Package ‘gtfs2gps’

July 22, 2025

Type Package

Title Converting Transport Data from GTFS Format to GPS-Like Records

Version 2.1-2

URL `https://github.com/ipeaGIT/gtfs2gps`,
`https://ipeagit.github.io/gtfs2gps/`

BugReports `https://github.com/ipeaGIT/gtfs2gps/issues`

Description Convert general transit feed specification (GTFS) data to global positioning system (GPS) records in 'data.table' format. It also has some functions to subset GTFS data in time and space and to convert both representations to simple feature format.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.5)

Suggests rmarkdown, markdown, knitr, testthat, dplyr, bit64

Imports data.table, furrr, future, gtfstools, Rcpp, units, sf, terra, sfheaders, progressr, lwgeom, checkmate

RoxygenNote 7.3.2

VignetteBuilder knitr

LinkingTo Rcpp

NeedsCompilation yes

Author Rafael H. M. Pereira [aut] (ORCID: <`https://orcid.org/0000-0003-2125-7465`>),
Pedro R. Andrade [aut, cre] (ORCID: <`https://orcid.org/0000-0001-8675-4046`>),
Joao Bazzo [aut] (ORCID: <`https://orcid.org/0000-0003-4536-5006`>),
Daniel Herszenhut [ctb] (ORCID: <`https://orcid.org/0000-0001-8066-1105`>),
Marcin Stepniak [ctb],
Marcus Saraiva [ctb] (ORCID: <`https://orcid.org/0000-0001-6218-2338`>),
Ipea - Institue for Applied Economic Research [cph, fnd]

Maintainer Pedro R. Andrade <`pedro.andrade@inpe.br`>

Repository CRAN

Date/Publication 2024-10-08 07:00:06 UTC

Contents

`adjust_arrival_departure`
Adjust the arrival and departure times of a GTFS data

Description

Some GTFS.zip data have issues related to arrival and departure time on stops. This function makes sure the GTFS has dis/embarking times at each stop. For each stop time row, this function applies the following steps:

1. If there is ‘arrival_time‘ but no ‘departure_time‘, it creates a departure_time column by summing the arrival plus a pre-defined ‘min_lag‘.

2. If there is ‘departure_time‘ but no ‘arrival_time‘, it creates an arrival_time column by subtracting a pre-defined ‘min_lag‘ from the departure.

3. If there is an ‘arrival_time‘ and a ‘departure_time‘ but their difference is smaller than ‘min_lag‘, it reduces the ‘arrival_time‘ and increases ‘departure_time‘ so that the difference will be exactly ‘min_lag‘.

Usage

```
adjust_arrival_departure(gtfs_data, min_lag = 20)
```

Arguments

`gtfs_data`	A GTFS data created with `read_gtfs`.
`min_lag`	Numeric. Minimum waiting time when a vehicle arrives at a stop. It can be a numeric or a units value that can be converted to seconds. Default is 20s.

Value

A GTFS with adjusted ‘arrival_time‘ and ‘departure_time‘ on data.table ‘stop_times‘.

Examples

```
poa <- read_gtfs(system.file("extdata/poa.zip", package="gtfs2gps"))

poa <- adjust_arrival_departure(poa)
```

`adjust_speed` *Adjust the speeds of a gps-like table created with* `gtfs2gps`

Description

Some GTFS.zip data sets might have quality issues, for example by assuming that a trip speed is unreasonably high (e.g. an urban bus running over 100 Km/h), or in other cases the ‘timestamp‘ information might be missing for some route segments. This can lead a gps-like table to have ‘NA‘ or unrealistic ‘speed‘ and ‘timestamp‘ values. This function allows the user to adjust the speed of trips and updates ‘timestamp‘ values accordingly. The user can adjust the problematic speeds by either setting a custom constant value, or by considering the average of all valid trips speed (Default). The columns ‘timestamp‘ and ‘cumtime‘ are updated accordingly.

Usage

```
adjust_speed(
  gps_data,
  min_speed = 2,
  max_speed = 80,
  new_speed = NULL,
  clone = TRUE
)
```

Arguments

`gps_data`	A GPS-like data.table created with `gtfs2gps`.
`min_speed`	Minimum speed to be considered as valid. It can be a numeric (in km/h) or a units value able to be converted to km/h. Values below minimum speed will be adjusted. Defaults to 2 km/h.
`max_speed`	Maximum speed to be considered as valid. It can be a numeric (in km/h) or a units value able to be converted to km/h. Values above maximum speed will be adjusted. Defaults to 80 km/h.

`new_speed`	Speed to replace missing values as well as values outside min_speed and max_speed range. It can be a numeric (in km/h) or a units value able to be converted to km/h. By default, ‘new_speed = NULL‘ and the function considers the average speed of the entire gps data.
`clone`	Use a copy of the gps_data? Defaults to TRUE.

Value

A GPS-like data with adjusted ‘speed‘ values. The columns ‘timestamp‘ and ‘cumtime‘ are also updated accordingly.

Examples

```
poa <- read_gtfs(system.file("extdata/poa.zip", package="gtfs2gps")) |>
  gtfstools::filter_by_shape_id("T2-1") |>
  gtfstools::filter_by_weekday(c("monday", "wednesday")) |>
  filter_single_trip()

poa_gps <- gtfs2gps(poa)
poa_gps_new <- adjust_speed(poa_gps)
```

`append_height` *Add a column with height to GPS data*

Description

Add a column named height to GPS data using a tif data as reference.

Usage

```
append_height(gps, heightfile)
```

Arguments

`gps`	A GPS data created from gtfs2gps().
`heightfile`	The pathname of a tif file with height data.

Value

The GPS data with a new column named height.

Examples

```
## Not run:
# this example takes more than 10s to run

fortaleza <- system.file("extdata/fortaleza.zip", package = "gtfs2gps")
srtmfile <- system.file("extdata/fortaleza-srtm.tif", package = "gtfs2gps")

gtfs <- read_gtfs(fortaleza) |>
  gtfstools::filter_by_shape_id("shape836-I") |>
  filter_single_trip()

fortaleza_gps <- gtfs2gps(gtfs, spatial_resolution = 500) |> append_height(srtmfile)

## End(Not run)
```

`filter_single_trip` *Filter GTFS trips in order to have one trip per shape_id*

Description

Filter a GTFS data by keeping only one trip per shape_id. It also removes the unnecessary routes and stop_times accordingly.

Usage

```
filter_single_trip(gtfs_data)
```

Arguments

`gtfs_data`	A list of data.tables read using gtfs2gps::reag_gtfs().

Value

A filtered GTFS data.

Examples

```
poa <- read_gtfs(system.file("extdata/poa.zip", package = "gtfs2gps"))

subset <- filter_single_trip(poa)
```

`filter_valid_stop_times` *Filter GTFS data using valid stop times*

Description

Filter a GTFS data read using gtfs2gps::read_gtfs(). It removes stop_times with NA values in arrival_time, departure_time, and arrival_time_hms. It also filters stops and routes accordingly.

Usage

```
filter_valid_stop_times(gtfs_data)
```

Arguments

`gtfs_data`	A list of data.tables read using gtfs2gps::reag_gtfs().

Value

A filtered GTFS data.

Examples

```
poa <- read_gtfs(system.file("extdata/poa.zip", package = "gtfs2gps"))

subset <- filter_valid_stop_times(poa)
```

`gps_as_sflinestring` *Converts a GPS-like data.table to a LineString Simple Feature (sf) object*

Description

Every interval of GPS data points between stops for each trip_id is converted into a linestring segment. The output assumes constant average speed between consecutive stops.

Usage

```
gps_as_sflinestring(gps)
```

Arguments

`gps`	A data.table with timestamp data.

Value

A simple feature (sf) object with LineString data.

Examples

```
library(gtfs2gps)

poa <- read_gtfs(system.file("extdata/poa.zip", package = "gtfs2gps"))
poa_subset <- gtfstools::filter_by_shape_id(poa, c("T2-1", "A141-1")) |>
  filter_single_trip()

poa_gps <- gtfs2gps(poa_subset)

poa_gps_sf <- gps_as_sflinestring(poa_gps)
```

`gps_as_sfpoints` *Convert GPS-like data.table to a Simple Feature points object*

Description

Convert a GPS data stored in a data.table into Simple Feature points.

Usage

```
gps_as_sfpoints(gps, crs = 4326)
```

Arguments

`gps`	A data.table with timestamp data.
`crs`	A Coordinate Reference System. The default value is 4326 (latlong WGS84).

Value

A simple feature (sf) object with point data.

Examples

```
library(gtfs2gps)

fortaleza <- read_gtfs(system.file("extdata/fortaleza.zip", package = "gtfs2gps"))
srtmfile <- system.file("extdata/fortaleza-srtm.tif", package="gtfs2gps")

subset <- fortaleza |>
  gtfstools::filter_by_weekday(c("monday", "wednesday")) |>
  filter_single_trip() |>
  gtfstools::filter_by_shape_id("shape806-I")

for_gps <- gtfs2gps(subset)
for_gps_sf_points <- gps_as_sfpoints(for_gps)
```

gtfs2gps *Convert GTFS to GPS-like data given a spatial resolution*

Description

Convert GTFS data to GPS format by sampling points using a given spatial resolution. This function creates additional points in order to guarantee that two points in a same trip will have at most a given distance, indicated as a spatial resolution. It is possible to use future package to parallelize the execution (or use argument plan). This function also uses progressr internally to show progress bars. See the example below on how to show a progress bar while executing this function.

Usage

```
gtfs2gps(
  gtfs_data,
  spatial_resolution = 100,
  parallel = TRUE,
  ncores = NULL,
  strategy = NULL,
  filepath = NULL,
  compress = FALSE,
  snap_method = "nearest2",
  continue = FALSE,
  quiet = FALSE
)
```

Arguments

`gtfs_data`	A path to a GTFS file to be converted to GPS, or a GTFS data represented as a list of data.tables.
`spatial_resolution`	The spatial resolution in meters. Default is 100m. This function only creates points in order to guarantee that the minimum distance between two consecutive points will be at most the spatial_resolution. If a given shape has two consecutive points with a distance lower than the spatial resolution, the algorithm will not remove such points.
`parallel`	Decides whether the function should run in parallel. Defaults is FALSE. When TRUE, it will use all cores available minus one using future::plan() with strategy "multisession" internally. Note that it is possible to create your own plan before calling gtfs2gps(). In this case, do not use this argument.
`ncores`	Number of cores to be used in parallel execution. When ‘parallel = FALSE‘, this argument is ignored. When ‘parallel = TRUE‘, then by default the function uses all available cores minus one.
`strategy`	This argument is deprecated. Please use argument plan instead or use future::plan() directly.

`filepath`	Output file path. As default, the output is returned when gtfs2gps finishes. When this argument is set, each route is saved into a txt file within filepath, with the name equals to its id. In this case, no output is returned. See argument compress for another option.
`compress`	Argument that can be used only with filepath. When TRUE, it compresses the output files by saving them using rds format. Default value is FALSE. Note that compress guarantees that the data saved will be read in the same way as it was created in R. If not compress, the txt extension requires the data to be converted from ITime to string, and therefore they need to manually converted back to ITime to be properly handled by gtfs2gps.
`snap_method`	The method used to snap stops to the route geometry. There are two available methods: ‘nearest1‘ and ‘nearest2‘. Defaults to ‘nearest2‘. See details for more info.
`continue`	Argument that can be used only with filepath. When TRUE, it skips processing the shape identifiers that were already saved into files. It is useful to continue processing a GTFS file that was stopped for some reason. Default value is FALSE.
`quiet`	Hide messages while processing the data? Defaults to FALSE.

Details

After creating geometry points for a given shape id, the ‘gtfs2gps()‘ function snaps the stops to the route geometry. Two strategies are implemented to do this. - The ‘nearest2‘ method (default) triangulates the distance between each stop and the two nearest points in the route geometry to decide which point the stop should be snapped to. If there is any stop that is further away to the route geometry than ‘spatial_resolution‘, the algorithm recursively doubles the ‘spatial_resolution‘ to do the search/snap of all stops. - The ‘nearest1‘ method traverses the geometry points computing their distances to the first stop. Whenever it finds a distance to the stop smaller than ‘spatial_resolution‘, then the stop will be snapped to such point. The algorithm then applies the same strategy to the next stop until the vector of stops end.

The ‘speed‘, ‘cumdist‘, and ‘cumtime‘ are based on the difference of distance and time between the current and previous row of the same trip. It means that the first data point at the first stop of each trip represens a stationary vehicle. The ‘adjust_speed()‘ function can be used to post-process the output to replace eventual ‘NA‘ values in the ‘speed‘ column.

Each stop is presented as two data points for each trip in the output. The ‘timestamp‘ value in the first data point represents the time when the vehicle arrived at that stop (corresponding the ‘arrival_time‘ column in the ‘stop_times.txt‘ file), while the ‘timestamp‘ in the second data point represents the time when the vehicle departured from that stop (corresponding the ‘departure_time‘ column in the ‘stop_times.txt‘ file). The second point considers that the vehicle is stationary at the stop, immediately before departing.

Some GTFS feeds do not report embark/disembark times (so ‘arrival_time‘ and ‘departure_time‘ are identical at the same stop). In this case, the user can call the ‘adjust_arrival_departure()‘ function to set the minimum time each vehicle will spend at stops to embark/disembark passengers.

To avoid division by zero, the minimum speed of vehicles in the output is 1e-12 Km/h, so that vehicles are never completely stopped.

Value

A ‘data.table‘, where each row represents a GPS point. The following columns are returned (units of measurement in parenthesis): dist and cumdist (meters), cumtime (seconds), shape_pt_lon and shape_pt_lat (degrees), speed (km/h), timestamp (hh:mm:ss).

Examples

```
library(gtfs2gps)

gtfs <- read_gtfs(system.file("extdata/poa.zip", package = "gtfs2gps")) |>
  gtfstools::filter_by_shape_id("T2-1") |>
  filter_single_trip()

poa_gps <- progressr::with_progress(gtfs2gps(gtfs, quiet=TRUE))
```

`gtfs_shapes_as_sf` *Convert GTFS shapes to simple feature object*

Description

Convert a GTFS shapes data loaded using gtfs2gps::read_gtf() into a line simple feature (sf).

Usage

```
gtfs_shapes_as_sf(gtfs, crs = 4326)
```

Arguments

`gtfs`	A GTFS data.
`crs`	The coordinate reference system represented as an EPSG code. The default value is 4326 (latlong WGS84)

Value

A simple feature (sf) object.

Examples

```
poa <- read_gtfs(system.file("extdata/saopaulo.zip", package = "gtfs2gps"))
poa_sf <- gtfs_shapes_as_sf(poa)
```

`gtfs_stops_as_sf` *Convert GTFS stops to simple feature object*

Description

Convert a GTFS stops data loaded using gtfs2gps::read_gtf() into a point simple feature (sf).

Usage

```
gtfs_stops_as_sf(gtfs, crs = 4326)
```

Arguments

`gtfs`	A GTFS data.
`crs`	The coordinate reference system represented as an EPSG code. The default value is 4326 (latlong WGS84)

Value

A simple feature (sf) object.

Examples

```
poa <- read_gtfs(system.file("extdata/poa.zip", package = "gtfs2gps"))
poa_shapes <- gtfs_shapes_as_sf(poa)
poa_stops <- gtfs_stops_as_sf(poa)
```

`read_gtfs` *Read GTFS data into a list of data.tables*

Description

Read files of a zipped GTFS feed and load them to memory as a list of data.tables. It will load the following files: "shapes.txt", "stop_times.txt", "stops.txt", "trips.txt", "agency.txt", "calendar.txt", "routes.txt", and "frequencies.txt", with this last four being optional. If one of the mandatory files does not exit, this function will stop with an error message.

Usage

```
read_gtfs(gtfszip, quiet = FALSE)
```

Arguments

`gtfszip`	A zipped GTFS data.
`quiet`	A logical. Whether to hide log messages and progress bars. Defaults to ‘FALSE‘.

Value

A list of data.tables, where each index represents the respective GTFS file name.

Examples

```
poa <- read_gtfs(system.file("extdata/poa.zip", package = "gtfs2gps"))
```

`remove_invalid` *Remove invalid objects from GTFS data*

Description

Remove all objects from GTFS data that are not used in all relations that they are required to be. That is, agency-routes relation (agency_id), routes-trips relation (route_id), trips-shapes relation (shape_id), trips-frequencies relation (trip_id), trips-stop_times relation (trip_id), stop_times-stops relation (stop_id), and trips-calendar relation (service_id), recursively, until GTFS data does not reduce its size anymore. For example, if one agency_id belongs to routes but not to agency will be removed. This might cause one cascade removal of objects in other relations that originally did not have any inconsistency.

Usage

```
remove_invalid(gtfs_data, only_essential = TRUE, prompt_invalid = FALSE)
```

Arguments

`gtfs_data`	A list of data.tables read using gtfs2gps::reag_gtfs().
`only_essential`	Remove only the essential files? The essential files are all but agency, calendar, and routes. Default is TRUE, which means that agency-routes, routes-trips, and trips-calendar relations will not be processed as restrictions to remove objects.
`prompt_invalid`	Show the invalid objects. Default is FALSE.

Value

A subset of the input GTFS data.

Examples

```
poa <- read_gtfs(system.file("extdata/poa.zip", package = "gtfs2gps"))
object.size(poa)
subset <- remove_invalid(poa)
object.size(subset)
```

`simplify_shapes` *Simplify shapes of a GTFS file*

Description

Remove points from the shapes of a GTFS file in order to reduce its size. It uses Douglas-Peucker algorithm internally.

Usage

```
simplify_shapes(gtfs_data, tol = 0)
```

Arguments

`gtfs_data`	A list of data.tables read using gtfs2gps::read_gtfs().
`tol`	Numerical tolerance value to be used by the Douglas-Peucker algorithm. The default value is 0, which means that no data will be lost.

Value

A GTFS data whose shapes is a subset of the input data.

`test_gtfs_freq` *Test whether a GTFS feed is frequency based*

Description

Test whether a GTFS feed is frequency based or whether it presents detailed time table for all routes and trip ids.

Usage

```
test_gtfs_freq(gtfs)
```

Arguments

`gtfs`	A GTFS data set stored in memory as a list of data.tables/data.frames.

Value

A string "frequency" or "simple".

`write_gtfs` *Write GTFS data into a zip file*

Description

Write GTFS stored in memory as a list of data.tables into a zipped GTFS feed. This function overwrites the zip file if it exists.

Usage

```
write_gtfs(gtfs, zipfile, overwrite = TRUE, quiet = FALSE)
```

Arguments

`gtfs`	A GTFS data set stored in memory as a list of data.tables/data.frames.
`zipfile`	The pathname of a .zip file to be saved with the GTFS data.
`overwrite`	A logical. Whether to overwrite an existing `.zip` file. Defaults to `TRUE`.
`quiet`	A logical. Whether to hide log messages and progress bars. Defaults to `TRUE`.

Value

The status value returned by the external zip command, invisibly.

Examples

```
# read a gtfs.zip to memory
poa <- read_gtfs(system.file("extdata/poa.zip", package = "gtfs2gps")) |>
  gtfstools::filter_by_shape_id("T2-1") |>
  filter_single_trip()

# write GTFS data into a zip file
write_gtfs(poa, paste0(tempdir(), "/mypoa.zip"))
```

Index