Package 'GTFShift'

September 27, 2026

Type Package

Title Explore and Analyse General Transit Feed Specification (GTFS) Files with a Focus on Urban Mobility

Version 1.0.0

Description A bundle of methods to harmonize GTFS and OSM data, enabling the integration and exploration of different layers of transit data, starting with the planned operations (GTFS), but also the infrastructure topology (OSM) and real-time information (GTFS-RT).

License GPL

URL https://github.com/U-Shift/GTFShift, https://u-shift.github.io/GTFShift/

Depends R (>= 4.1.0)

Imports tidytransit, gtfstools, sf, tidyselect, tidyr, lubridate, httr, jsonlite, dplyr, osmdata, stringr, callr, purrr, rlang, xml2, withr

Suggests knitr, rmarkdown, mapview, osmextract, rosmium, testthat, zip, gtfsrouter (>= 0.1.4), reticulate, parallel, RProtoBuf, stplanr, lwgeom, progress, stringi, spelling

VignetteBuilder knitr

Config/Needs/website rmarkdown

Config/roxygen2/version 8.1.0

Encoding UTF-8

Language en-US

NeedsCompilation no

Author Gonçalo F. Matos [aut, cre] (ORCID: <https://orcid.org/0009-0001-3489-1732>), Rosa Félix [aut] (ORCID: <https://orcid.org/0000-0002-5642-6006>), Miguel Relvas Pires [ctb]

Maintainer Gonçalo F. Matos <goncaloafmatos@tecnico.pt>

Repository CRAN

Date/Publication 2026-09-27 16:30:21 UTC

Contents

`calendar_nextBusinessWednesday` *Get next business Wednesday*

Description

Get next business Wednesday

Usage

```
calendar_nextBusinessWednesday(start_date = Sys.Date(), country_code = "PT")
```

Arguments

start_date	String (Default Sys.Date()). Reference date.
country_code	String (Default PT). Country code in the format ISO 3166-1 alpha-2. When provided, public holidays are considered.

Details

Find the next Wednesday that is not a holiday. When country is given, public holidays are considered, using Nager.Date API.

Value

Date. The next business Wednesday date.

Examples

```
# Example of Portuguese holiday (10/06/2026) ignored
GTFShift::calendar_nextBusinessWednesday(start_date = "2026-06-09", country_code="PT")

# Example of Hong Kong holiday (01/07/2026) ignored
GTFShift::calendar_nextBusinessWednesday(start_date = "2026-06-30", country_code="HK")
```

---

classify_frequency_los

Classify bus frequency level of service based on HCM

Description

Classify bus frequency level of service based on HCM

Usage

```
classify_frequency_los(frequencies, frequency_col = "frequency")
```

Arguments

frequencies	data.frame. Data frame with frequency information.
frequency_col	String (Default "frequency"). Name of the column with frequency values.

Details

Classifies bus frequency level of service (LOS) based on the Highway Capacity Manual (HCM) 2000 guidelines on "Service Frequency LOS for Urban Scheduled Transit Service" (Exhibit 27-1).

Refer to vignette("classify") for more details on this classification.

Value

data.frame. Input data frame with an additional column `frequency_los` indicating the LOS classification.

Examples

```
# Subset GTFS for one route only, for demo purposes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)
gtfs <- GTFShift::filter_by_route_name(gtfs, c("1", "2", "3", "4"))

# Get route frequency
frequency_analysis <- GTFShift::get_route_frequency_hourly(
  gtfs,
  date = gtfs$calendar$start_date[1]
)

# Compute LOS
frequency_los = GTFShift::classify_frequency_los(frequency_analysis)

frequency_los |>
  sf::st_drop_geometry() |>
  dplyr::select(route_id, frequency_los)
```

`create_calendar` *Create calendar.txt from calendar_dates.txt*

Description

Create calendar.txt from calendar_dates.txt

Usage

```
create_calendar(gtfs)
```

Arguments

`gtfs`	tidygtfs. GTFS feed.

Details

When `calendar_dates.txt` declares all service dates, `calendar.txt` becomes optional in the GTFS feed specification. However, to perform some operations, this table might be necessary.

This method allows to create a `calendar.txt` table, based on the `calendar_dates.txt`. It performs an approximation, considering, for each `service_id`, the minimum and maximum dates and setting each week day to true if it has any date that matches that date. The results might not be 100

Value

data.frame. Table for calendar.txt.

Examples

```
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_ttsl_sample_no_shapes.zip", package = "GTFShift")
)

head(gtfs$calendar_dates |> dplyr::filter(exception_type == 1))

gtfs_calendar <- GTFShift::create_calendar(gtfs)

gtfs_calendar
```

---

create_shapes_from_sf *Build shapes from simple feature object*

Description

Build shapes from simple feature object

Usage

```
create_shapes_from_sf(
  sf_shapes,
  gtfs,
  metric_crs = 3857,
  shape_dist_traveled = FALSE
)
```

Arguments

sf_shapes	sf object associating shape_id with an sf object (either LINESTRING or MULTILINESTRING).
gtfs	tidygtfs. GTFS feed.
metric_crs	numeric (Default 3857). EPSG code for a metric CRS used when computing distances (passed to multiline_to_sorted_linestring).
shape_dist_traveled	Boolean (Default FALSE). If TRUE, computes shape_dist_traveled for each generated shape.

Details

This function builds the shapes.txt file from a simple feature object.

It first converts any MULTILINESTRING geometries to LINESTRING geometries using the `multiline_to_sorted_linest` using a point guide per shape: all ordered stops when the selected trip is circular (first and last `stop_id` are equal), or the first two stops otherwise. Then, it converts the LINESTRING geometries to a data.frame representing a GTFS shapes table using `gtfstools::convert_sf_to_shapes`.

Coordinates are 4326 (WGS 84) by default, following GTFS specifications.

Optionally, when `shape_dist_traveled = TRUE`, it estimates cumulative distance along each shape for all generated points and appends this as `shape_dist_traveled`. This metric is computed in the units of `metric_crs`, using `GTFShift::project_points_along_geometry()`.

Value

data.frame. A GTFS shapes table. Includes `shape_dist_traveled` if `shape_dist_traveled = TRUE`.

See Also

`gtfstools::convert_sf_to_shapes()`

`GTFShift::multiline_to_sorted_linestring()`

`GTFShift::project_points_along_geometry()`

Examples

```
# Load sample GTFS
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)

# Load TCB OSM routes sample linestring
osm_routes = sf::st_read(
  system.file("extdata/samples", "osm_routes_tcb.gpkg", package = "GTFShift"),
  quiet = TRUE
) |> dplyr::filter(shape_id %in% gtfs$shapes$shape_id) |> dplyr::sample_n(1)

head(osm_routes)

# Create shapes.txt for geometries
shapes_txt <- GTFShift::create_shapes_from_sf(
  osm_routes, gtfs,
  metric_crs = 3763, # Make sure to addapt to the projection that better suits your location
  shape_dist_traveled = TRUE
)

head(shapes_txt)
```

create_shapes_from_stops *Build shapes from GTFS stops data*

Description

Build shapes from GTFS stops data

Usage

```
create_shapes_from_stops(gtfs)
```

Arguments

gtfs	tidygtfs. GTFS feed.

Details

The function builds the shapes.txt file from the stop_times.txt and stops.txt files, by grouping trips with the same stop sequence and assigning them the same shape_id. The resulting shapes are a simplified version of the original ones, as they do not take into account the actual path followed by the vehicles, but only the stop sequence. This can be useful for some applications that do not require high precision in the shapes, and can be used as a fallback when the original feed does not include shapes.txt file.

Value

tidygtfs. The GTFS feed with the shapes table defined and the trips table updated with the matching shape_id.

Examples

```
# Load GTFS without shapes
gtfs <- tidytransit::read_gtfs(
  system.file("extdata/samples", "gtfs_ttsl_sample_no_shapes.zip", package = "GTFShift")
)

summary(gtfs)

# Create shapes from GTFS stops data
gtfs_with_shapes <- GTFShift::create_shapes_from_stops(gtfs)

head(gtfs_with_shapes$shapes)

head(
  gtfs_with_shapes$trips |>
    dplyr::select(trip_id, shape_id) |>
    dplyr::distinct(shape_id, .keep_all = TRUE)
)
```

```
summary(gtfs_with_shapes)
```

`filter_by_agency` *Filter GTFS feed by agency*

Description

Filter GTFS feed by agency

Usage

```
filter_by_agency(gtfs, id = NA, name = NA)
```

Arguments

`gtfs`	tidygtfs. GTFS feed.
`id`	Integer (Optional when name). Ids of the agency.
`name`	String (Optional when id). Name of the agency.

Details

Allows to filter a GTFS feed for the agency, using the id, name or both. Returns empty feed it none provided.

Value

tidygtfs. The filtered GTFS feed.

Examples

```
# Load sample feed with multiple agencies
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_merged_sample.zip", package = "GTFShift")
)

summary(gtfs)


# Filter by id
gtfs_id_8 = gtfs |> GTFShift::filter_by_agency(id = "8")

summary(gtfs_id_8)


# Filter by name
gtfs_ttsl <- gtfs |> GTFShift::filter_by_agency(name = "TTSL - Transtejo Soflusa")
```

```
summary(gtfs_ttsl)
```

`filter_by_modes` *Filter GTFS feed by mode*

Description

Filter GTFS feed by mode

Usage

```
filter_by_modes(gtfs, modes = list())
```

Arguments

`gtfs`	tidygtfs. GTFS feed.
`modes`	Integer[]. A list with the ids of modes to consider.

Details

Allows to filter a GTFS feed for the type of transportation used, allowing for a more narrow analysis of multimodal files. Refer to `routes.txt` `route_type` parameter on GTFS documentation for more details.

Value

tidygtfs. The filtered GTFS feed.

Examples

```
# Load sample feed with multiple modes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_merged_sample.zip", package = "GTFShift")
)

gtfs$routes |> dplyr::select(route_id, route_type)

summary(gtfs)


# Filter by bus mode (ferry agency should be excluded)
gtfs_bus <- gtfs |> GTFShift::filter_by_modes(modes = c(3))

gtfs_bus$routes |> dplyr::select(route_id, route_type)

summary(gtfs_bus)
```

filter_by_route_name *Filter GTFS feed by route name*

Description

Filter GTFS feed by route name

Usage

```
filter_by_route_name(gtfs, values, short_name = TRUE, exact_match = TRUE)
```

Arguments

gtfs	tidygtfs. GTFS feed.
values	String[]. List of the route names to filter the feed.
short_name	Boolean. If TRUE, query for route_short_name, otherwise, route_long_name is considered.
exact_match	Boolean. If TRUE, route name is queried for an exact match, otherwise, partial match is considered.

Details

On a GTFS feed, the route_id rarely matches the real name of the route, that can range from numbers, letters, words or combinations of both. This method allows to filter the feed for the route short or long name, with a partial or exact match.

Value

tidygtfs. The filtered GTFS feed.

Examples

```
# Load GTFS
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)

summary(gtfs)


# Filter by route
gtfs_route <- GTFShift::filter_by_route_name(gtfs, c("4"))

summary(gtfs_route)
```

`get_network_extension` *Get network routes extension*

Description

Get total extension of GTFS feed routes

Usage

```
get_network_extension(
  gtfs,
  route_identifier = "route_id",
  direction_wise = TRUE,
  unified = FALSE,
  date = GTFShift::calendar_nextBusinessWednesday(),
  use_osm_routes = NA,
  metric_crs = 3857
)
```

Arguments

`gtfs`	tidygtfs. GTFS feed.
`route_identifier`	String. (Default `"route_id"`). routes.txt attribute that identifies routes. Accepted values: route_id, route_short_name, route_long_name.
`direction_wise`	Boolean (Default `TRUE`). If TRUE, extension considers sum of both directions. Otherwise, only one direction is considered.
`unified`	Boolean (Default `FALSE`). If TRUE, overlapping route segments are only counted once in the total extension.
`date`	Date (Default `GTFShift::calendar_nextBusinessWednesday()`). Reference date to consider when analyzing the GTFS file.
`use_osm_routes`	osmdata::opq (Default NA). If overpass query for transit network is defined, analysis is performed considering OSM route geometry, using `GTFShift::osm_shapes_to_routes`.
`metric_crs`	Integer or character (Default 3857). Projected CRS used to compute route lengths in meters.

Details

This method calculates the sum of the GTFS feed routes length, considering, for each, the shape of the variant with the highest frequency for the given date (using `GTFShift::get_route_frequency_hourly()`). For a detailed example, see the `vignette("analyse")`.

Value

Numeric. The routes extension, in meters.

See Also

```
GTFShift::get_route_frequency_hourly()
```

Examples

```
# Load GTFS
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip",
  package = "GTFShift"
))

# Get route extension
GTFShift::get_network_extension(
  gtfs,
  metric_crs = 3763, # Make sure to addapt to the projection that better suits your location
  date = gtfs$calendar$start_date[1]
)
```

`get_prioritisation_stats` *Get prioritisation stats*

Description

Get statistics about lane prioritisation

Usage

```
get_prioritisation_stats(
  lane_prioritisation,
  weight = c("length", "frequency"),
  metric_crs = 3857
)
```

Arguments

`lane_prioritisation`	sf data.frame. Lane prioritisation.
`weight`	Character. Weight to use for weighted mean. Accepted values: "length", "frequency".
`metric_crs`	Integer or character (Default 3857). Projected CRS used to compute lengths in meters.

Value

List. Statistics about lane prioritisation, with the following attributes:

extension Total length of the prioritised network, in meters.

extension_bus_lane Total length of the bus lane segments, in meters.

speed_avg Average speed of the prioritised network, in km/h.

speed_min Minimum speed of the prioritised network, in km/h.

speed_max Maximum speed of the prioritised network, in km/h.

n_lanes_circulation_avg Average number of lanes in the prioritised network.

n_lanes_circulation_min Minimum number of lanes in the prioritised network.

n_lanes_circulation_max Maximum number of lanes in the prioritised network.

Examples

```
# Subset GTFS for one route only, for demo purposes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)
gtfs <- GTFShift::filter_by_route_name(gtfs, c("4"))

# Build query and prepare osm extract (possible to use API as alternative)
q <- osmdata::opq(bbox = sf::st_bbox(tidytransit::shapes_as_sf(gtfs$shapes))) |>
  osmdata::add_osm_feature(key = "route", value = "bus") |>
  osmdata::add_osm_feature(key = "operator", value = "Transportes Colectivos do Barreiro")
osm_file <- system.file("extdata/samples", "osmextract_tcb_network.pbf", package = "GTFShift")

# Prioritise lanes
lane_prioritisation <- GTFShift::prioritise_lanes(
  gtfs, q,
  osm_file = osm_file,
  date = gtfs$calendar$start_date[1]
)

# Get statistics for prioritisation
stats <- GTFShift::get_prioritisation_stats(lane_prioritisation, metric_crs = 3763)

data.frame(metric = names(stats), value = unlist(stats, use.names = FALSE))
```

---

get_route_frequency_hourly

Get aggregated frequency per hour for each bus route

Description

For each route, returns the number of departures aggregated per hour and direction.

Usage

```
get_route_frequency_hourly(
  gtfs,
  date = GTFShift::calendar_nextBusinessWednesday(),
  use_osm_routes = NA,
  overline = FALSE
)
```

Arguments

gtfs	tidygtfs. GTFS feed.
date	Date (Default GTFShift::calendar_nextBusinessWednesday()). Reference date to consider when analyzing the GTFS file.
use_osm_routes	osmdata::opq (Default NA). If overpass query for transit network is defined, analysis is performed considering OSM route geometry, using GTFShift::osm_shapes_to_routes().
overline	Boolean (Default FALSE). If TRUE, routes are aggregated using stplanr::overline2(), overlapping lines and converting them into a single route network.

Details

This method analyses the GTFS feed for a representative day, generating for each route the number of services aggregated per hour and direction. It assumes the time of departure at the first stop as a reference for each trip geometry.

By default, it estimates the next business Wednesday, relevant for the peak hour.

The overline parameter enables the aggregation of bus routes that share common line segments, returning a sum of frequencies per road segment, using stplanr::overline2().

Optionally, using use_osm_routes parameter, it retrieves the geometries from OpenStreetMap by matching the tag gtfs:shape_id, overwriting the original GTFS shapes.txt. This is particularly useful if the GTFS shapes do not share the same geometry. For instance, if the edges of the lines do not overlap or do not follow the same route-over-the-road – which is very common, even besides GTFS recommendation – geometries might not be aggregated correctly, causing inconsistent results. By relying on a common road network, such as OSM, it is possible to overcome this issue and aggregate the bus routes correctly.

For a detailed example, see the vignette("analyse").

Adapted from github.com/Bondify/GTFS_in_R.

Value

sf data.frame. Hourly route frequencies, with the following columns (the first three are only present if overline=FALSE):

route_id The route_id attribute from routes.txt file.

route_short_name The route_short_name attribute from routes.txt file.

shape_id The shape_id attribute from shapes.txt file.

direction_id The direction_id attribute from trips.txt file (if attribute present in GTFS feed).

hour The hour for which the frequency applies (24 hour format).

frequency The number of services for the route that depart from the first stop for the corresponding 60 minutes period.

geometry The route shape.

See Also

`GTFShift::calendar_nextBusinessWednesday()`

`GTFShift::osm_shapes_to_routes()`

`stplanr::overline2()`

Examples

```
# Subset GTFS for one route only, for demo purposes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)
gtfs <- GTFShift::filter_by_route_name(gtfs, c("1", "2", "3", "4"))

# Get frequency
frequency_analysis <- GTFShift::get_route_frequency_hourly(
  gtfs,
  date = gtfs$calendar$start_date[1]
)

head(frequency_analysis |> sf::st_drop_geometry())
```

---

get_stop_frequency_hourly

Get aggregated frequency per hour for each bus stop

Description

For each stop, returns the number of departures aggregated per hour.

Usage

```
get_stop_frequency_hourly(
  gtfs,
  date = GTFShift::calendar_nextBusinessWednesday()
)
```

Arguments

`gtfs`	tidygtfs. GTFS feed.
`date`	Date (Default `GTFShift::calendar_nextBusinessWednesday()`). Reference date to consider when analyzing the GTFS file.

Details

This method analyses the GTFS feed for a representative day, generating for each stop the number of services aggregated per hour. For a detailed example, see the `vignette("analyse")`.

Value

sf data.frame. Hourly stop frequencies, with the following columns:

stop_id The `stop_id` attribute from `stops.txt` file.

hour The hour for which the frequency applies (24 hour format).

frequency The number of services provided at the stop for the corresponding 60 minutes period.

geometry The stop coordinates.

See Also

`GTFShift::calendar_nextBusinessWednesday()`

Examples

```
# Subset GTFS for one route only, for demo purposes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)
gtfs <- GTFShift::filter_by_route_name(gtfs, c("1", "2", "3", "4"))

# Get frequency
frequency_analysis <- GTFShift::get_stop_frequency_hourly(
  gtfs,
  date = gtfs$calendar$start_date[1]
)

head(frequency_analysis)
```

---

get_way_frequency_hourly

Get aggregated frequency per hour for each OSM way

Description

For each OSM way with GTFS service, returns the number of departures aggregated per hour and direction.

Usage

```
get_way_frequency_hourly(
  gtfs,
  q,
  date = GTFShift::calendar_nextBusinessWednesday(),
  keep_osm_attributes = FALSE,
  osm_file = NULL
)
```

Arguments

gtfs	tidygtfs. GTFS feed.
q	osmdata::opq. Overpass query for transit network, to obtain OSM route ways, using `GTFShift::osm_shapes_to_routes()`.
date	Date (Default `GTFShift::calendar_nextBusinessWednesday()`). Reference date to consider when analyzing the GTFS file.
keep_osm_attributes	Boolean (Default FALSE). Whether to keep all OSM way attributes in the output `sf` object.
osm_file	character (Optional). Location of OSM extract file with `osm.pbf` format. Refer to `osmextract::oe_download()` for more details. If not provided OSM Overpass API is called through `osmdata::osmdata_sf()`.

Details

This method analyses the GTFS feed for a representative day, finding for each route the corresponding OSM ways using `GTFShift::osm_shapes_to_routes()` (routes not on OSM are ignored), aggregating the number of services per hour and direction for each.

For a detailed example, see the `vignette("analyse")`.

Value

sf data.frame. Hourly way frequencies, with the following columns:

way_osm_id The `osm_id` attribute from OSM way.

hour The hour for which the frequency applies (24 hour format).

frequency The number of services for the route that depart from the first stop for the corresponding 60 minutes period.

routes The list of route_ids that use the way.

shapes The list of shape_ids that use the way.

geometry The route shape.

(if `keep_osm_attributes = TRUE`**)** All OSM way attributes.

See Also

`GTFShift::calendar_nextBusinessWednesday()`

`GTFShift::osm_shapes_to_routes()`

Examples

```
# Subset GTFS for one route only, for demo purposes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)
gtfs <- GTFShift::filter_by_route_name(gtfs, c("1", "2", "3", "4"))

# Build query and prepare osm extract (possible to use API as alternative)
q <- osmdata::opq(bbox = sf::st_bbox(tidytransit::shapes_as_sf(gtfs$shapes))) |>
  osmdata::add_osm_feature(key = "route", value = "bus") |>
 osmdata::add_osm_feature(key = "operator", value = "Transportes Colectivos do Barreiro")
osm_file <- system.file("extdata/samples", "osmextract_tcb_network.pbf", package = "GTFShift")

# Get frequency
frequency_analysis <- GTFShift::get_way_frequency_hourly(
  gtfs, q,
  date = gtfs$calendar$start_date[1],
  osm_file = osm_file
)

head(frequency_analysis |> sf::st_drop_geometry())
```

`load_feed` *Read GTFS feed, fixing integrity errors*

Description

Read GTFS feed, fixing integrity errors

Usage

```
load_feed(
  path,
  store_path = NA,
  create_transfers = FALSE,
  transfer_distance = 300,
  transfer_time = 120,
  transfer_street_routing = FALSE,
  headers = NULL
)
```

Arguments

`path`	String. The location of the GTFS zip file. Either local or URL.
`store_path`	String (Optional). If provided, GTFS feed zip is stored at location. The file is overwritten if it already exists.

`create_transfers`
: Boolean (Default FALSE). When true, generates `transfers.txt`, aggregating close stops.

`transfer_distance`
: Integer (Default 300). Upper straight-line distance limit in meters for transfers.

`transfer_time`
: Integer (Default 120). Minimum time in seconds for transfers; all values below this will be replaced with this value, particularly all those defining in-place transfers where stop longitudes and latitudes remain identical.

`transfer_street_routing`
: Boolean (Default FALSE). If TRUE, transfer times are calculated by routing throughout the underlying street network (downloaded automatically).

`headers`
: Named list or character vector (Optional). Custom HTTP headers for credentials when accessing the GTFS zip file URL.

Details

In addition to loading the GTFS feed, this method validates its integrity and applies the proper corrections if it does not comply with the following validations:

- `stop_times.txt` with empty `arrival_time` or `departure_time`, filtering rows that do not comply.
- Feeds with missing `shapes.txt` file, generating it using `GTFShift::create_shapes_from_stops()`.

When generating transfers, those already existing in each GTFS file are kept, extended with new ones computed based on the stops network of the final aggregated version. This computation is executed with `gtfsrouter::gtfs_transfer_table()`, with the parameters `d_limit=transfer_distance`, `min_transfer_time=transfer_time` and `network_times=transfer_street_routing`. The other parameters are applied the library default values.

Value

tidygtfs. The loaded GTFS feed.

See Also

`GTFShift::create_shapes_from_stops()`

`tidytransit::read_gtfs()`

`gtfsrouter::gtfs_transfer_table()`

Examples

```
# Simple call
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)

summary(gtfs)
```

```
# Simple call with missing shapes (triggering shapes creation because missing on GTFS file)
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_ttsl_sample_no_shapes.zip", package = "GTFShift")
)

summary(gtfs)


# With some parameters to build transfers and store to given location
store_path <- withr::local_tempfile(fileext = ".zip")

gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift"), create_transfers = TRUE, store_path
)

head(gtfs$transfers)

file.exists(store_path)
```

`multiline_to_sorted_linestring` *Convert a MULTILINESTRING to a sorted LINESTRING*

Description

Convert a MULTILINESTRING to a sorted LINESTRING

Usage

```
multiline_to_sorted_linestring(
  multilinestring,
  points = NULL,
  metric_crs = 3857
)
```

Arguments

`multilinestring`	sf object with MULTILINESTRING geometry
`points`	(Optional) collection of sorted point geometries used to guide ordering. If provided, the first point defines the initial segment and the second point (when available) is used as tie-break guidance for its orientation. Remaining points are used as iterative tie-break guidance.
`metric_crs`	Integer or character (Default 3857). Projected CRS used to compute distances and lengths during sorting.

Details

The function takes a MULTILINESTRING object and converts it to a LINESTRING object by sorting the linestrings and combining them in the correct order.

The algorithm is formulated as follows: Let $\mathcal{L} = \{L_1, \ldots, L_n\}$ be the set of individual LINESTRING components. Each component L_i is characterized by its start point $S(L_i)$ and end point $E(L_i)$.

1. Initialization

If guiding points are provided, let $\text{start\_point} = P_1$ be the first point and P_2 the second point (if available). The initial segment is chosen as

$$L^{(1)} = \underset{L \in \mathcal{L}}{\operatorname{argmin}}\, d(\text{start\_point}, L).$$

where $d(\cdot)$ is the Euclidean distance. If no points are provided, $L^{(1)} = L_1$ (assuming the input MULTILINESTRING is ordered).

Additionally, the orientation of $L^{(1)}$ is determined by comparing the distances from its edges to the remaining segments in $\mathcal{L} \setminus \{L^{(1)}\}$. The edge that is closest to any remaining segment is designated as the end of $L^{(1)}$.

If both edges are equidistant to the remaining segments, the orientation is determined by the proximity to P_2 (if available) or by orienting away from P_1.

2. Iterative Step

At iteration k, with current segment endpoint $e^{(k)} = E(L^{(k)})$, define for each remaining segment $L \in \mathcal{R}^{(k)}$:

$$d_s(L) = d\Big(e^{(k)}, S(L)\Big), \qquad d_e(L) = d\Big(e^{(k)}, E(L)\Big).$$

Segments with geometry equal to $L^{(k)}$ are excluded. Candidate segments minimize endpoint proximity:

$$\mathcal{C}^{(k)} = \Big\{ L \in \mathcal{R}^{(k)} : \min\big(d_s(L), d_e(L)\big) = m^{(k)} \Big\}, \quad m^{(k)} = \min_{J \in \mathcal{R}^{(k)}} \min\big(d_s(J), d_e(J)\big).$$

Ties are broken as follows:

(i) if next unvisited point Q exists, choose closest candidate, minimizing $d(Q, L)$ over $L \in \mathcal{C}^{(k)}$;

(ii) if still tied, choose candidate closest to the current endpoint $e^{(k)}$, minimizing $d\Big(e^{(k)}, S(L)\Big)$ over $L \in \mathcal{C}^{(k)}$.

3. Verification and Assembly

Let L^* be the selected candidate. If

$$d\Big(L^{(k)}, L^*\Big) > \text{len}\Big(L^{(k)}\Big) + \text{len}(L^*),$$

then L^* is removed from the remaining set and the loop restarts. Otherwise, L^* is oriented to connect from $e^{(k)}$ and appended to the ordered sequence. When points are provided, consecutive unvisited points Q are marked visited when

$$d\Big(L^{(k+1)}, Q\Big) \leq \min_{J \in \mathcal{R}^{(k+1)}} d(J, Q).$$

The ordered segments are concatenated into a single `LINESTRING` and transformed back to the original CRS of `multilinestring`.

Value

sfc. LINESTRING geometry object.

Examples

```
# Get OSM route geometries (MULTILINESTRING)
osm_routes <- sf::st_read(
  system.file("extdata/samples", "osm_routes_tcb.gpkg", package = "GTFShift"),
  quiet = TRUE
) |> dplyr::sample_n(1)

head(osm_routes)

# Convert geometry to LINESTRING
osm_routes <- osm_routes |>  dplyr::mutate(
  geom = GTFShift::multiline_to_sorted_linestring(geom, metric_crs = 3763)
)

head(osm_routes)
```

---

network_overline *Aggregate lines based on overlap with target network*

Description

Aggregate lines based on overlap with target network

Usage

```
network_overline(
  target_network,
  lines,
  attr,
  target_network_split = 100,
  fun = sum,
  join_dist = 10,
  metric_crs = 3857
)
```

Arguments

target_network	sf. A spatial object representing the target network.
lines	sf. A spatial object representing the lines to aggregate.
attr	String. The attribute to aggregate the lines by.
target_network_split	Integer (Default 100). If not NA, network is split in segments of defined meters.

fun	Method (Default base::sum). Function to summarise the attributes by.
join_dist	Integer (Default 10). Meters to consider when joining routes and network segments.
metric_crs	Integer or character (Default 3857). Projected CRS used to compute segment lengths and join distances in meters.

Details

This method allows for the lines aggregation. Given a target network, it identifies (using stplanr::rnet_join()) the segments corresponding to each line and uses them to aggregate the attribute defined in the parameters.

It provides an alternative to GTFShift::get_route_frequency_hourly() with the attribute overline=TRUE, which creates an aggregated network based on the lines overlap. Instead, GTFShift::network_overline() finds, for each network segment, the overlapping lines and aggregates their attr values, using fun.

Value

sf. Spatial network object extended with aggregated values.

See Also

stplanr::rnet_join()

Examples

```
# Subset GTFS for one route only, for demo purposes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)
gtfs <- GTFShift::filter_by_route_name(gtfs, c("4", "1"))

# Load OSM network to serve as target network
target_network = sf::st_read(
  system.file("extdata/samples", "osm_ways_tcb.gpkg", package = "GTFShift"),
  quiet = TRUE
)

head(target_network)

# Get route frequency (and geometry)
frequency_analysis <- GTFShift::get_route_frequency_hourly(
  gtfs,
  date = gtfs$calendar$start_date[1]
) |>
dplyr::group_by(shape_id) |>
dplyr::summarize(frequency = max(frequency))

head(frequency_analysis)

# Aggregate frequencies based on geometry overlap using GTFShift::network_overline
suppressWarnings({
```

```
  overline <- GTFShift::network_overline(
    target_network = target_network,
    lines = frequency_analysis,
    attr = "frequency",
   metric_crs = 3763 # Make sure to addapt to the projection that better suits your location
  )
})

head(overline |> st_drop_geometry())
```

`osm_bus_lanes` *Export designated bus lanes from OpenStreetMaps*

Description

Export designated bus lanes from OpenStreetMaps

Usage

```
osm_bus_lanes(bbox, osm_file = NULL)
```

Arguments

bbox	bbox. Area from which to export bus lanes.
osm_file	character (Optional). Location of OSM extract file with `osm.pbf` format. Refer to `osmextract::oe_download()` for more details. If not provided OSM Overpass API is called through `osmdata::osmdata_sf()`.

Details

Exports roads tagged as designated bus lanes on OpenStreetMaps for given area.

Value

sf data.frame. OSM bus lanes.

Examples

```
# Create bbox for Lisbon
bbox <- sf::st_as_sfc(sf::st_bbox(c(
  xmin = -9.229836, ymin = 38.691399,
  xmax = -9.087387, ymax = 38.796760
), crs = 4326))

# Use sample osmextract for Lisbon highways
osm_file <- system.file(
  "extdata/samples", "osmextract_lisbon_highways_sample.pbf", package = "GTFShift"
```

```
)

# Export bus lanes
bus_lanes <- GTFShift::osm_bus_lanes(bbox, osm_file = osm_file)

names(bus_lanes)

head(bus_lanes |> dplyr::select(`osm:id`, name))
```

`osm_centerlines` *Get centerlines for OSM road network*

Description

Get centerlines for OSM road network

Usage

```
osm_centerlines(
  bbox = NULL,
  place = NULL,
  osm_file = NULL,
  use_buildings = TRUE,
  venv = NA
)
```

Arguments

`bbox`	bbox (Optional, if place provided). Area from which to export bus lanes.
`place`	String (Optional, if bbox provided). Place from which to export bus lanes.
`osm_file`	String (Optional). Path to a local OpenStreetMap PBF file ('.pbf').
`use_buildings`	Boolean (Default TRUE). Uses buildings from OSM as exclusion_mask for neatnet.
`venv`	String (Default creates a new one). Python environment where neatnet will run.

Details

Exports road network from OpenStreetMaps for given area and uses Python neatnet package to compute its centerlines.

One of `bbox`, `place`, or `osm_file` must be provided.

Parameter `use_buildings` exports building footprints from OSM for better results on the network simplification process.

This method was adapted from uscuni.org/neatnet by Miguel Relvas Pires in the scope of his master's thesis. The full code (Python) of his work is openly available at GitHub.

Value

sf data.frame. OSM centerlines.

Author(s)

Miguel Relvas Pires

Examples

```
## Not run:
# Get sample OSM extract
osm_file <- system.file("extdata/samples", "relation_6384187.pbf", package = "GTFShift")

network <- GTFShift::osm_centerlines(
  place = "Arroios, Lisboa, Portugal",
  osm_file = osm_file
)

head(network)

table(network$X_status)

## End(Not run)
```

---

osm_shapes_match_routes

Get OSM routes that match shapes, based on geometrical match

Description

Get OSM routes that match shapes, based on geometrical match

Usage

```
osm_shapes_match_routes(
  gtfs,
  q,
  geometry = TRUE,
  gtfs_match = "route_short_name",
  osm_match = "ref",
  gtfs_osm_match_exact = TRUE,
  log_file = NA,
  osm_file = NULL,
  num_cores = 1,
  osm_stop_order_relaxed = FALSE,
  osm_route_type = "bus",
  metric_crs = 3857
)
```

Arguments

gtfs	tidygtfs. GTFS feed.
q	osmdata::opq. Overpass query for transit network
geometry	Boolean (Default TRUE). If TRUE, returns sf object with geometry, otherwise, a simple data.frame.
gtfs_match	String (Default route_short_name). routes.txt attribute that identifies routes. Accepted values: route_id, route_short_name, route_long_name.
osm_match	String (Default ref). OSM attribute that identifies routes by matching with gtfs_match. Accepted values: ref, name, gtfs:route_id.
gtfs_osm_match_exact	Boolean (Default TRUE). If TRUE, gtfs and route names are matched strictly. Otherwise, partial string match is considered (all words in gtfs_match must be in osm_match, ignoring case).
log_file	String (Optional). If provided, will log warnings to this file, in addition to the console.
osm_file	character (Optional). Location of OSM extract file with `osm.pbf` format. Refer to `osmextract::oe_download()` for more details. If not provided OSM Overpass API is called through `osmdata::osmdata_sf()`.
num_cores	Integer (Default 1). Number of cores to use for parallel computation. Only supported on Unix-like systems (Linux, macOS).
osm_stop_order_relaxed	Boolean (Default FALSE). If TRUE, OSM routes with entry/exit stops not respecting the right order will still be matched (this may indicate OSM data integrity problems). If FALSE, these routes will be ignored.
osm_route_type	character (Default "bus"). OSM route type. Used to query OSM network (e.g., 'bus', 'train').
metric_crs	Integer or character (Default 3857). Projected CRS used to compute shapes and routes lengths and stop-to-stop distances.

Details

For each route, matches its trips' shapes with OSM route relations.

The matching algorithm is formulated as follows: Let R be a GTFS route identifier.

1. Filtering and Base Data Selection: Let $\mathcal{O}_R = \{O_1, \ldots, O_m\}$ be the set of candidate OSM route relations matching the identifier R (based on `gtfs_match` and `osm_match`). If $\mathcal{O}_R$ is empty, route R is skipped. Unless `osm_stop_order_relaxed = TRUE`, any relation in $\mathcal{O}_R$ with entry/exit stops not in the correct order is discarded. We also retrieve the set of GTFS shapes associated with route R, denoted as $\mathcal{S}_R = \{S_1, \ldots, S_n\}$.

2. Feature Extraction: For each GTFS shape $S_i \in \mathcal{S}_R$:

- Extract the start and end coordinates of its trips' first and last stops: $\text{init}_{GTFS,i}$ and $\text{fin}_{GTFS,i}$.
- Compute the shape's total length $L_{GTFS,i}$ and the number of stop times $N_{stops,i}$.

For each candidate OSM route relation $O_j \in \mathcal{O}_R$:

- Extract the coordinates of the first and last stops/platforms: $\text{init}_{OSM,j}$ and $\text{fin}_{OSM,j}$.
- Compute the relation's geometry length $L_{OSM,j}$ and the number of stop/platform nodes $N_{stops,j}$.

3. Closeness Metric Evaluation: For each GTFS shape S_i, we calculate the closeness metric $C(i,j)$ for all candidate OSM routes $O_j \in \mathcal{O}_R$:

$$C(i,j) = d(\text{init}_{GTFS,i}, \text{init}_{OSM,j}) + d(\text{fin}_{GTFS,i}, \text{fin}_{OSM,j}) + |L_{GTFS,i} - L_{OSM,j}| + \frac{L_{GTFS,i}}{N_{stops,i}} \cdot |N_{stops,i} - N_{stops,j}|$$

where:

- $d(\cdot)$ is the Euclidean distance.
- The term $\frac{L_{GTFS,i}}{N_{stops,i}}$ represents the average distance between stops on the GTFS shape, serving as a scale factor for the difference in the number of stops.

Shape S_i is associated with the OSM route O_{j^*} that minimizes the closeness metric:

$$j^* = \text{argmin}_j \, C(i,j)$$

4. Conflict Resolution: If multiple GTFS shapes are associated with the same OSM route O_j, only the shape S_i that minimizes the closeness metric is retained. The other conflicting shapes are ignored and a warning is triggered.

Be aware that the result might ignore some GTFS routes, in the following cases:

- If there is no OSM route relation that matches the GTFS route identifier;
- If, for a GTFS route, there is any OSM route relation that has entry/exit stops not respecting the right order (unless `osm_stop_order_relaxed` is set to TRUE);
- If, for the same route, distinct shapes are associated to the same OSM route. In that case, only the shape that minimizes the closeness metric is retained.

If any of these errors occurs, warnings will be thrown at end of the method execution, and those GTFS route will be ignored in the results.

Nevertheless, provided there are enough OSM routes, all the GTFS shapes for each route will necessarily be associated with an OSM one. This might generate wrong results if the topology of routes on OSM does not match the GTFS shapes for that route. Refer to `distance_diff`, `points_diff` and `stops_diff` on the results table to validate the results and identify misassociations.

Value

data.frame. Matched routes (`sf` if `geometry=TRUE`) with the following columns:

route_id The `route_id` attribute from `routes.txt` file.

shape_id The `shape_id` attribute from `shapes.txt` file.

osm_id The `osm_id` attribute from OSM route relation.

distance_diff The difference, in meters, between GTFS shape and OSM route lengths.

points_diff The sum of the difference, in meters, between GTFS shape and OSM route start and end points.

stops_diff The difference between GTFS and OSM routes number of stops.

route_short_name The route_short_name attribute from routes.txt file.

route_long_name The route_long_name attribute from routes.txt file.

osm_ref The ref attribute from OSM route relation.

osm_name The name attribute from OSM route relation.

geometry The geometrical data for the OSM route relation.

Examples

```
# Subset GTFS for one route only, for demo purposes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip",
  package = "GTFShift"
))
gtfs <- GTFShift::filter_by_route_name(gtfs, c("1", "2", "3", "4"))

# Build query and prepare osm extract (possible to use API as alternative)
q <- osmdata::opq(bbox = sf::st_bbox(tidytransit::shapes_as_sf(gtfs$shapes))) |>
  osmdata::add_osm_feature(key = "route", value = "bus") |>
  osmdata::add_osm_feature(key = "operator", value = "Transportes Colectivos do Barreiro")
osm_file <- system.file("extdata/samples", "osmextract_tcb_network.pbf", package = "GTFShift")

# Get OSM route geometries based on geometrical match
shapes_osm_routes <- GTFShift::osm_shapes_match_routes(
  gtfs, q,
  osm_file = osm_file,
  metric_crs = 3763, # Make sure to addapt to the projection that better suits your location
)

head(shapes_osm_routes |> dplyr::select(shape_id, osm_id, distance_diff, points_diff, stops_diff))
```

---

osm_shapes_to_routes *Get OSM routes geometry considering gtfs:shape_id match*

Description

Get OSM routes geometry considering gtfs:shape_id match

Usage

```
osm_shapes_to_routes(
  gtfs,
  q,
  ways = FALSE,
  ways_tags = c("lanes", "psv", "bus", "way", "parking", "name"),
  osm_file = NULL,
  osm_route_type = "bus"
)
```

Arguments

gtfs	tidygtfs. GTFS feed.
q	osmdata::opq. Overpass query for transit network.
ways	boolean (Default False). If true, relation is disaggregated in ways.
ways_tags	character vector (Default c("lanes", "psv", "bus", "way", "parking", "name")). List of OSM way tags to extract when ways parameter is set to true. Match is done using tidyselect::contains().
osm_file	character (Optional). Location of OSM extract file with osm.pbf format. Refer to osmextract::oe_download() for more details. If not provided OSM Overpass API is called through osmdata::osmdata_sf().
osm_route_type	character (Default "bus"). OSM route type. Used to query OSM network (e.g., 'bus', 'train').

Details

For each route, matches its trips' shapes with OSM route relations, considering the OSM gtfs:shape_id attribute.

Value

sf data.frame. Matched shape to route geometries with the following columns:

shape_id The shape_id attribute from shapes.txt file.

osm_id The osm_id attribute from OSM route relation.

way_osm_id The osm_id attribute from OSM way (if ways parameter is set to true).

***** Any column that matches ways_tags parameter.

geometry The geometrical data for the OSM route relation.

Shapes that do not have a match on OSM are ignored. If that occurs, a warning is displayed during the method execution, informing about the missing geometries.

Examples

```
# Subset GTFS for one route only, for demo purposes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)
gtfs <- GTFShift::filter_by_route_name(gtfs, c("1", "2", "3", "4"))

# Build query and prepare osm extract (possible to use API as alternative)
q <- osmdata::opq(bbox = sf::st_bbox(tidytransit::shapes_as_sf(gtfs$shapes))) |>
  osmdata::add_osm_feature(key = "route", value = "bus") |>
 osmdata::add_osm_feature(key = "operator", value = "Transportes Colectivos do Barreiro")
osm_file <- system.file("extdata/samples", "osmextract_tcb_network.pbf", package = "GTFShift")

# Get OSM route geometries based on gtfs:shape_id match
shapes_osm_routes <- GTFShift::osm_shapes_to_routes(
  gtfs, q,
```

```
  osm_file = osm_file
)

head(shapes_osm_routes |> dplyr::select(shape_id, osm_id))

nrow(shapes_osm_routes)

# Get OSM ways instead
shapes_osm_ways <- GTFShift::osm_shapes_to_routes(
  gtfs, q,
  osm_file = osm_file,
  ways = TRUE
)

head(shapes_osm_ways |> dplyr::select(way_osm_id, shape_id, osm_id))

nrow(shapes_osm_ways)
```

`prioritise_lanes` *Prioritise road network lanes for bus lane implementation*

Description

For each OSM way with GTFS service, aggregates its characteristics to assist in the bus lane implementation prioritisation

Usage

```
prioritise_lanes(
  gtfs,
  q,
  date = GTFShift::calendar_nextBusinessWednesday(),
  keep_osm_attributes = FALSE,
  osm_file = NULL
)
```

Arguments

`gtfs`	tidygtfs. GTFS feed.
`q`	osmdata::opq. Overpass query for transit network, to obtain OSM route ways, using `GTFShift::osm_shapes_to_routes()`.
`date`	Date (Default `GTFShift::calendar_nextBusinessWednesday()`). Reference date to consider when analyzing the GTFS file.
`keep_osm_attributes`	Boolean (Default FALSE). Whether to keep all OSM way attributes in the output `sf` object.

osm_file character (Optional). Location of OSM extract file with osm.pbf format. Refer to osmextract::oe_download() for more details. If not provided OSM Overpass API is called through osmdata::osmdata_sf().

Details

This method analyses the GTFS feed for a representative day, returning a data.frame with the road segments where transit routes run and for each, a set of parameters that can be used to prioritise bus lane implementations.

Its functionality is a bundle that encapsulates the logic of several methods from the package, including GTFShift::get_way_frequency_hourly() and GTFShift::osm_bus_lanes(), that can be used separately if needed.

Mind that this method uses GTFShift::get_way_frequency_hourly() to match routes with OSM ways, which requires that the OSM relation mapping is well defined for the transit routes. Routes that do not have an OSM match are ignored.

Value

sf data.frame. Prioritised lanes with the following columns:

way_osm_id The osm_id attribute from OSM way.

hour The hour for which the frequency applies (24 hour format).

frequency The number of services for the route that depart from the first stop for the corresponding 60 minutes period.

is_bus_lane Whether the way has a bus lane.

n_lanes_parking The number of parking lanes.

n_lanes_circulation The number of circulation lanes.

n_directions The number of travel directions.

n_lanes_circulation_direction The number of circulation lanes per direction.

routes The list of route_id that use the way.

shapes The list of shape_id that use the way.

geometry The route shape.

(if keep_osm_attributes = TRUE**)** All OSM way attributes.

Examples

```
# Subset GTFS for one route only, for demo purposes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)
gtfs <- GTFShift::filter_by_route_name(gtfs, c("4"))

# Build query and prepare osm extract (possible to use API as alternative)
q <- osmdata::opq(bbox = sf::st_bbox(tidytransit::shapes_as_sf(gtfs$shapes))) |>
  osmdata::add_osm_feature(key = "route", value = "bus") |>
 osmdata::add_osm_feature(key = "operator", value = "Transportes Colectivos do Barreiro")
```

```
osm_file <- system.file("extdata/samples", "osmextract_tcb_network.pbf", package = "GTFShift")

lane_prioritisation <- GTFShift::prioritise_lanes(
  gtfs, q,
  osm_file = osm_file,
  date = gtfs$calendar$start_date[1]
)

head(
  lane_prioritisation |>
  dplyr::select(way_osm_id, hour, frequency, is_bus_lane, n_lanes_circulation, routes)
)
```

`project_points_along_geometry` *Project points onto a linear geometry*

Description

Projects point geometries to the closest location along a single LINESTRING or MULTILINESTRING and estimates each projected point position as cumulative distance from the start of the line.

Usage

```
project_points_along_geometry(
  geometry,
  points,
  geometry_sample_meters = 10,
  metric_crs = 3857
)
```

Arguments

`geometry`	sf or sfc object with exactly one linear geometry (LINESTRING or MULTILINESTRING).
`points`	sf or sfc object with point geometries to be projected.
`geometry_sample_meters`	Numeric (Default 10). Sampling step used to discretize the line when estimating cumulative distance along geometry.
`metric_crs`	Integer or character (Default 3857). Projected CRS used to compute nearest points, line sampling, and cumulative distances.

Details

The function first computes nearest points from each input point to `geometry` with `sf::st_nearest_points()`, keeping the point on the line. Then, it samples the line at regular intervals and assigns cumulative distance by nearest sampled location.

Distances are always computed in `metric_crs` units. The returned projected points are transformed back to the original `geometry` CRS.

Value

data.frame. Input points projected along geometry with four columns:

closest_on_geometry An `sfc_POINT` column with the projected location on the line.

distance_to_closest_on_geometry Numeric distance from each input point to its projected location on the line.

distance_along_geometry Numeric cumulative distance from the line start to the projected location.

distance_along_geometry_reversed Numeric cumulative distance from the line end to the projected location.

If `points` is empty, returns an empty data.frame with the same columns.

Examples

```
# Get sample points from GTFS-RT collection
rt_collect_file <- system.file(
  "extdata/samples", "gtfs_rt_sample_tcb_4_4-CS-TERM.csv", package = "GTFShift"
)
points <- read.csv(rt_collect_file) |>
  sf::st_as_sf(coords = c("longitude", "latitude"), crs = 4326) |> dplyr::sample_n(5)

head(points |> dplyr::select(geometry))

# Get route geometry for points
osm_routes <- sf::st_read(
  system.file("extdata/samples", "osm_routes_tcb.gpkg", package = "GTFShift"),
  quiet = TRUE
) |> dplyr::filter(route_id %in% points$route_id)

head(osm_routes)

# Project points to geometry
points_projected <- GTFShift::project_points_along_geometry(
  geometry = osm_routes,
  points = points,
  metric_crs = 3763 # Make sure to addapt to the projection that better suits your location
)

head(points_projected)
```

query_mobilitydatabase *Query Mobility Database API for GTFS feeds*

Description

Query Mobility Database API for GTFS feeds

Usage

```
query_mobilitydatabase(
  access_token = NA,
  refresh_token = NA,
  bounding_filter_method = "partially_enclosed",
  limit = 10,
  offset = 0,
  country_code = NA,
  subdivision_name = NA,
  municipality = NA,
  bbox = NA,
  is_official = NA
)
```

Arguments

access_token	String (Optional when refresh_token). Access token.
refresh_token	String (Optional when access_token). Refresh token.
bounding_filter_method	String (Default partially_enclosed). Filtering method to use with the dataset_latitudes and dataset_longitudes parameters.
limit	Integer (Default 10). The number of items to be returned.
offset	Integer (Default 0). Offset of the first item to return.
country_code	String (Optional). Filter feeds by their exact country code.
subdivision_name	String (Optional). List only feeds with the specified value. Can be a partial match.
municipality	String (Optional). List only feeds with the specified value. Can be a partial match. Case insensitive.
bbox	bbox (Optional). Area from which to get GTFS feeds. Converted to API dataset_latitudes and dataset_longitudes URL parameters.
is_official	Boolean (Optional). If TRUE, only return official feeds.

Details

This method queries Mobility Database API, allowing to get a list of GTFS feeds documented at this platform. To use it, an access or a refresh token must be provided. It can be obtained for free at Mobility Database website.

For more details on the parameters, refer to https://mobilitydata.github.io/mobility-feed-api/SwaggerUI/index.html#/feeds/getGtfsFeeds.

Some useful columns of the returned data.frame (refer to the API documentation for a full list) are:

provider The name of the GTFS provider.

status Tells if the feed is active, inactive or deprecated.

producer_url The GTFS feed URL. Can be used to download.

Value

data.frame. Query results from Mobility Database.

Examples

```
feeds <- GTFShift::query_mobilitydatabase(
  refresh_token = Sys.getenv("MOBILITY_DATABASE"),
  country_code = "PT",
  is_official = TRUE
)

head(feeds |> dplyr::select(id, provider, producer_url))
```

---

rt_average_speed *Estimate average speed for GTFS-RT trip updates*

Description

Projects each real-time vehicle position to its corresponding trip geometry, computes cumulative distance along the geometry, and derives segment speed between consecutive updates.

Usage

```
rt_average_speed(
  rt_collection,
  trips_geometries,
  rt_collection_trips_geometries_match_col = "trip_id",
  geometry_sample_meters = 10,
  metric_crs = 3857
)
```

Arguments

`rt_collection`	sf data.frame with GTFS-RT updates for multiple trips. Must include at least `trip_id` and `timestamp` columns.
`trips_geometries`	sf data.frame with trip geometries. Geometry must be LINESTRING.
`rt_collection_trips_geometries_match_col`	Character (Default `"trip_id"`). Column name present in both `rt_collection` and `trips_geometries` used to match updates to trip geometry.
`geometry_sample_meters`	Numeric (Default 10). Sampling step used when projecting points along trip geometry and estimating cumulative distance.
`metric_crs`	Integer or character (Default 3857). Projected CRS used to compute distances and speeds.

Details

For each trip (grouped by `trip_id`), let $\{(x_i, t_i)\}_{i=1}^{n}$ denote the ordered sequence of real-time observations, where x_i is the vehicle position and t_i the corresponding timestamp, with $t_1 \leq t_2 \leq \cdots \leq t_n$. Each observation is projected onto the trip geometry using `GTFShift::project_points_along_geometry()`, yielding a projected point $\hat{x}_i$ and two cumulative distances:

$$d_i = \text{distance\_along\_geometry}(\hat{x}_i)$$

$$d_i^{\text{rev}} = \text{distance\_along\_geometry\_reversed}(\hat{x}_i)$$

For each pair of consecutive observations $(i-1, i)$, the elapsed time is computed as

$$\Delta t_i = t_i - t_{i-1}.$$

The distance increment is defined as the minimum of the forward and reversed cumulative-distance differences:

$$\Delta d_i^{\text{fwd}} = |d_i - d_{i-1}|$$

$$\Delta d_i^{\text{rev}} = \left|d_i^{\text{rev}} - d_{i-1}^{\text{rev}}\right|$$

$$\Delta d_i = \min\left(\Delta d_i^{\text{fwd}}, \Delta d_i^{\text{rev}}\right).$$

Trips with fewer than 2 updates are ignored with a warning. The distance increment is defined as the minimum of two alternative cumulative-distance differences:

$$\Delta d_i^{\text{fwd}} = |d_i - d_{i-1}|$$

$$\Delta d_i^{\text{circ}} = \left|d_i - d_{i-1}^{\text{rev}}\right|$$

$$\Delta d_i = \min\left(\Delta d_i^{\text{fwd}}, \Delta d_i^{\text{circ}}\right).$$

The second term is a redundancy designed to avoid overstating movement on circular geometries. In particular, after a vehicle completes a loop, a forward comparison may treat two nearby physical positions as far apart in cumulative distance if the geometry origin has been crossed. Comparing d_i

against d_{i-1}^{rev} provides an auxiliary distance candidate that helps avoid overstating movement in that situation.

Average speed is then estimated by

$$v_i = \frac{\Delta d_i}{\Delta t_i}$$

and reported in kilometers per hour as

$$v_i^{\text{km/h}} = \frac{\Delta d_i}{1000} \cdot \frac{3600}{\Delta t_i}.$$

Trips with fewer than two observations are ignored with a warning because Δt_i and Δd_i are undefined in that case.

Method `GTFShift::multiline_to_sorted_linestring()` can be used to convert MULTILINESTRING geometries to LINESTRING if needed.

Value

sf data.frame. Object based on `rt_collection`, with added columns:

closest_on_shape Projected point on trip geometry.

distance_to_closest_on_geometry Distance from each update point to its projected location on the shape (meters).

distance_along_geometry Cumulative distance along trip geometry (meters).

distance_along_geometry_reversed Cumulative distance from shape end to projected location (meters).

time_since_prev_sec Elapsed time since previous update (seconds).

distance_since_prev_meters Distance increment since previous update (meters).

speed_kmh Estimated speed between consecutive updates (km/h).

See Also

`GTFShift::project_points_along_geometry()`

`GTFShift::multiline_to_sorted_linestring()`

Examples

```
# Get GTFS-RT data collection
rt_collect_file <- system.file(
  "extdata/samples", "gtfs_rt_sample_tcb_4_4-CS-TERM.csv", package = "GTFShift"
)
rt_collection <- read.csv(rt_collect_file) |>
  sf::st_as_sf(coords = c("longitude", "latitude"), crs = 4326) |> dplyr::select(-speed)

head(rt_collection |> dplyr::select(trip_id, timestamp, geometry))

nrow(rt_collection)

# Get route geometry for data collected
```

```
osm_routes <- sf::st_read(
  system.file("extdata/samples", "osm_routes_tcb.gpkg", package = "GTFShift"),
  quiet = TRUE
) |>
  dplyr::filter(route_id %in% rt_collection$route_id) |>
  dplyr::mutate(geom = GTFShift::multiline_to_sorted_linestring(geom, metric_crs = 3763))

head(osm_routes)

# Compute average speed (aggregated at route level) based on cumulative distance along the geometry
speed <- GTFShift::rt_average_speed(
  rt_collection = rt_collection,
  trips_geometries = osm_routes,
  rt_collection_trips_geometries_match_col = "route_id",
  metric_crs = 3763 # Make sure to addapt to the projection that better suits your location
)

head(speed |>
  dplyr::filter(!is.na(speed_kmh)) |>
  dplyr::select(
    trip_id, timestamp, speed_kmh,
    distance_along_geometry, distance_to_closest_on_geometry
  )
)

nrow(speed)
```

`rt_collect_json` *Collect GTFS-RT data from a JSON feed at regular intervals*

Description

Collect GTFS-RT data from a JSON feed at regular intervals

Usage

```
rt_collect_json(
  gtfs_rt_url,
  destination_file,
  header_key = "header",
  entity_key = "entity",
  fields_collect = c("id", "vehicle.trip.trip_id", "vehicle.position.latitude",
   "vehicle.position.longitude", "vehicle.position.speed", "vehicle.timestamp",
   "vehicle.current_status", "vehicle.current_stop_sequence", "vehicle.stop_id"),
  scrape_interval = 60,
  log_file = NA,
  headers = NULL
)
```

Arguments

gtfs_rt_url	String. URL of the GTFS-RT feed in JSON format.
destination_file	String. File to save the downloaded GTFS-RT data. Content is appended in each iteration.
header_key	String (Default "header"). Key in the JSON corresponding to the feed header. Set to NA if not present.
entity_key	String (Default "entity"). Key in the JSON corresponding to the feed entities. Set to NA if response is a flat list. Use "." for nested keys.
fields_collect	Character vector. Fields to extract from each entity in the feed. Use "." for nested keys.
scrape_interval	Integer (Default 60). Interval in seconds between each download. Negative to run only once.
log_file	String (Optional). Path to a log file to save download logs.
headers	Named list or character vector (Optional). Custom HTTP headers for credentials when accessing the GTFS-RT feed URL.

Details

Downloads GTFS-RT data from the specified URL at regular intervals and saves them to the destination file.

This function will run indefinitely until manually stopped (CTRL + C).

Value

String. The location of the file where data was collected.

Examples

```
# Create file
destination_file <- withr::local_tempfile(fileext = ".csv")

# Collect data
GTFShift::rt_collect_json(
 gtfs_rt_url = "https://go.tmlmobilidade.pt/hub/api/v1/realtime/vehicles/positions/gtfs",
  entity_key = "data.entity",
  destination_file = destination_file,
  scrape_interval = -1 # Negative to run only once
)

# Read data
collection <- read.csv(destination_file)

names(collection)

head(
  collection |>
```

```
  dplyr::select("vehicle.trip.trip_id", "vehicle.position.latitude", "vehicle.position.longitude")
)
```

`rt_collect_protobuf` *Collect GTFS-RT data from a Protocol Buffers feed at regular intervals*

Description

Collect GTFS-RT data from a Protocol Buffers feed at regular intervals

Usage

```
rt_collect_protobuf(
  gtfs_rt_url,
  destination_file,
  fields_collect = c("id", "vehicle.trip.trip_id", "vehicle.position.latitude",
   "vehicle.position.longitude", "vehicle.position.speed", "vehicle.timestamp",
   "vehicle.current_status", "vehicle.current_stop_sequence", "vehicle.stop_id"),
  scrape_interval = 60,
  log_file = NA,
  headers = NULL
)
```

Arguments

`gtfs_rt_url`	String. URL of the Protocol Buffers GTFS-RT feed.
`destination_file`	String. File to save the downloaded GTFS-RT data. Content is appended in each iteration.
`fields_collect`	Character vector. Fields to extract from each entity in the feed.
`scrape_interval`	Integer (Default 60). Interval in seconds between each download. Negative to run only once.
`log_file`	String (Optional). Path to a log file to save download logs.
`headers`	Named list or character vector (Optional). Custom HTTP headers for credentials when accessing the GTFS-RT feed URL.

Details

Downloads GTFS-RT data from the specified URL at regular intervals and saves them to the destination file.

This function will run indefinitely until manually stopped (CTRL + C).

Value

String. The location of the file where data was collected.

Examples

```
## Not run:
# Create file
destination_file <- withr::local_tempfile(fileext = ".csv")

# Collect data
GTFShift::rt_collect_protobuf(
 gtfs_rt_url = "https://go.tmlmobilidade.pt/hub/api/v1/realtime/vehicles/positions/gtfs.pb",
  destination_file = destination_file,
  scrape_interval = -1 # Negative to run only once
)

# Read data
collection <- read.csv(destination_file)

names(collection)

head(
  collection |>
   dplyr::select("vehicle.trip.trip_id", "vehicle.position.latitude", "vehicle.position.longitude")
)

## End(Not run)
```

---

rt_extend_prioritisation

Extend prioritisation with GTFS-RT based speed metrics

Description

This function extends lane segment indicators for prioritisation with speed metrics produced with GTFS-RT data.

Usage

```
rt_extend_prioritisation(
  lane_prioritisation,
  rt_collection,
  rt_current_status = c("IN_TRANSIT_TO"),
  lane_buffer = 15,
  metric_crs = 3857
)
```

Arguments

lane_prioritisation	sf data.frame. Result of GTFShift::prioritise_lanes()
rt_collection	sf data.frame. GTFS-RT data collection. Must include speed column.
rt_current_status	Character vector (Default c("IN_TRANSIT_TO")). If the current_status column is present in the rt_collection data, only points with current_status in this vector are considered.
lane_buffer	numeric (Default 15). Buffer distance (in meters) to create around lane segments to capture nearby GTFS-RT points.
metric_crs	Integer or character (Default 3857). Projected CRS used to apply lane buffer distances in meters.

Details

Extends the lane_prioritisation data with speed metrics calculated from the GTFS-RT data points that fall within a buffer around each lane segment.

If GTFS-RT data does not provide speed information, it can be inferred from the progression of position updates through time using GTFShift::rt_average_speed().

Refer to GTFShift::rt_collect_json() or GTFShift::rt_collect_protobuf() for details on GTFS-RT data collection.

Value

sf data.frame. Extended lane prioritisation with the following columns:

speed_avg The average speed of the vehicles on the way.

speed_median The median speed of the vehicles on the way.

speed_p25 The 25th percentile speed of the vehicles on the way.

speed_p75 The 75th percentile speed of the vehicles on the way.

speed_count The number of speed observations on the way.

Examples

```
# Subset GTFS for one route only, for demo purposes
gtfs <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip",
  package = "GTFShift"
))
gtfs <- GTFShift::filter_by_route_name(gtfs, c("4"))

# Build query and prepare osm extract (possible to use API as alternative)
q <- osmdata::opq(bbox = sf::st_bbox(tidytransit::shapes_as_sf(gtfs$shapes))) |>
  osmdata::add_osm_feature(key = "route", value = "bus") |>
 osmdata::add_osm_feature(key = "operator", value = "Transportes Colectivos do Barreiro")
osm_file <- system.file("extdata/samples", "osmextract_tcb_network.pbf", package = "GTFShift")
```

```
# Prioritise lanes
lane_prioritisation <- GTFShift::prioritise_lanes(
  gtfs, q,
  osm_file = osm_file,
  date = gtfs$calendar$start_date[1]
)

# Extend with GTFS-RT data collection
rt_collect_file <- system.file(
  "extdata/samples", "gtfs_rt_sample_tcb_4_4-CS-TERM.csv",
  package = "GTFShift"
)
rt_collection <- read.csv(rt_collect_file) |>
  sf::st_as_sf(coords = c("longitude", "latitude"), crs = 4326)

lane_prioritisation_extended <- GTFShift::rt_extend_prioritisation(
  lane_prioritisation = lane_prioritisation,
  rt_collection = rt_collection,
  metric_crs = 3763 # Make sure to addapt to the projection that better suits your location
)

head(
  lane_prioritisation_extended |>
    sf::st_drop_geometry() |>
    dplyr::filter(!is.na(speed_count)) |>
    dplyr::select(way_osm_id, speed_avg, speed_count)
)
```

`unify` *Merge multiple GTFS into a single aggregated file*

Description

Merge multiple GTFS into a single aggregated file

Usage

```
unify(
  ...,
  prefix = FALSE,
  store_path = NA,
  create_transfers = FALSE,
  transfer_distance = 300,
  transfer_time = 120,
  transfer_street_routing = FALSE
)
```

Arguments

`...`	tidygtfs[]. List of GTFS feeds.
`prefix`	Boolean (Default FALSE). If TRUE, prefixes all tables with the agency_id, to avoid conflicts in case of same IDs in different feeds.
`store_path`	String (Optional). If provided, aggregated feed zip is stored at location. The file is overwritten if it already exists.
`create_transfers`	Boolean (Default FALSE). When true, generates transfers table, aggregating close stops, even if from different GTFS.
`transfer_distance`	Integer (Default 300). Upper straight-line distance limit in meters for transfers.
`transfer_time`	Integer (Default 120). Minimum time in seconds for transfers; all values below this will be replaced with this value, particularly all those defining in-place transfers where stop longitudes and latitudes remain identical.
`transfer_street_routing`	Boolean (Default FALSE). If TRUE, transfer times are calculated by routing throughout the underlying street network (downloaded automatically).

Details

Aggregates multiple feeds using `gtfstools::merge_gtfs()`. When generating transfers, those already existing in each GTFS file are kept, extended with new ones computed based on the stops network of the final aggregated version.

This computation is executed with `gtfsrouter::gtfs_transfer_table()`, with the parameters `d_limit=transfer_distance`, `min_transfer_time=transfer_time` and `network_times=transfer_street_routing`. The other parameters are applied the library default values.

For a detailed example, see the `vignette("unify")`.

Value

tidygtfs. The unified GTFS feed.

See Also

`gtfstools::merge_gtfs()`

`gtfsrouter::gtfs_transfer_table()`

Examples

```
# Load multiple GTFS files
gtfs_1 <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_tcb_sample.zip", package = "GTFShift")
)

summary(gtfs_1)

gtfs_1$agency
```

```
head(gtfs_1$trips)

gtfs_2 <- GTFShift::load_feed(system.file("extdata/samples",
  "gtfs_ttsl_sample_no_shapes.zip", package = "GTFShift")
)

summary(gtfs_2)

gtfs_2$agency

head(gtfs_2$trips)

# Unify them
unified <- GTFShift::unify(gtfs_1, gtfs_2, prefix = TRUE)

summary(unified)

unified$agency

head(unified$trips)
```

Index