the trajectory data may be obtained by applying the `aggregate` method described in Section 6.4.1 with a `STF` object containing just one time interval which covers the whole duration of the **trajectories** object of interest.

With the further down following commands such an aggregation is performed using in each case the second trajectory of the example data sets `pigeon_R_moveStack` and

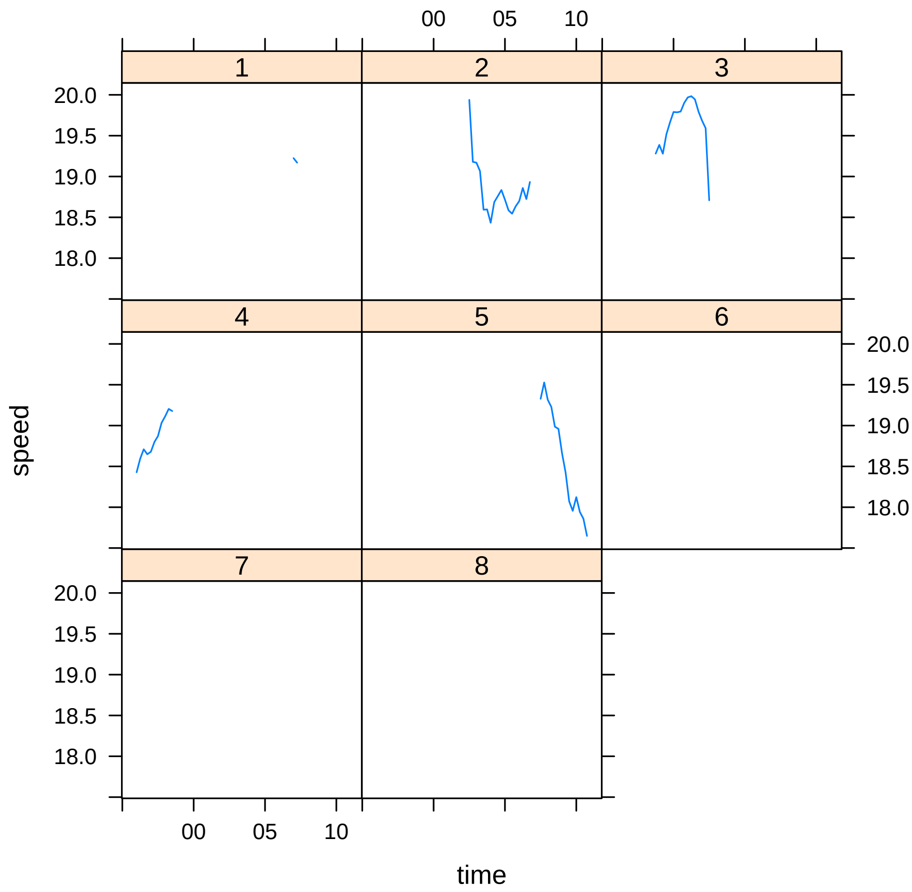

Figure 6.7.: Time series of aggregated speed values from two synchronously and regularly sampled trajectories of two pigeon individuals (*Columba livia*) flying along the same course. The time series are separated for each spatial region of the spatial grouping predicate used for the weighted aggregation.

`pigeon_S_moveStack`. The characteristics of these trajectories are similar to those used for aggregation in Subsection 6.4.2. For further details about the example data sets call for instance `?pigeon_R_moveStack` or see Section 4.1.

In this approach we are aggregating the attribute `ground_speed` again using the weighted aggregation function `weighted.mean`, and thus the arguments `weight.points` and `weight.tracks` are set in the same manner as in the call of the method `aggregate` in Subsection 6.4.2. But this time the argument `simplify` needs to be `TRUE`, so that the returned object, that is normally of class `STFDF` may be *simplified* to an object equivalent to the `sp` slot of the `STF` object passed to the argument `by`. Indeed the argument `simplify` is `TRUE` by default, but its importance in this context needs to be mentioned and it is explicitly set in the method call to clarify this importance.

Before we may call the method `aggregate` we need to coerce the `MoveStack` objects to Tracks objects, whose second Track objects are combined to a TracksCollection afterwards. Moreover we have to create an adequate STF object with one time interval covering the whole duration of all Track objects which are part of the TracksCollection. All that including the aggregation and the inspection of the result is done by the following commands.

```
> # Load data
> data(pigeon_R_moveStack, pigeon_S_moveStack)
> # Coerce to Tracks
> pigeon_R_Trcs <- as.Tracks(pigeon_R_moveStack)
> pigeon_S_Trcs <- as.Tracks(pigeon_S_moveStack)
> # Create a TracksCollection
> pigeons_TrC <- TracksCollection(list(Tracks(list(pigeon_R_Trcs[2])),
+                                      Tracks(list(pigeon_S_Trcs[2]))))
> # Create SpatialPixels covering the extent of the TracksCollection
> spPix_pTrC_dim15 <- createSpatialArealObjFromPoints(
+   as(pigeons_TrC, "SpatialPointsDataFrame"),
+   desDim = 15, out = "SpatialPixels")
> # Create a STF object with one overall time interval
> # Note: currently time zone problem in tracksCollectionData with V. 0.1-1
> #stf_px_pTrC_1t <- STF(spPix_pTrC_dim15,
> #                       time = pigeons_TrC@tracksCollectionData$tmin[1],
> #                       endTime = pigeons_TrC@tracksCollectionData$tmax[1])
> stf_px_pTrC_1t <- STF(spPix_pTrC_dim15,
+                       time = pigeon_R_Trcs@tracksData$tmin[2],
+                       endTime = pigeon_R_Trcs@tracksData$tmax[2])
> # Aggregate the attribute 'height_raw' with FUN = weighted.mean
> agg_pTrC_wMeanSpd <- aggregate(x = pigeons_TrC, by = stf_px_pTrC_1t,
+                                FUN = weighted.mean, na.rm = TRUE,
+                                use.data = "ground_speed",
+                                simplify = TRUE, weight.points = "equal",
+                                weight.tracks = "byTime", byID = FALSE)
> class(agg_pTrC_wMeanSpd)[1]

[1] "SpatialPixelsDataFrame"

> summary(agg_pTrC_wMeanSpd@data[c(1, 2, 5)])

  ground_speed       nlocs            ntraj     
 Min.   :14.57   Min.   :  2.00   Min.   :1.00  
 1st Qu.:15.83   1st Qu.: 23.00   1st Qu.:2.00  
 Median :17.64   Median : 39.00   Median :2.00  
 Mean   :17.24   Mean   : 38.32   Mean   :1.92  
 3rd Qu.:18.17   3rd Qu.: 46.00   3rd Qu.:2.00  
 Max.   :19.46   Max.   :108.00   Max.   :2.00  
 NA's   :80      NA's   :80       NA's   :80    
```

The output shows that the returned object is of class `SpatialPixelsDataFrame` and thus the object representing the aggregation result is *simplified*, because typically an object of class `STFDF` is returned.

Moreover the `summary` command gives an overview of the aggregated speed values (`ground_speed`) as well as of the number of trajectory points (`nlocs`) and the number of `Track` objects (`ntraj`), that is determined by the presence of `Track` points, for each spatial geometry of the `SpatialPixelsDataFrame`. The attribute `ntraj` shows that the majority of the spatial geometries, that are intersected by at least one `Track` point, are intersected by both `Tracks` objects, which confirms the statement from the Subsection 6.4.2, that the pigeons are flying along the same course.

A visual presentation of the aggregation result is given by Figure 6.8. The figure shows the weighted average speed values in metres per second of the two pigeon individuals for each spatial geometry of the grouping predicate.

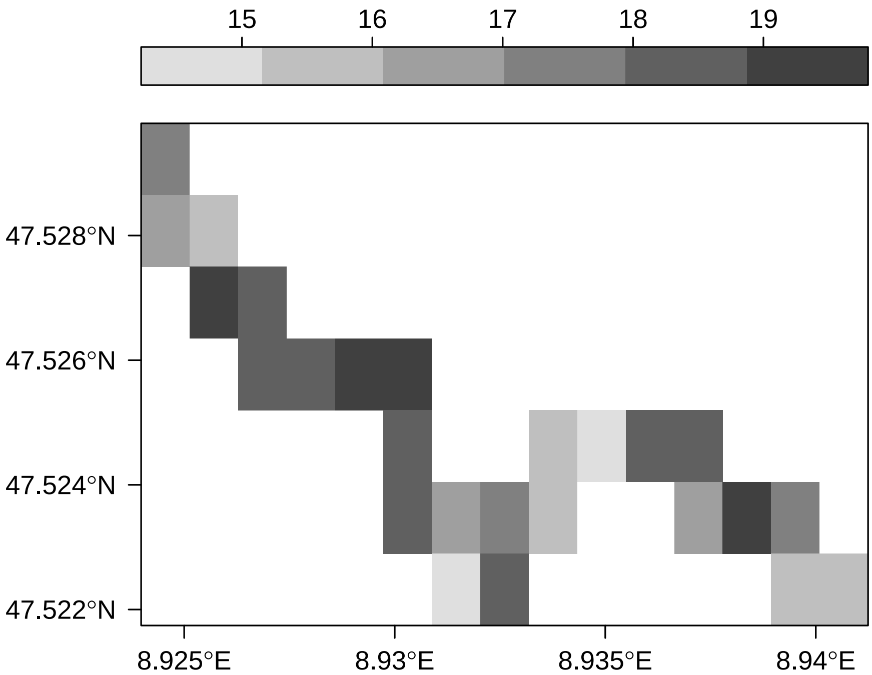

Figure 6.8.: Weighted average speed of two pigeon individuals (*Columba livia*) obtained by weighted aggregation over a `Spatial` grouping predicate with no respect of the temporal characteristics of the trajectories. Speed is given in metres per second.

7. Discussion

In this chapter the strengths and weaknesses of the implemented methods are discussed. In general meaningful methods for spatial and spatio-temporal overlay of `Track` objects and for spatial and spatio-temporal aggregation of objects of all classes for trajectories that are defined in the package **trajectories** are implemented. As grouping predicates objects of the classes `STF`, `SpatialPolygons`, `SpatialPixels` or `SpatialGrid` as well as their `data.frame` counterparts may be used. Generally the implemented methods are as far as possible consistent with the methods for overlay and aggregation defined in the packages **sp** and **spacetime** regarding to the used arguments and the data structure of the returned objects. The spatio-temporal overlay constitutes the basis of the aggregation methods.

The chosen approach for the methods `over` and `aggregate` with respect to the characteristic of the sampling rate of the trajectories is based on a simple estimation of the duration and distance which is assigned to the observed trajectory points' attributes. These estimated values are used to enable a weighted aggregation of the observed values with respect to the sampling rate.

This estimating approach is based on a vague assumption, that however enables a quite efficient and adequate weighted spatio-temporal aggregation. But due to the estimation of the values about duration and distance a bias in the aggregated attribute values can not be excluded. A better approach would exactly calculate such values and would not need such a vague assumption. Moreover the current implementation allows just weighting by duration or distance, in which a weighting of attribute values with respect to a combination of duration and distance would be desirable at least in some situations.

The method `aggregate` in conjunction with a `Spatial` grouping predicate preserves the temporal information corresponding to the trajectory points. But due to its implementation approach this method is especially interesting for aggregation of synchronously sampled trajectories of different individuals. In other cases *huge* `STFDF` objects may be returned, in which each trajectory point corresponds to a particular spatio-temporal geometry. Moreover the method requires huge computation times and thus seems to be little helpful in the daily practice. Nevertheless this approach was chosen, because a *pure* spatial aggregation with ignoring and loosing the temporal information of the trajectory points may be easily obtained by the `aggregate` method in conjunction with `STF` objects as grouping predicates. This is shown in Subsection 6.4.3.

Meaningful aggregation also depends on the aggregation functions used to aggregate the attribute data. In the current implementation the function `sum`, which is typically not meaningful regarding the aggregation of trajectory data (Stasch et al., 2014), is simply attended with a warning message, if it is passed to the aggregation function argument of the methods `over` and `aggregate`. This functionality might be extended like or linked to the approach of meaningful spatial statistics implemented in the package **mss** (Stasch et al., 2014).

The `data` slot of the objects returned by the method `aggregate` contains additionally to the aggregated attribute values some metadata. This metadata is partly based on the estimated values about duration and distance assigned to the particular trajectory points or to a set of consecutive trajectory points intersecting a particular geometry of the grouping predicate. It is *nice-to-have* but it would be desirable to provide exact values instead of estimated approximate values.

A useful functionality is the ability to create missing attribute columns in particular `Track` objects that are itself part of a `Tracks` or `TracksCollection`, if the `data` slots of these objects are inconsistent related to the existent attribute columns. These added columns are filled with `NA` values. However the comparison of the `data` slots is just based on the column names and one might think about the possibility to extent that for instance by checking also the data types of the data stored in these columns.

The method `count` provides an easy and efficient way to count the number of trajectories that spatially intersect `Spatial` objects or spatio-temporally intersect `STF` objects. The counting is based on the presence of at least one trajectory point of a `Track` object intersecting a geometry of the grouping predicate. For instance the method may be suitably applied to trajectories that have no observed attributes corresponding to trajectory points (respectively a `data` slot with zero columns) or whose observed attributes may not be aggregated in a meaningful way. The method may be used to analyse the spatial or spatio-temporal distribution of trajectories. Regarding to the class of the returned object the method `count` is inconsistent with the method `aggregate` due to the return of `Spatial` objects in cases where a `Spatial` object is passed to the `by` argument.

A great many methods regarding the (bidirectional) coercion between classes defined in **trajectories** and classes representing trajectory data, that are defined in the packages **adehabitatLT** and **move**, are implemented. These coercion methods enable an easy way to apply the implemented methods `over` and `aggregate` to objects of classes defined in those two packages as well as in particular to the huge amount of data sets stored in the movebank database.

However there are some limitations regarding the coercion. Objects of classes defined in **move** provide additional data that is in case of coercion not stored in the resulting **trajectories** objects, because this data is considered as less important and thus would (just) blow up the resulting objects. Moreover the coercion of `Tracks` objects to `MoveBurst` objects is limited due to the restriction that the particular trajectories from the `Tracks` object need to be stored in the `Tracks` object in temporal order.

Objects of class `ltraj` that is defined in the package **adehabitatLT** may contain tracked *point locations* with missing coordinates or timestamps. In the current implementation of the coercion of such objects to **trajectories** objects these locations are simply ignored and thus the resulting trajectories object represents a modified path of the tracked individual due the reduced number of point locations. One may think about an advanced approach which does not simply ignore such data but for instance interpolates the missing values.

The implemented methods are exclusively applied to animal trajectories in the context of their illustration. Generally the methods may be applied to trajectories of any kind of moving objects. One may think about trajectories of pedestrians, bicyclists, robots or motor vehicles like for instance cars. However in the case of objects that are moving controlled by an external element like for instance a car's movement is bounded by the available roads one may think about advanced aggregation approaches which directly respect this additional characteristic.

In the provided examples the methods are exclusively called with rectangular shaped geometries as (the spatial component of the) grouping predicates. This is not a limitation but it is realized due to the ability to easily create such objects as well as the lack of suitable semantically appropriate *complex* polygons. Generally the implemented methods work as well with complex polygon structures. However one need to concern that the assumption regarding to the sampling rate is based on rectangular shaped geometries (at least) when the sampling rate is based in spatial distances. Proving the requirements related to the assumption in the case of spatial sampling rates is hard to realize when complex polygons are used as (the spatial component of the) grouping predicates.

Finally it should be mentioned that the implemented methods in conjunction with a `Spatial` grouping predicate seem to work well also with objects of the classes `SpatialPoints` and `SpatialLines`. But due to the fact that on the one hand this is not intensively tested and on the other hand this is considered as little useful, accepted `Spatial` grouping predicates are limited to the classes mentioned in the first paragraph of this chapter.

8. Conclusions

This work presents an adequate and basic approach of meaningful overlay and (weighted) aggregation of trajectory point data over spatial and spatio-temporal grouping predicates, in which the basis of the aggregation methods is constitutes by the spatio-temporal overlay. All methods are implemented by S4 generic functions.

The approach of (weighted) aggregation is considered as meaningful under the assumption that the time intervals of the (temporal) sampling rate are smaller than the time intervals characterizing the spatio-temporal geometries of the grouping predicate, which may be easily evaluated. This simple but adequate assumption ensures a sufficient estimation of values about duration and distance that may be interpreted as being correspondent to the attribute values measured at the trajectory point locations. For instance related to duration these values are determined by calculating the sum of the half of the durations corresponding to both segments bordering on the appendant trajectory point. Based on this estimated values about duration and distance a suitable weighted aggregation of trajectory point attribute values with weights based on the duration or distance may be performed.

The implemented methods are applicable in a meaningful manner to trajectories of all types of moving entities as long as they fulfil the requirements derived from the explained assumption. As grouping predicates all types of spatial area-measured features including *complex* polygons may be used. In the case of *complex* polygons in conjunction with a weighted aggregation the bias due to the estimation of the weights may be higher than in the case of regular shaped rectangular features like for instance grid cells.

Aggregation by spatial grouping predicates is performed in conjunction with preserving the temporal information of the trajectory points. This approach is considered as meaningful and is especially suitable for aggregation of trajectories with synchronous sampling rates. Moreover the current implementation of this approach seems to be a little impractical due to the requirement of huge computation times.

Methods for counting the number of trajectories over spatial and spatio-temporal grouping predicates are provided. A trajectory is *counted* related to a geometry of the grouping predicate if at least one of its points intersects that geometry. The method may be used to analyse the spatial or spatio-temporal distribution of trajectories and is especially suitable for trajectories without attributes (that may be aggregated in a meaningful way).

Last but not least a great many methods regarding bidirectional coercion between classes defined in the package **trajectories** and classes defined in the packages **adehabitatLT** and **move** are implemented. Besides the usual implementation these methods are also implemented by S4 generic functions. The main objective of these methods is to access movement data stored in objects whose classes are defined in the mentioned packages, and especially to access data stored in the movebank database.

Concluding one may argue that the implemented aggregation methods provide a first basic approach to fill the gap in the lack of suitable software solutions for (weighted) aggregation of trajectory data in R with respecting their spatial and temporal domain in a meaningful way.

9. Outlook

Useful and desirable improvements related to the implemented methods as well as additional meaningful aggregation approaches that are not considered by the implemented methods are presented in this chapter. In the beginning the improvements are explained and afterwards the additional approaches are shortly presented.

An obvious remaining challenge is to consider a combination of time and space in the calculation of weights used for the weighted aggregation of observed attribute values. One problem related to that challenge is the fact that values representing time intervals or spatial distances may be based on different reference units like a time interval may be given in seconds or minutes for instance. Thus when calculating weights from time intervals and spatial distances different reference units will leads to different weights.

Calculating exact measures of duration and distance that is assigned to the observed attribute values for the purpose of weighting these values is another important challenge. That would avoid a bias in the aggregated values as well as the need of a (vague) assumption related to the sampling rate of the trajectory data.

Another challenge is an improvement of the implemented approach related to aggregation over spatial geometries with preserving and storing the temporal information related to the trajectory points. The presented approach works but need to be improved due to its enormous computation time. This might be maybe realized by a more sophisticated implementation and/or by storing the aggregated values in an object of another class than `STFDF`. This could either be an object of class `STIDF` or an appropriate class may need to be defined.

A final challenge is the improvement of the approach to judge the appropriateness of a desired aggregation due to its meaningfulness. Currently a simple and *static* warning is given if `sum` is the selected aggregation function. This should be extended by a more flexible and sophisticated approach like implemented in the package **mss** representing meaningful spatial statistics. Or it would be even better if the implemented aggregation and the approach implemented in **mss** would be further extended regarding to the aim to directly link both functionalities.

Independent from the implemented methods there are further aggregation approaches that are not considered in the context of this work but that would be part of a complete disquisition on meaningful aggregation of trajectory data.

Aggregation of trajectories over a temporal grouping predicate represents one of these additional approaches. Similar to the aggregation over space there are two imaginable approaches to aggregate over time whose difference is related to the question if the spatial information corresponding to the trajectory points should be preserved or ignored. If one would like to preserve the spatial information the (temporal) order and in conjunction with that the *path* structure of the trajectory gets lost. This would result in a spatial point pattern for each temporal region of the grouping predicate.

Another approach is about aggregation over cyclic time, for instance over hours or subsets of hours of a day. This is particularly interesting in the analysis of animal movement for instance to obtain information about typical behaviour relative to particular times of the day.

Moreover there is the idea to aggregate trajectories relative to the characteristic of one or more observed attributes. This seems not to be widely spread but there are certainly numerous cases in which this would be quite interesting. Challenging questions are how to treat the temporal and spatial domain in such an aggregation and which kind of data structure should be used to represent the result.

Finally all the implemented approaches as well as the additionally introduced ideas about aggregation of trajectory data corresponding to point locations may be transferred to the aim of aggregating trajectory data corresponding to the trajectory segments respectively to the `connections`. A meaningful aggregation of trajectory connection data is even more challenging than the aggregation of point data due to its line-like structure and the fact that they correspond not only to a distance but also to a temporal duration. In particular connections may intersect two or more spatial or spatio-temporal geometries of a grouping predicate due to its spatial and temporal dimensions which need to be respected in an aggregation procedure. A weighting approach of aggregation of connections essentially needs to respect its spatial and temporal domains.

An additional challenge occurs if the spatial (component of the) grouping predicate is characterized by *complex* polygons. In such a situation a particular connection may intersect one particular polygon even more than once which needs to be respected. The relevant connection parts intersecting that particular polygon need to be identified and the duration and distance corresponding to each of these parts need to be calculated. That represents another remaining difficult challenge.

Bibliography

Andrienko, G. and Andrienko, N. (2008). Spatio-temporal aggregation for visual analysis of movements. In *Visual Analytics Science and Technology, 2008. VAST'08. IEEE Symposium on*, pages 51–58. IEEE.

Andrienko, G. and Andrienko, N. (2010). A general framework for using aggregation in visual exploration of movement data. *The Cartographic Journal*, 47(1):22–40.

Andrienko, G., Andrienko, N., Bak, P., Keim, D., Kisilevich, S., and Wrobel, S. (2011). A conceptual framework and taxonomy of techniques for analyzing movement. *Journal of Visual Languages & Computing*, 22(3):213–232.

Andrienko, N. and Andrienko, G. (2006). *Exploratory analysis of spatial and temporal data*. Springer.

Andrienko, N. and Andrienko, G. (2011). Spatial generalization and aggregation of massive movement data. *IEEE transactions on visualization and computer graphics*, 17(2):205–219.

Andrienko, N. and Andrienko, G. (2012). Visual analytics of movement: An overview of methods, tools and procedures. *Information Visualization*.

Andrienko, N., Andrienko, G., and Gatalsky, P. (2003). Exploratory spatio-temporal visualization: an analytical review. *Journal of Visual Languages & Computing*, 14(6):503–541.

Andrienko, N., Andrienko, G., Pelekis, N., and Spaccapietra, S. (2008). Basic concepts of movement data. In *Mobility, Data Mining and Privacy*, pages 15–38. Springer.

Bivand, R. S., Pebesma, E. J., and Gómez-Rubio, V. (2008). *Applied Spatial Data Analysis With R*. Use R. Springer-Verlag, New York.

Calenge, C. (2006). The package adehabitat for the r software: tool for the analysis of space and habitat use by animals. *Ecological Modelling*, 197:1035.

Calenge, C., Dray, S., and Royer-Carenzi, M. (2009). The concept of animals' trajectories from a data analysis perspective. *Ecological Informatics*, 4(1):34–41.

D'Hondt, E., Stevens, M., and Jacobs, A. (2013). Participatory noise mapping works! an evaluation of participatory sensing as an alternative to standard techniques for environmental monitoring. *Pervasive and Mobile Computing*, 9(5):681–694.

Elen, B., Peters, J., Poppel, M. V., Bleux, N., Theunis, J., Reggente, M., and Standaert, A. (2012). The aeroflex: a bicycle for mobile air quality measurements. *Sensors*, 13(1):221–240.

Fredrikson, A., North, C., Plaisant, C., and Shneiderman, B. (1999). Temporal, geographical and categorical aggregations viewed through coordinated displays: A case study with highway incident data. In *Proceedings of the 1999 Workshop on New Paradigms in Information Visualization and Manipulation in Conjunction with the Eighth ACM Internation Conference on Information and Knowledge Management*, NPIVM '99, pages 26–34, New York, NY, USA. ACM.

Giannotti, F., Nanni, M., Pinelli, F., and Pedreschi, D. (2007). Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 330–339, New York, NY, USA. ACM.

Goldstein, J. and Roth, S. F. (1994). Using aggregation and dynamic queries for exploring large data sets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pages 23–29, New York, NY, USA. ACM.

Hijmans, R. J. (2014). *raster: Geographic data analysis and modeling.* R package version 2.3-12.

Klus, B. and Pebesma, E. (2014). *Analysing Trajectory Data in R.*

Kranstauber, B. and Smolla, M. (2014). move: Visualizing and analyzing animal track data. R package version 1.2.475.

Lenz, H.-J. and Shoshani, A. (1997). Summarizability in OLAP and statistical data bases. pages 132–143. IEEE Computer Society.

Mazón, J.-N., Lechtenbörger, J., and Trujillo, J. (2009). A survey on summarizability issues in multidimensional modeling. *Data Knowl. Eng.*, 68(12):1452–1469.

Meratnia, N. and de By, R. A. (2002). Aggregation and comparison of trajectories. In *Proceedings of the 10th ACM International Symposium on Advances in Geographic Information Systems*, GIS '02, pages 49–54, New York, NY, USA. ACM.

Pebesma, E. (2012). spacetime: Spatio-temporal data in R. *Journal of Statistical Software*, 51(7):1–30.

Pebesma, E. (2014). CRAN task view: Handling and analyzing spatio-temporal data.

Pebesma, E. and Klus, B. (2014). *trajectories: Classes and methods for trajectory data.* R package version 0.1-2.

Pebesma, E. J. and Bivand, R. S. (2005). Classes and methods for spatial data in R. *R News*, 5(2):9–13.

R Development Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Roduit, P. (2009). Trajectory analysis using point distribution models.

Ryan, J. A. and Ulrich, J. M. (2014). *xts: eXtensible Time Series*. R package version 0.9-7.

Santos, C. D., Neupert, S., Lipp, H.-P., Wikelski, M., and Dechmann, D. K. N. (2014). Temporal and contextual consistency of leadership in homing pigeon flocks. *PLoS ONE*, 9(7):e102771.

Schabenberger, O. and Gotway, C. A. (2004). *Statistical methods for spatial data analysis*. CRC Press.

Spiegel, O., Getz, W. M., and Nathan, R. (2013). Factors influencing foraging search efficiency: Why do scarce lappet-faced vultures outperform ubiquitous white-backed vultures? *The American Naturalist*, 181(5):E102–E115.

Stasch, C., Scheider, S., Pebesma, E., and Kuhn, W. (2014). Meaningful spatial prediction and aggregation. *Environmental Modelling & Software*, 51:149–165.

Stevens, S. S. (1946). On the theory of scales of measurement. *Science (New York, N.Y.)*, 103(2684):677–680.

Sumner, M. D. (2013). *trip: Spatial analysis of animal track data*. R package version 1.1-17.

Wikelski, M. and Kays, R. (2011). Movebank: archive, analysis and sharing of animal movement data. *World Wide Web electronic publication*.

Zeileis, A. and Grothendieck, G. (2005). zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27.

A. Appendix

A.1. Preparation and coercion of vulture_moveStack example data

As described in detail in the Chapter 4 the vulture_moveStack data set contains three individuals tracked over 14 days from around 6 a.m. to 6 p.m. With the implemented coercion methods an object of class `MoveStack` may be coerced to an object of class `Tracks` or `TracksCollection` as described in Section 5.1. Because `vulture_moveStack` contains three individuals the latter would be appropriate:

```
> # library(trajaggr)
> data(vulture_moveStack)
> class(vulture_moveStack)[1]

[1] "MoveStack"

> vulture_TrC <- as(vulture_moveStack, "TracksCollection")
> dim(vulture_TrC)

       IDs     tracks geometries
         3          3       9639

> dim(vulture_TrC[1])

    tracks geometries
         1       3250
```

The output shows that the each `Tracks` object from the created `TracksCollection` just contains one `Track` (representing 14 days). Due to the definition of `Track` objects from the **trajectories** this is inadequate considering the fact that tracking was just performed during the day and not at night. Thus there is the need to restructure the data which will be performed directly on the `MoveStack` object by the following commands. Subsequent the coercion is applied and a `TracksCollection` is created:

```
> # Create a list of adequate MoveStack objects
> # library(move)
> vulture_moveObjList <- move::split(vulture_moveStack)
> vulture_moveStackList <- lapply(vulture_moveObjList, function(x) {
```

```
+   dates <- as.Date(x@timestamps)
+   uniquedates <- unique(dates)
+   moveObjList <- lapply(seq_along(uniquedates), function(y) {
+     w <- which(dates == uniquedates[y])
+     x[w]
+   })
+   ms <- move::moveStack(moveObjList)
+   # However the timezone in the timestamps slot is dropped
+   # when applying move::moveStack. Need to redefine the timezone...
+   attr(ms@timestamps, "tzone") <- attr(ms@data$timestamp, "tzone")
+   return(ms)
+ })
> # Coerce MoveStack objects to Tracks objects
> Tracks_X1 <- as(vulture_moveStackList[[1]], "Tracks")
> Tracks_X2 <- as(vulture_moveStackList[[2]], "Tracks")
> Tracks_X3 <- as(vulture_moveStackList[[3]], "Tracks")
> # Create TracksCollection
> vulture_TrC <- TracksCollection(list(Tracks_X1 = Tracks_X1,
+                                      Tracks_X2 = Tracks_X2,
+                                      Tracks_X3 = Tracks_X3))
> dim(vulture_TrC)

       IDs     tracks geometries
         3         42       9639

> names(vulture_TrC@tracksCollection)

[1] "Tracks_X1" "Tracks_X2" "Tracks_X3"

> dim(vulture_TrC[1])

    tracks geometries
        14       3250

> names(vulture_TrC[1]@tracks)

 [1] "X1"   "X11"  "X12"  "X13"  "X14"  "X15"  "X16"  "X17"
 [9] "X18"  "X19"  "X110" "X111" "X112" "X113"
```

We realize from the last output that the structure of the TracksCollection has changed and that (for instance) the first Tracks object from the TracksCollection now contains 14 Track objects instead of one. This newly created TracksCollection object is used to illustrate implemented methods.

A.2. Validation of the coercion of move objects to trajectories objects

```
> ### Coercion of MoveStack to Tracks
> # library(trajaggr)
> data(vulture_moveStack)
> class(vulture_moveStack)[1]

[1] "MoveStack"

> v_Tracks <- as(vulture_moveStack, "Tracks")
> class(v_Tracks)[1]

[1] "Tracks"

> # Compare as-method and generic method
> v_Tracks_gen <- as.Tracks(vulture_moveStack)
> identical(v_Tracks, v_Tracks_gen)

[1] TRUE

> # Compare some selected data (first and third Track)
> rowNames <- lapply(v_Tracks@tracks, function(x) row.names(x@data))
> identical(vulture_moveStack@data[rowNames[[1]], ],
+           v_Tracks@tracks[[1]]@data[rowNames[[1]], ])

[1] TRUE

> identical(vulture_moveStack@data[rowNames[[3]], ],
+           v_Tracks@tracks[[3]]@data[rowNames[[3]], ])

[1] TRUE

> # Compare some selected coords (first Track)
> nrows <- lapply(v_Tracks@tracks, function(x) nrow(x@data))
> identical(vulture_moveStack@coords[1:nrows[[1]], ],
+           v_Tracks@tracks[[1]]@sp@coords)

[1] TRUE

> # Compare some selected timestamps (second Track)
> identical(as.numeric( # due to ignore attribute tclass from Track time
+   vulture_moveStack@timestamps[(nrows[[1]] + 1):(nrows[[1]]+nrows[[2]])]),
+           as.numeric(index(v_Tracks@tracks[[2]]@time)))

[1] TRUE
```

```
> # Compare tracksData and idData (ignoring row.names)
> idDataNames <- names(vulture_moveStack@idData)
> identical(data.frame(vulture_moveStack@idData, row.names = NULL),
+           data.frame(v_Tracks@tracksData[ , idDataNames], row.names = NULL))

[1] TRUE

> ### Coercion of MoveStack to TracksCollection
> v_TrColl <- as.TracksCollection(vulture_moveStack)
> class(v_TrColl)[1]

[1] "TracksCollection"

> # Compare some selected data (Track of second Tracks object)
> rowNames <- row.names(v_TrColl@tracksCollection[[2]]@tracks[[1]]@data)
> identical(vulture_moveStack@data[rowNames, ],
+           v_TrColl@tracksCollection[[2]]@tracks[[1]]@data[rowNames, ])

[1] TRUE

> # Compare some selected coords (Track of first Tracks object)
> identical(vulture_moveStack@coords[1:nrows[[1]], ],
+           v_TrColl@tracksCollection[[1]]@tracks[[1]]@sp@coords)

[1] TRUE

> # Compare some selected timestamps (Track of first Tracks object)
> identical(as.numeric(vulture_moveStack@timestamps[1:nrows[[1]]]),
+           as.numeric( # due to ignore attribute tclass from Track time
+             index(v_TrColl@tracksCollection[[1]]@tracks[[1]]@time)))

[1] TRUE

> ### Coercion of MoveBurst to Tracks
> # Create a MoveBurst object from the first individuals' first day track
> # First subset the Move object to the first tracked day
> v_X1_Move <- vulture_moveStack[[1]]
> day1 <- which(as.Date(v_X1_Move@timestamps) ==
+                 as.Date(v_X1_Move@timestamps[1]))
> v_X1_1_Move <- v_X1_Move[day1]
> # Create MoveBurst object with bursts specifying the type of
> # locomotion (on.ground or flying) based on vultures' speed
> behav <- rep("on.ground", length(day1))
> behav[which(v_X1_Move@data$ground_speed[day1] > 5)] <- "flying"
> v_X1_1_mb <- move::burst(v_X1_1_Move, f = behav[1:length(behav) - 1])
> # Coerce MoveBurst to Tracks ...
> v_X1_1_mbTracks <- as.Tracks(v_X1_1_mb)
> class(v_X1_1_mbTracks)[1]

[1] "Tracks"
```

A.3. Validation of the coercion of trajectories objects to move objects

As input the **trajectories** objects created in Subsection 5.1 respectively in the Appendix A.2 are used and the coercion is validated by comparing the newly created **move** objects with the original **move** objects used as input for the coercion presented in Subsection 5.1 respectively in Appendix A.2.

```
> ### Coercion of Tracks to MoveStack
> # library(trajaggr)
> class(v_Tracks)[1]

[1] "Tracks"

> # library(move)
> v_moveSt <- as.MoveStack(v_Tracks)
> class(v_moveSt)[1]

[1] "MoveStack"

> # Compare data, coords and time of original and re-coerced MoveStack
> vars <- names(vulture_moveStack@data)
> identical(vulture_moveStack@data[ , vars], v_moveSt@data[ , vars])

[1] TRUE

> identical(vulture_moveStack@coords, v_moveSt@coords)

[1] TRUE

> identical(vulture_moveStack@timestamps, v_moveSt@timestamps)

[1] TRUE

> ### Coercion of TracksCollection to MoveStack
> class(v_TrColl)[1]

[1] "TracksCollection"

> v_moveSt <- as.MoveStack(v_TrColl)
> class(v_moveSt)[1]

[1] "MoveStack"

> # Compare data, coords and time of original and re-coerced MoveStack
> identical(vulture_moveStack@data[ , vars], v_moveSt@data[ , vars])

[1] TRUE
```

```
> identical(vulture_moveStack@coords, v_moveSt@coords)

[1] TRUE

> identical(vulture_moveStack@timestamps, v_moveSt@timestamps)

[1] TRUE

> # Compare idData (ignoring row.names)
> idDataNames <- names(vulture_moveStack@idData)
> identical(data.frame(vulture_moveStack@idData, row.names = NULL),
+           data.frame(v_moveSt@idData[ , idDataNames], row.names = NULL))

[1] TRUE

> ### Coerce Tracks to MoveBurst
> class(v_X1_1_mbTracks)[1]

[1] "Tracks"

> v_X1_1_newMB <- as.MoveBurst(v_X1_1_mbTracks)
> class(v_X1_1_newMB)[1]

[1] "MoveBurst"

> # Different row.names in data.frame but identical data, e.g. height
> identical(v_X1_1_mb@data$height_raw, v_X1_1_newMB@data$height_raw)

[1] TRUE

> # as well as identical coords ...
> identical(v_X1_1_mb@coords, v_X1_1_newMB@coords)

[1] TRUE

> # ... and identical time
> identical(as.numeric(v_X1_1_mb@timestamps), # as. numeric to ignore attributes
+           as.numeric(v_X1_1_newMB@timestamps))

[1] TRUE
```

A.4. Validation of the coercion of `ltraj` objects defined in adehabitatLT to objects defined in trajectories

```
> # library(trajaggr)
> data(wildboars_4Ind_ltraj)
> class(wildboars_4Ind_ltraj[1])

[1] "ltraj" "list"

> # Coerce ltraj track of first individual (first burst) to Track object
> wb_1_Track <- as(wildboars_4Ind_ltraj[1], "Track")
> class(wb_1_Track)[1]

[1] "Track"

> # Compare some selected data incl. coords and time
> identical(wildboars_4Ind_ltraj[[1]]$x, wb_1_Track@sp@coords[ , 1])

[1] TRUE

> identical(as.numeric(wildboars_4Ind_ltraj[[1]]$date),
+           as.numeric(index(wb_1_Track@time)))

[1] TRUE

> identical(wildboars_4Ind_ltraj[[1]]$date, wb_1_Track@endTime) # time=endTime

[1] TRUE

> identical(wildboars_4Ind_ltraj[[1]]$R2n, wb_1_Track@data$R2n)

[1] TRUE

> identical(head(wildboars_4Ind_ltraj[[1]]$abs.angle, -1),
+           wb_1_Track@connections$abs.angle)

[1] TRUE

> str(attr(wildboars_4Ind_ltraj[[1]], "infolocs"))

'data.frame':        30 obs. of  1 variable:
 $ pkey: Factor w/ 119 levels "Brock.1993-07-01",..: 1 2 3 4 5 6 7 8 9 10 ...

> identical(attr(wildboars_4Ind_ltraj[[1]], "infolocs")[["pkey"]],
+           wb_1_Track@data$pkey)

[1] TRUE
```

```
> # Compare as-method and generic method
> # Note: Generic method just works with the adjusted move package!
> #wb_1_Track_gen <- as.Track(wildboars_4Ind_ltraj[1])
> #identical(wb_1_Track, wb_1_Track_gen)
>
> # Coercion of third and fourth burst which belong to the same individual
> wb_Tracks <- as(wildboars_4Ind_ltraj[3:4], "Tracks")
> class(wb_Tracks)[1]

[1] "Tracks"

> dim(wb_Tracks)

    tracks geometries
         2         40

> # Coercion of whole ltraj object (4 ind., 5 bursts) to TracksCollection
> wb_TracksColl <- as(wildboars_4Ind_ltraj, "TracksCollection")
> wb_TracksColl@tracksCollectionData

      n   xmin   xmax    ymin    ymax       tmin       tmax
Brock 1 698626 700387 3160768 3161559 1993-07-01 1993-08-31
Calou 1 699656 700419 3160553 3161678 1993-07-03 1993-08-31
Chou  2 699131 701410 3157848 3159572 1992-07-29 1993-08-30
Jean  1 699294 700306 3158012 3161450 1993-07-01 1993-08-31
```

A.5. Validation of the coercion of objects defined in trajectories to `ltraj` objects defined in adehabitatLT

The following commands illustrate and validate the coercion to `ltraj` objects by coercing the **trajectories** objects created in Appendix A.4 back to objects of class `ltraj` in conjunction with a subsequent object comparison.

```
> # Coercion of Track to ltraj
> # library(trajaggr)
> wb_1_ltraj <- as(wb_1_Track, "ltraj")
> class(wb_1_ltraj)

[1] "ltraj" "list"

> # Compare new ltraj object with original ltraj object / burst
> identical(wb_1_ltraj[[1]][1:length(wb_1_ltraj[[1]])],
+           wildboars_4Ind_ltraj[[1]][1:length(wildboars_4Ind_ltraj[[1]])])

[1] TRUE

> # Compare as-method and generic method
> # Note: Generic method just works with the adjusted move package!
> #wb_1_ltraj_gen <- as.ltraj(wb_1_Track)
> #identical(wb_1_ltraj, wb_1_ltraj_gen)
>
> # Coercion of Tracks to ltraj
> wb_3and4_ltraj <- as(wb_Tracks, "ltraj")
> class(wb_3and4_ltraj)

[1] "ltraj" "list"

> # Compare new ltraj object with original ltraj object / bursts
> len <- length(wb_3and4_ltraj[[1]])
> identical(wb_3and4_ltraj[[1]][1:len], wildboars_4Ind_ltraj[[3]][1:len])

[1] TRUE

> identical(wb_3and4_ltraj[[2]][1:len], wildboars_4Ind_ltraj[[4]][1:len])

[1] TRUE

> # Coercion of TracksCollection to ltraj
> wb_ltraj <- as(wb_TracksColl, "ltraj")
> class(wb_ltraj)

[1] "ltraj" "list"

> # Compare new ltraj object with original ltraj object / bursts
> identical(wb_ltraj, wildboars_4Ind_ltraj)

[1] TRUE
```

Statement on Plagiarism - Plagiatserklärung

Hiermit versichere ich, dass die vorliegende Arbeit über "Meaningful aggregation of trajectory data in R" selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken - auch elektronischen Medien - dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

____
(Datum, Unterschrift)

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

____
(Datum, Unterschrift)