Package 'r5r'

November 8, 2026

Type Package

Title Rapid Realistic Routing with 'R5'

Version 2.5.0

Description Rapid realistic routing on multimodal transport networks (walk, bike, public transport and car) using 'R5', the Rapid Realistic Routing on Real-world and Reimagined networks engine <https://github.com/conveyal/r5>. The package allows users to generate detailed routing analysis or calculate travel time and monetary cost matrices using seamless parallel computing on top of the R5 Java machine. While R5 is developed by Conveyal, the package r5r is independently developed by a team at the Institute for Applied Economic Research (Ipea) with contributions from collaborators. Apart from the documentation in this package, users will find additional information on R5 documentation at <https://docs.conveyal.com/>. Although we try to keep new releases of r5r in synchrony with R5, the development of R5 follows Conveyal's independent update process. Hence, users should confirm the R5 version implied by the Conveyal user manual (see <https://docs.conveyal.com/changelog>) corresponds with the R5 version that r5r depends on.

License GPL (>= 3) | MIT + file LICENSE

URL https://github.com/ipea/r5r, https://ipea.github.io/r5r/

BugReports https://github.com/ipea/r5r/issues

Depends R (>= 4.1.0)

Imports checkmate, cli, data.table, isoband, jsonlite, lifecycle, methods, purrr, rJava (>= 0.9-10), rlang, sf (>= 1.0-12), sfheaders, utils, zip

Suggests accessibility, covr, dplyr, fs, ggplot2 (>= 3.3.1), gtfstools (>= 2.0.0), h3jsr (>= 1.3.0), interp, knitr, mapview, parallel, patchwork, rJavaEnv, rmarkdown, testthat, withr

VignetteBuilder knitr

Encoding UTF-8

SystemRequirements Java JDK (>= 21.0)

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Marcus Saraiva [aut] (ORCID: <https://orcid.org/0000-0001-6218-2338>),
Rafael H. M. Pereira [aut, cre] (ORCID:
<https://orcid.org/0000-0003-2125-7465>),
Daniel Herszenhut [aut] (ORCID:
<https://orcid.org/0000-0001-8066-1105>),
Alex Magnus [aut],
Matthew Wigginton Bhagat-Conway [aut] (ORCID:
<https://orcid.org/0000-0002-1210-2982>),
Carlos Kaue Vieira Braga [ctb] (ORCID:
<https://orcid.org/0000-0002-6104-7297>),
Luyu Liu [ctb] (ORCID: <https://orcid.org/0000-0002-6684-5570>),
Egor Kotov [ctb] (ORCID: <https://orcid.org/0000-0001-6690-5345>),
Daniel Snow [ctb],
Ipea - Institute for Applied Economic Research [cph, fnd],
Department of Geography & Planning, University of Toronto [fnd]

Maintainer Rafael H. M. Pereira <rafa.pereira.br@gmail.com>

Repository CRAN

Date/Publication 2026-11-07 23:50:02 UTC

Contents

accessibility . . . 3
arrival_travel_time_matrix . . . 10
build_network . . . 16
check_transit_availability . . . 18
detailed_itineraries . . . 19
download_r5 . . . 26
expanded_travel_time_matrix . . . 27
find_snap . . . 33
get_gtfs_errors . . . 35
isochrone . . . 36
pareto_frontier . . . 41
r5r_cache . . . 47
r5r_sitrep . . . 47
read_fare_structure . . . 48
setup_fare_structure . . . 49
setup_r5 . . . 50
stop_r5 . . . 52
street_network_bbox . . . 53
street_network_to_sf . . . 54
transit_network_to_sf . . . 55
travel_time_matrix . . . 56
write_fare_structure . . . 62

Index **64**

accessibility *Calculate access to opportunities*

Description

Fast computation of access to opportunities given a selected decay function.

Usage

```
accessibility(
  r5r_network,
  origins,
  destinations,
  opportunities_colnames = "opportunities",
  mode = "WALK",
  mode_egress = "WALK",
  departure_datetime = NULL,
  time_window = 10L,
  percentiles = 50L,
  decay_function = "step",
  cutoffs = NULL,
  decay_value = NULL,
  max_walk_time = Inf,
  max_bike_time = Inf,
  max_car_time = Inf,
  max_trip_duration = 120L,
  walk_speed = 3.6,
  bike_speed = 12,
  max_rides = 3,
  max_lts = 2,
  fare_structure = NULL,
  max_fare = Inf,
  new_carspeeds = NULL,
  carspeed_scale = 1,
  new_lts = NULL,
  draws_per_minute = 5L,
  seed = NULL,
  n_threads = Inf,
  verbose = FALSE,
  progress = TRUE,
  output_dir = NULL,
  r5r_core = deprecated()
)
```

Arguments

r5r_network	A routable transport network created with build_network().

origins, destinations
: Either a POINT sf object with WGS84 CRS, or a data.frame containing the columns id, lon and lat.

opportunities_colnames
: A character vector. The names of the columns in the destinations input that tells the number of opportunities in each location. Several different column names can be passed, in which case the accessibility to each kind of opportunity will be calculated.

mode
: A character vector. The transport modes allowed for access, transfer and vehicle legs of the trips. Defaults to WALK. See details for other options.

mode_egress
: A character vector. The transport mode used after egress from the last public transport. It can be either WALK, BICYCLE or CAR. Defaults to WALK. Ignored when public transport is not used.

departure_datetime
: A POSIXct object. Only affects public transport legs; when routing with public transport, it must fall within the service period of the GTFS feeds (calendar.txt; see check_transit_availability()). Required for public transport; optional otherwise. See details for how datetimes are parsed.

time_window
: An integer. The time window in minutes. Departures are simulated every minute from departure_datetime until time_window minutes later, and travel times are summarized over these departures using percentiles (the median by default). With a fare_structure, and always in pareto_frontier(), departures are instead sampled at random within the window: draws_per_minute per minute on average (one in pareto_frontier()), different for each origin and the same in every call from the same origin with the same seed. Defaults to 10. See vignette("time_window", package = "r5r").

percentiles
: An integer vector (max length of 5, an R5 limit). The travel time percentiles within the time window from which accessibility is calculated. They apply to travel times, not to the accessibility distribution: with 25, accessibility is calculated from the 25th percentile travel time, which may differ from the 25th percentile of accessibility. Defaults to 50 (the median travel time). With more than one value, the output gets a column identifying the percentile of each estimate. See the R5 documentation at https://docs.conveyal.com/analysis/methodology#accounting-for-variability.

decay_function
: A string. Which decay function to use when calculating accessibility. One of step, exponential, fixed_exponential, linear or logistic. Defaults to step, which is equivalent to a cumulative opportunities measure. See details for how each alternative works and how they relate to the cutoffs and decay_value parameters.

cutoffs
: A numeric vector (maximum length of 12). This parameter has different effects for each decay function: it indicates the cutoff times in minutes when calculating cumulative opportunities accessibility with the step function (only trips strictly shorter than the cutoff are counted), the median (or inflection point) of the decay curves in the logistic and linear functions, and the half-life in the exponential function. It must be NULL when using the fixed_exponential function. Values must be whole numbers between 1 and 120 minutes (R5's limit) and are sorted in ascending order.

decay_value — A number. Extra parameter to be passed to the selected decay_function. Must be NULL when decay_function is either step or exponential.

max_walk_time — An integer. The maximum walking time (in minutes) to access and egress the transit network, make transfers, or complete walk-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In walk-only trips, the lower of max_walk_time and max_trip_duration applies.

max_bike_time — An integer. The maximum cycling time (in minutes) to access and egress the transit network, make transfers, or complete bicycle-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In bicycle-only trips, the lower of max_bike_time and max_trip_duration applies.

max_car_time — An integer. The maximum driving time (in minutes) to access and egress the transit network, or to complete car-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In car-only trips, the lower of max_car_time and max_trip_duration applies.

max_trip_duration — An integer. The maximum trip duration in minutes. Defaults to 120.

walk_speed — A numeric. Average walk speed in km/h. Defaults to 3.6.

bike_speed — A numeric. Average cycling speed in km/h. Defaults to 12.

max_rides — An integer. The maximum number of public transport rides allowed in the same trip. Defaults to 3.

max_lts — An integer between 1 and 4. The maximum level of traffic stress that cyclists will tolerate. A value of 1 means cyclists will only travel through the quietest streets, while a value of 4 indicates cyclists can travel through any road. Defaults to 2. See details.

fare_structure — A fare structure object, following the convention set in setup_fare_structure(). This object describes how transit fares should be calculated. See vignette("fare_structure", package = "r5r") for its structure.

max_fare — A number. The maximum value that trips can cost when calculating the fastest journey between each origin and destination pair. Defaults to Inf (no limit). A finite value requires a fare_structure; an error is raised otherwise.

new_carspeeds — A data.frame specifying the new car speed for each OSM edge id. This table must contain columns osm_id, max_speed and speed_type. The "speed_type" column is of class character and it indicates whether the values in "max_speed" should be interpreted as percentages of original speeds ("scale") or as absolute speeds ("km/h"). Alternatively, the new_carspeeds parameter can receive an sf data.frame with POLYGON geometry that indicates the new car speed for all the roads that fall within each polygon. In this case, the table must contain the columns poly_id with a unique id for each polygon, scale with the new speed scaling factors and priority, which is a number ranking which polygon should be considered in case of overlapping polygons. See more info in the scenarios vignette (vignette("scenarios", package = "r5r")).

carspeed_scale Numeric. The scaling factor applied to the car speed of road segments not specified in new_carspeeds. Defaults to 1, which keeps the speeds of the unlisted roads unchanged.

new_lts A data.frame specifying the new LTS levels for each OSM edge id. The table must contain columns osm_id and lts. Alternatively, the new_lts parameter can receive an sf data.frame with LINESTRING geometry. R5 will then find the nearest road for each LINESTRING and update its LTS value accordingly.

draws_per_minute An integer. The number of Monte Carlo draws to perform per minute of time_window. Defaults to 5. This would mean 300 draws in a 60-minute time window, for example. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a frequencies.txt table; otherwise, r5r still routes a departure every minute of time_window, in a deterministic way. In functions that accept a fare_structure, passing one switches to the McRaptor algorithm, which instead routes time_window * draws_per_minute departure times sampled at random within the window (different for each origin, and the same in every call from the same origin with the same seed). This parameter then affects the results with or without frequencies. R5's McRaptor cannot sample many departure times per minute over long windows, so such combinations raise an error (e.g. 8 draws per minute in a 10-minute window, or 5 in a window of 150 minutes or more), as does a total of exactly 200 draws.

seed An integer or NULL. The seed of the Monte Carlo draws of the departure offsets of frequency-based transit routes and, with the McRaptor algorithm, of the departure times sampled within time_window. Defaults to NULL, in which case the draws are random and results may vary slightly between runs. With an integer, the same inputs always give the same results, regardless of n_threads. Results are still a Monte Carlo sample, so a different seed may give slightly different results: report the seed used and check how sensitive the results are to it. Note that set.seed() has no effect on these draws. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a frequencies.txt table. In functions that accept a fare_structure, passing one switches to the McRaptor algorithm, and seed then affects the results with or without frequencies.

n_threads An integer. The number of threads to use when running the router in parallel. Defaults to Inf (all available threads).

verbose A logical. Whether to show R5 informative messages when running the function. Defaults to FALSE (R5 error messages are still shown). TRUE shows detailed output, useful for debugging issues not caught by r5r.

progress A logical. Whether to show a progress counter when running the router. Defaults to TRUE. Only works when verbose is FALSE. May slightly slow computation, as the counter is synchronized across threads.

output_dir Either NULL (the default) or a path to an existing directory. When a path is given, the function writes the results as .csv files to that directory and returns the path instead of the results. Useful in memory-constrained settings, as results are not loaded into RAM. Missing values (NA) are written as empty fields.

r5r_core The r5r_core argument is deprecated as of r5r v2.3.0. Use the r5r_network argument instead.

Value

A `data.table` with accessibility estimates for all origin points. This `data.table` contains the columns `id` (origin id), `opportunity` (the type of opportunities to which accessibility was calculated), `percentile` (the travel time percentile considered in the estimate), `cutoff` (the specified cutoff values, except when `decay_function` is `fixed_exponential`, in which case the `cutoff` parameter is not used) and `accessibility` (the accessibility estimate). Origins that cannot be snapped to the street network get an accessibility of 0 (see `find_snap()`). If `output_dir` is not `NULL`, the function returns the path specified in that parameter, in which the `.csv` files containing the results are saved. With `fixed_exponential`, these files keep a `cutoff` column filled with the placeholder value `0`.

Decay functions

`R5` allows one to use different decay functions when calculating accessibility. See the original `R5` documentation from Conveyal for more information on each one (https://docs.conveyal.com/learn-more/decay-functions). A summary of each available option, as well as the value passed to `decay_function` to use it (inside parentheses) are listed below:

- **Step**, also known as cumulative opportunities (`"step"`):
  a binary decay function used to find the sum of available opportunities within a specific travel time cutoff.
- **Logistic CDF** (`"logistic"`):
  This is the logistic function, i.e. the cumulative distribution function of the logistic distribution, expressed such that its parameters are the median (inflection point) and standard deviation. This function applies a sigmoid rolloff that has a convenient relationship to discrete choice theory. Its parameters can be set to reflect a whole population's tolerance for making trips with different travel times. The function's value represents the probability that a randomly chosen member of the population would accept making a trip, given its duration. Opportunities are then weighted by how likely it is that a person would consider them "reachable".
  - Calibration: The median parameter is controlled by the `cutoff` parameter, leaving only the standard deviation to configure through the `decay_value` parameter.
- **Fixed Exponential** (`"fixed_exponential"`):
  This function is of the form `exp(-Lt)` where L is a single fixed decay constant in the range (0, 1) and t is the travel time in **seconds**. It is constrained to be positive to ensure weights decrease (rather than grow) with increasing travel time. Note that L is a per-second constant: to use a decay constant expressed per minute (e.g. from the literature), pass it divided by 60.
  - Calibration: This function is controlled exclusively by the `L` constant, given by the `decay_value` parameter. `cutoffs` must be `NULL`.
- **Half-life Exponential Decay** (`"exponential"`):
  This is similar to the fixed-exponential option above, but in this case the decay parameter is inferred from the `cutoffs` parameter values, which is treated as the half-life of the decay.
- **Linear** (`"linear"`):
  This is a simple, vaguely sigmoid option, which may be useful when you have a sense of a maximum travel time that would be tolerated by any traveler, and a minimum time below which all travel is perceived to be equally easy.

  - Calibration: The transition region is transposable and symmetric around the `cutoffs` parameter values, taking `decay_value` minutes to taper down from one to zero. `decay_value` must be a whole number of minutes between 1 and 59.

Transport modes

`R5` allows for multiple combinations of transport modes. The options include:

- **Transit modes:** `TRAM`, `SUBWAY`, `RAIL`, `BUS`, `FERRY`, `CABLE_CAR`, `GONDOLA`, `FUNICULAR`. The option `TRANSIT` automatically considers all public transport modes available.
- **Non transit modes:** `WALK`, `BICYCLE`, `CAR`.

Level of Traffic Stress (LTS)

When cycling is enabled in `R5` (by passing the value `BICYCLE` to either `mode` or `mode_egress`), setting `max_lts` will allow cycling only on streets with a given level of danger/stress. Setting `max_lts` to 1, for example, will allow cycling only on separated bicycle infrastructure or low-traffic streets and routing will revert to walking when traversing any links with LTS exceeding 1. Setting `max_lts` to 3 will allow cycling on links with LTS 1, 2 or 3. Routing also reverts to walking if the street segment is tagged as non-bikable in OSM (e.g. a staircase), independently of the specified max LTS.

The default methodology for assigning LTS values to network edges is based on commonly tagged attributes of OSM ways. See more info about LTS in the original documentation of R5 from Conveyal at https://docs.conveyal.com/learn-more/traffic-stress. In summary:

- **LTS 1**: Tolerable for children. This includes low-speed, low-volume streets, as well as those with separated bicycle facilities (such as parking-protected lanes or cycle tracks).
- **LTS 2**: Tolerable for the mainstream adult population. This includes streets where cyclists have dedicated lanes and only have to interact with traffic at formal crossing.
- **LTS 3**: Tolerable for "enthused and confident" cyclists. This includes streets which may involve close proximity to moderate- or high-speed vehicular traffic.
- **LTS 4**: Tolerable only for "strong and fearless" cyclists. This includes streets where cyclists are required to mix with moderate- to high-speed vehicular traffic.

For advanced users, you can provide custom LTS values by adding a tag `<key = "lts">` to the `osm.pbf` file.

Datetime parsing

`r5r` ignores the timezone attribute of datetime objects when parsing dates and times, using the study area's timezone instead. For example, let's say you are running some calculations using Rio de Janeiro, Brazil, as your study area. The datetime `as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S")` will be parsed as May 13th, 2019, 14:00h in Rio's local time, as expected. But `as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S", tz = "Europe/Paris")` will also be parsed as the exact same date and time in Rio's local time, perhaps surprisingly, ignoring the timezone attribute.

Routing algorithm

The travel_time_matrix(), expanded_travel_time_matrix(), arrival_travel_time_matrix() and accessibility() functions use an R5-specific extension to the RAPTOR routing algorithm (see Conway et al., 2017). This RAPTOR extension uses a systematic sample of one departure per minute over the time window set by the user in the 'time_window' parameter. A detailed description of base RAPTOR can be found in Delling et al (2015). However, whenever the user includes transit fares inputs to these functions, they automatically switch to use an R5-specific extension to the McRAPTOR routing algorithm. detailed_itineraries() also uses RAPTOR when it only returns the fastest itinerary (shortest_path = TRUE) and no fares are used.

- Conway, M. W., Byrd, A., & van der Linden, M. (2017). Evidence-based transit and land use sketch planning using interactive accessibility methods on combined schedule and headway-based networks. Transportation Research Record, 2653(1), 45-53. doi:10.3141/265306
- Delling, D., Pajor, T., & Werneck, R. F. (2015). Round-based public transit routing. Transportation Science, 49(3), 591-604. doi:10.1287/trsc.2014.0534

Examples

```
library(r5r)

data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path)
points <- read.csv(file.path(data_path, "poa_hexgrid.csv"))[1:500, ]

departure_datetime <- as.POSIXct(
  "13-05-2019 14:00:00",
  format = "%d-%m-%Y %H:%M:%S"
)

access <- accessibility(
  r5r_network ,
  origins = points,
  destinations = points,
  opportunities_colnames = "schools",
  mode = "WALK",
  departure_datetime = departure_datetime,
  decay_function = "step",
  cutoffs = 30,
  max_trip_duration = 30
)
head(access)

# using a different decay function
access <- accessibility(
  r5r_network ,
  origins = points,
  destinations = points,
  opportunities_colnames = "schools",
  mode = "WALK",
  departure_datetime = departure_datetime,
  decay_function = "logistic",
```

```
  cutoffs = 30,
  decay_value = 1,
  max_trip_duration = 30
)
head(access)

# using several cutoff values
access <- accessibility(
  r5r_network ,
  origins = points,
  destinations = points,
  opportunities_colnames = "schools",
  mode = "WALK",
  departure_datetime = departure_datetime,
  decay_function = "step",
  cutoffs = c(15, 30),
  max_trip_duration = 30
)
head(access)

# calculating access to different types of opportunities
access <- accessibility(
  r5r_network ,
  origins = points,
  destinations = points,
  opportunities_colnames = c("schools", "healthcare"),
  mode = "WALK",
  departure_datetime = departure_datetime,
  decay_function = "step",
  cutoffs = 30,
  max_trip_duration = 30
)
head(access)

stop_r5(r5r_network )
```

`arrival_travel_time_matrix` *Calculate travel time matrix between origin destination pairs considering a time of arrival*

Description

Computes travel times between origin-destination pairs for a given arrival time: for each pair, the trip with the latest departure that arrives by `arrival_datetime`. Departures are searched minute by minute between `arrival_datetime` - `max_trip_duration` and `arrival_datetime`, so `max_trip_duration` also sets the search window. For a departure time, use `travel_time_matrix()`. This function wraps `expanded_travel_time_matrix()`, so its output has more columns than that of `travel_time_matrix()`, and it can be very memory intensive with a long `max_trip_duration`.

destinations can have at most 5000 rows, a limit of R5 for detailed path information; split larger sets into chunks.

Usage

```
arrival_travel_time_matrix(
  r5r_network,
  origins,
  destinations,
  mode = "WALK",
  mode_egress = "WALK",
  arrival_datetime,
  breakdown = FALSE,
  max_walk_time = Inf,
  max_bike_time = Inf,
  max_car_time = Inf,
  max_trip_duration = 120L,
  walk_speed = 3.6,
  bike_speed = 12,
  max_rides = 3,
  max_lts = 2,
  new_carspeeds = NULL,
  carspeed_scale = 1,
  new_lts = NULL,
  draws_per_minute = 5L,
  seed = NULL,
  n_threads = Inf,
  verbose = FALSE,
  progress = TRUE,
  output_dir = NULL,
  r5r_core = deprecated()
)
```

Arguments

r5r_network	A routable transport network created with build_network().
origins, destinations	Either a POINT sf object with WGS84 CRS, or a data.frame containing the columns id, lon and lat.
mode	A character vector. The transport modes allowed for access, transfer and vehicle legs of the trips. Defaults to WALK. See details for other options.
mode_egress	A character vector. The transport mode used after egress from the last public transport. It can be either WALK, BICYCLE or CAR. Defaults to WALK. Ignored when public transport is not used.
arrival_datetime	A POSIXct object, required for all modes. The time by which trips must arrive. When routing with public transport, services must run on the date of

arrival_datetime - max_trip_duration (see check_transit_availability()). See details for how datetimes are parsed.

breakdown
: A logical. Whether to include detailed information about each trip in the output. If FALSE (the default), the output lists the total time between each origin-destination pair and the routes used to complete the trip for each minute of the specified time window. If TRUE, the output also includes the access, waiting, in-vehicle, transfer and egress time of each trip, and its number of public transport rides. For trips made only by walking, cycling or driving (routes equal to [WALK], [BICYCLE] or [CAR]), these components are all 0 and total_time holds the whole trip.

max_walk_time
: An integer. The maximum walking time (in minutes) to access and egress the transit network, make transfers, or complete walk-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In walk-only trips, the lower of max_walk_time and max_trip_duration applies.

max_bike_time
: An integer. The maximum cycling time (in minutes) to access and egress the transit network, make transfers, or complete bicycle-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In bicycle-only trips, the lower of max_bike_time and max_trip_duration applies.

max_car_time
: An integer. The maximum driving time (in minutes) to access and egress the transit network, or to complete car-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In car-only trips, the lower of max_car_time and max_trip_duration applies.

max_trip_duration
: An integer. The maximum trip duration in minutes. Defaults to 120.

walk_speed
: A numeric. Average walk speed in km/h. Defaults to 3.6.

bike_speed
: A numeric. Average cycling speed in km/h. Defaults to 12.

max_rides
: An integer. The maximum number of public transport rides allowed in the same trip. Defaults to 3.

max_lts
: An integer between 1 and 4. The maximum level of traffic stress that cyclists will tolerate. A value of 1 means cyclists will only travel through the quietest streets, while a value of 4 indicates cyclists can travel through any road. Defaults to 2. See details.

new_carspeeds
: A data.frame specifying the new car speed for each OSM edge id. This table must contain columns osm_id, max_speed and speed_type. The "speed_type" column is of class character and it indicates whether the values in "max_speed" should be interpreted as percentages of original speeds ("scale") or as absolute speeds ("km/h"). Alternatively, the new_carspeeds parameter can receive an sf data.frame with POLYGON geometry that indicates the new car speed for all the roads that fall within each polygon. In this case, the table must contain the columns poly_id with a unique id for each polygon, scale with the new speed scaling factors and priority, which is a number ranking which polygon should

be considered in case of overlapping polygons. See more info in the scenarios vignette (`vignette("scenarios", package = "r5r")`).

`carspeed_scale` Numeric. The scaling factor applied to the car speed of road segments not specified in `new_carspeeds`. Defaults to `1`, which keeps the speeds of the unlisted roads unchanged.

`new_lts` A `data.frame` specifying the new LTS levels for each OSM edge id. The table must contain columns `osm_id` and `lts`. Alternatively, the `new_lts` parameter can receive an `sf data.frame` with LINESTRING geometry. R5 will then find the nearest road for each LINESTRING and update its LTS value accordingly.

`draws_per_minute`

An integer. The number of Monte Carlo draws to perform per minute of `time_window`. Defaults to 5. This would mean 300 draws in a 60-minute time window, for example. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a `frequencies.txt` table; otherwise, r5r still routes a departure every minute of `time_window`, in a deterministic way. In functions that accept a `fare_structure`, passing one switches to the McRaptor algorithm, which instead routes `time_window * draws_per_minute` departure times sampled at random within the window (different for each origin, and the same in every call from the same origin with the same `seed`). This parameter then affects the results with or without frequencies. R5's McRaptor cannot sample many departure times per minute over long windows, so such combinations raise an error (e.g. 8 draws per minute in a 10-minute window, or 5 in a window of 150 minutes or more), as does a total of exactly 200 draws.

`seed` An integer or `NULL`. The seed of the Monte Carlo draws of the departure offsets of frequency-based transit routes and, with the McRaptor algorithm, of the departure times sampled within `time_window`. Defaults to `NULL`, in which case the draws are random and results may vary slightly between runs. With an integer, the same inputs always give the same results, regardless of `n_threads`. Results are still a Monte Carlo sample, so a different seed may give slightly different results: report the seed used and check how sensitive the results are to it. Note that `set.seed()` has no effect on these draws. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a `frequencies.txt` table. In functions that accept a `fare_structure`, passing one switches to the McRaptor algorithm, and `seed` then affects the results with or without frequencies.

`n_threads` An integer. The number of threads to use when running the router in parallel. Defaults to `Inf` (all available threads).

`verbose` A logical. Whether to show `R5` informative messages when running the function. Defaults to `FALSE` (`R5` error messages are still shown). `TRUE` shows detailed output, useful for debugging issues not caught by `r5r`.

`progress` A logical. Whether to show a progress counter when running the router. Defaults to `TRUE`. Only works when `verbose` is `FALSE`. May slightly slow computation, as the counter is synchronized across threads.

`output_dir` Either `NULL` (the default) or a path to an existing directory. When a path is given, the function writes the results as `.csv` files to that directory and returns the path instead of the results. Useful in memory-constrained settings, as results are not loaded into RAM. Missing values (`NA`) are written as empty fields.

r5r_core The r5r_core argument is deprecated as of r5r v2.3.0. Use the r5r_network argument instead.

Value

A data.table with one row per origin-destination pair that can be reached by arrival_datetime, describing the trip with the latest departure that still arrives in time: its departure_time, the routes used and its total_time (in minutes), plus the columns added by breakdown = TRUE (see expanded_travel_time_matrix()). Pairs that cannot be reached in time are absent from the output. When the search window crosses midnight, departure times after midnight are reported as "24:MM:SS", and only the public transport services of the departure day are considered. Trips made only by walking, cycling or driving have travel times in whole minutes, so they can arrive up to 59 seconds after arrival_datetime. If output_dir is not NULL, the function returns the path specified in that parameter, in which the .csv files containing the results are saved.

Transport modes

R5 allows for multiple combinations of transport modes. The options include:

- **Transit modes:** TRAM, SUBWAY, RAIL, BUS, FERRY, CABLE_CAR, GONDOLA, FUNICULAR. The option TRANSIT automatically considers all public transport modes available.
- **Non transit modes:** WALK, BICYCLE, CAR.

Level of Traffic Stress (LTS)

When cycling is enabled in R5 (by passing the value BICYCLE to either mode or mode_egress), setting max_lts will allow cycling only on streets with a given level of danger/stress. Setting max_lts to 1, for example, will allow cycling only on separated bicycle infrastructure or low-traffic streets and routing will revert to walking when traversing any links with LTS exceeding 1. Setting max_lts to 3 will allow cycling on links with LTS 1, 2 or 3. Routing also reverts to walking if the street segment is tagged as non-bikable in OSM (e.g. a staircase), independently of the specified max LTS.

The default methodology for assigning LTS values to network edges is based on commonly tagged attributes of OSM ways. See more info about LTS in the original documentation of R5 from Conveyal at https://docs.conveyal.com/learn-more/traffic-stress. In summary:

- **LTS 1**: Tolerable for children. This includes low-speed, low-volume streets, as well as those with separated bicycle facilities (such as parking-protected lanes or cycle tracks).
- **LTS 2**: Tolerable for the mainstream adult population. This includes streets where cyclists have dedicated lanes and only have to interact with traffic at formal crossing.
- **LTS 3**: Tolerable for "enthused and confident" cyclists. This includes streets which may involve close proximity to moderate- or high-speed vehicular traffic.
- **LTS 4**: Tolerable only for "strong and fearless" cyclists. This includes streets where cyclists are required to mix with moderate- to high-speed vehicular traffic.

For advanced users, you can provide custom LTS values by adding a tag <key = "lts"> to the osm.pbf file.

Datetime parsing

r5r ignores the timezone attribute of datetime objects when parsing dates and times, using the study area's timezone instead. For example, let's say you are running some calculations using Rio de Janeiro, Brazil, as your study area. The datetime `as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S")` will be parsed as May 13th, 2019, 14:00h in Rio's local time, as expected. But `as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S", tz = "Europe/Paris")` will also be parsed as the exact same date and time in Rio's local time, perhaps surprisingly, ignoring the timezone attribute.

Routing algorithm

The travel_time_matrix(), expanded_travel_time_matrix(), arrival_travel_time_matrix() and accessibility() functions use an R5-specific extension to the RAPTOR routing algorithm (see Conway et al., 2017). This RAPTOR extension uses a systematic sample of one departure per minute over the time window set by the user in the 'time_window' parameter. A detailed description of base RAPTOR can be found in Delling et al (2015). However, whenever the user includes transit fares inputs to these functions, they automatically switch to use an R5-specific extension to the McRAPTOR routing algorithm. detailed_itineraries() also uses RAPTOR when it only returns the fastest itinerary (shortest_path = TRUE) and no fares are used.

- Conway, M. W., Byrd, A., & van der Linden, M. (2017). Evidence-based transit and land use sketch planning using interactive accessibility methods on combined schedule and headway-based networks. Transportation Research Record, 2653(1), 45-53. doi:10.3141/265306
- Delling, D., Pajor, T., & Werneck, R. F. (2015). Round-based public transit routing. Transportation Science, 49(3), 591-604. doi:10.1287/trsc.2014.0534

See Also

Other routing: detailed_itineraries(), expanded_travel_time_matrix(), pareto_frontier(), travel_time_matrix()

Examples

```
library(r5r)

# build transport network
data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path )

# load origin/destination points
points <- read.csv(file.path(data_path, "poa_points_of_interest.csv"))

arrival_datetime <- as.POSIXct(
  "13-05-2019 14:00:00",
  format = "%d-%m-%Y %H:%M:%S"
)

# by default only returns the total time between each pair in each minute of
# the specified time window
arrival_ttm <- arrival_travel_time_matrix(
```

```
  r5r_network,
  origins = points,
  destinations = points,
  mode = c("WALK", "TRANSIT"),
  arrival_datetime = arrival_datetime,
  max_trip_duration = 60
)

head(arrival_ttm)

# when breakdown = TRUE the output contains much more information
arrival_ttm2 <- arrival_travel_time_matrix(
  r5r_network,
  origins = points,
  destinations = points,
  mode = c("WALK", "TRANSIT"),
  arrival_datetime = arrival_datetime,
  max_trip_duration = 60,
  breakdown = TRUE
)

head(arrival_ttm2)

stop_r5(r5r_network)
```

`build_network` *Build a transport network used for routing in R5*

Description

Builds a multimodal transport network used for routing in R5, combining multiple data inputs present in the directory where the network should be saved to. The directory must contain only one street network file (in `.osm.pbf` format). It may optionally contain one or more public transport GTFS feeds (in `.zip` format), when used for public transport routing, and a `.tif` file describing the elevation profile of the study area. If there is more than one GTFS feed in the directory, all feeds are automatically merged. If there is already a `'network.dat'` file in the directory, the function reads it instead (see `overwrite`).

Usage

```
build_network(
  data_path,
  verbose = FALSE,
  temp_dir = FALSE,
  elevation = "TOBLER",
  overwrite = FALSE
)
```

Arguments

data_path	A string pointing to the directory where data inputs are stored and where the built network.dat will be saved.
verbose	A logical. Whether to show R5 informative messages when running the function. Defaults to FALSE (R5 error messages are still shown). TRUE shows detailed output, useful for debugging issues not caught by r5r.
temp_dir	A logical. Whether the network.dat file should be saved to a temporary directory. Defaults to FALSE.
elevation	A string. The name of the impedance function to be used to calculate impedance for walking and cycling based on street slopes. Available options include TOBLER (Default) and MINETTI, or NONE to ignore elevation. R5 loads elevation data from .tif files saved inside the data_path directory. Elevation raster must be in WGS 84 (EPSG:4326) coordinate reference system. See more info in the Details section below.
overwrite	A logical. Whether to overwrite an existing network.dat or to use a cached file. Defaults to FALSE (i.e. use a cached network).

Value

An r5r_network object, used by r5r's routing and support functions.

Elevation

More information about the TOBLER and MINETTI options to calculate the effects of elevation on travel times can be found in the references below:

- Campbell, M. J., et al (2019). Using crowdsourced fitness tracker data to model the relationship between slope and travel rates. Applied geography, 106, 93-107. doi:10.1016/j.apgeog.2019.03.008.
- Minetti, A. E., et al (2002). Energy cost of walking and running at extreme uphill and downhill slopes. Journal of applied physiology. doi:10.1152/japplphysiol.01177.2001.
- Tobler, W. (1993). Three presentations on geographical analysis and modeling: Non-isotropic geographic modeling speculations on the geometry of geography global spatial analysis. Technical Report. National center for geographic information and analysis. 93 (1). https://escholarship.org/uc/item/05r820mz.

See Also

Other Build network: download_r5(), setup_r5()

Examples

```
library(r5r)

# directory with street network and gtfs files
data_path <- system.file("extdata/poa", package = "r5r")

r5r_network <- build_network(data_path)
```

`check_transit_availability` *Check transit service availability by date*

Description

Counts the public transport services in the GTFS feeds of a `r5r_network` that are active on given dates, and their proportion of all services. Use it to pick a departure date with adequate service: public transport routing only finds trips on dates when services run (per the GTFS calendars and date exceptions).

Usage

```
check_transit_availability(
  r5r_network,
  dates = NULL,
  start_date = NULL,
  end_date = NULL,
  r5r_core = deprecated()
)
```

Arguments

`r5r_network`	A routable transport network created with `build_network()`.
`dates`	A vector of specific dates to be checked. Can be character strings in "YYYY-MM-DD" format, or objects of class `Date`. This argument cannot be used with `start_date` or `end_date`. Defaults to `NULL`; when all date arguments are `NULL`, operating dates are discovered from the network.
`start_date`	The start date for a continuous date range. Must be a single character string in "YYYY-MM-DD" format or a `Date` object. Must be used with `end_date`.
`end_date`	The end date for a continuous date range. Must be a single character string in "YYYY-MM-DD" format or a `Date` object. Must be used with `start_date`.
`r5r_core`	The `r5r_core` argument is deprecated as of r5r v2.3.0. Use the `r5r_network` argument instead.

Details

When no date arguments are supplied, the function discovers all dates with an active service referenced by a retained transit trip, including frequency-based trips. This accounts for weekday calendars and date exceptions, including exception-only services. A date with active service does not guarantee a trip at every departure time or between every pair of points.

You can also specify the dates to check in two ways:

- Using the `dates` argument to provide a vector of specific dates.
- Using the `start_date` and `end_date` arguments to provide a continuous date range.

You cannot combine these methods. In all modes, `total_services` counts every imported service, not only those used by trips; counts are of services, not trips. If calendar bounds are invalid, automatic discovery reports the affected service; explicit date queries remain available.

Value

A `data.table` with four columns: `date`, `total_services`, `active_services`, and `pct_active` (the proportion of active services). Automatic discovery returns dates in ascending order, or a zero-row table with these columns if no retained transit trips operate.

Examples

```
library(r5r)
data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path)

# Discover operating dates without knowing the feed's date range
availability1 <- check_transit_availability(r5r_network)
head(availability1)
# plot availability over the year
library(ggplot2)
ggplot(availability1, aes(x = date, y = pct_active)) +
  geom_line() +
  geom_point() +
  theme_classic(base_size = 16)

# Example 2: Check a vector of specific dates
# Let's check a regular weekday and a Sunday, where service may differ.
dates_to_check <- c("2019-05-13", "2019-05-19")
availability2 <- check_transit_availability(
  r5r_network,
  dates = dates_to_check
)
head(availability2)

# Example 3: Check a continuous date range using start_date and end_date
availability3 <- check_transit_availability(
  r5r_network,
  start_date = "2019-01-01",
  end_date = "2019-12-31"
)
availability3[121:124,]


stop_r5(r5r_network)
```

`detailed_itineraries` *Detailed itineraries between origin-destination pairs*

Description

Returns detailed trip information between origin-destination pairs: the waiting and moving time, distance, route and geometry of each trip leg. Designed for trip planning (like Google's Directions API or OpenTripPlanner), it takes much more time and memory than the package's other, more analytical, routing functions.

Usage

```
detailed_itineraries(
  r5r_network,
  origins,
  destinations,
  mode = "WALK",
  mode_egress = "WALK",
  departure_datetime = NULL,
  time_window = 10L,
  suboptimal_minutes = 0L,
  max_walk_time = Inf,
  max_bike_time = Inf,
  max_car_time = Inf,
  max_trip_duration = 120L,
  walk_speed = 3.6,
  bike_speed = 12,
  max_rides = 3,
  max_lts = 2,
  shortest_path = TRUE,
  all_to_all = FALSE,
  fare_structure = NULL,
  max_fare = Inf,
  new_carspeeds = NULL,
  carspeed_scale = 1,
  new_lts = NULL,
  n_threads = Inf,
  verbose = FALSE,
  progress = TRUE,
  drop_geometry = FALSE,
  osm_link_ids = FALSE,
  output_dir = NULL,
  r5r_core = deprecated()
)
```

Arguments

`r5r_network`	A routable transport network created with `build_network()`.
`origins, destinations`	Either a `POINT sf` object with WGS84 CRS, or a `data.frame` containing the columns `id`, `lon` and `lat`.

mode
: A character vector. The transport modes allowed for access, transfer and vehicle legs of the trips. Defaults to WALK. See details for other options.

mode_egress
: A character vector. The transport mode used after egress from the last public transport. It can be either WALK, BICYCLE or CAR. Defaults to WALK. Ignored when public transport is not used.

departure_datetime
: A POSIXct object. Only affects public transport legs; when routing with public transport, it must fall within the service period of the GTFS feeds (calendar.txt; see check_transit_availability()). Required for public transport; optional otherwise. See details for how datetimes are parsed.

time_window
: An integer. The time window in minutes for which r5r will calculate multiple itineraries. Defaults to 10. Unlike other routing functions, results are not aggregated by percentile, so the output may contain trips departing after departure_datetime but within the window. With shortest_path = TRUE and no fare_structure, a trip is routed for each minute from departure_datetime until time_window minutes later, and the fastest one is returned (ties go to the earliest departure). Otherwise, departures are simulated at random seconds, about one per minute (30 to 90 seconds apart), with the same departure times in every call with the same origin, and each departure is routed separately: an itinerary may appear or disappear when departure_datetime changes by one minute, and when the same sequence of routes appears in several departures, only the fastest is kept. See vignette("time_window", package = "r5r").

suboptimal_minutes
: A number. How many minutes after the optimal arrival a non-optimal McRAPTOR branch may arrive and still be kept (e.g. with 10, routes arriving up to 10 minutes after the fastest one are considered). It can only be used when shortest_path = FALSE and fare_structure is NULL. The comparison is made separately for each departure within time_window and uses arrival time only: a route beyond this margin is discarded even if it has fewer transfers. The default, 0, keeps only routes arriving at the earliest time for each departure. Higher values return more itineraries, emulating people who take a slower route for practical reasons (e.g. mode preference, safety).

max_walk_time
: An integer. The maximum walking time (in minutes) to access and egress the transit network, make transfers, or complete walk-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In walk-only trips, the lower of max_walk_time and max_trip_duration applies.

max_bike_time
: An integer. The maximum cycling time (in minutes) to access and egress the transit network, make transfers, or complete bicycle-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In bicycle-only trips, the lower of max_bike_time and max_trip_duration applies.

max_car_time
: An integer. The maximum driving time (in minutes) to access and egress the transit network, or to complete car-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it).

Defaults to Inf (no limit besides max_trip_duration). In car-only trips, the lower of max_car_time and max_trip_duration applies.

max_trip_duration
: An integer. The maximum trip duration in minutes. Defaults to 120.

walk_speed
: A numeric. Average walk speed in km/h. Defaults to 3.6.

bike_speed
: A numeric. Average cycling speed in km/h. Defaults to 12.

max_rides
: An integer. The maximum number of public transport rides allowed in the same trip. Defaults to 3.

max_lts
: An integer between 1 and 4. The maximum level of traffic stress that cyclists will tolerate. A value of 1 means cyclists will only travel through the quietest streets, while a value of 4 indicates cyclists can travel through any road. Defaults to 2. See details.

shortest_path
: A logical. Whether the function should only return the fastest itinerary between each origin and destination pair (the default) or multiple alternatives. The fastest itinerary is the one with the shortest travel time, measured from its own departure time, among all departures within time_window. With a large time_window, it may depart later and arrive later than other itineraries; use time_window = 1 to get the itinerary with the earliest arrival for a given departure time. Without a fare_structure, public transport trips are then routed with R5's RAPTOR algorithm, which is much faster than the McRAPTOR algorithm used for alternative routes and fares (see the Fastest itinerary section).

all_to_all
: A logical. Whether to query routes between the 1st origin to the 1st destination, then the 2nd origin to the 2nd destination, and so on (FALSE, the default) or to query routes between all origins to all destinations (TRUE). With shortest_path = TRUE and no fare_structure, all_to_all = TRUE runs one public transport search per origin for all its destinations, which is much faster than listing the same pairs with all_to_all = FALSE.

fare_structure
: A fare structure object, following the convention set in setup_fare_structure(). This object describes how transit fares should be calculated. See vignette("fare_structure", package = "r5r") for its structure.

max_fare
: A number. The maximum value that trips can cost when calculating the fastest journey between each origin and destination pair. Defaults to Inf (no limit). A finite value requires a fare_structure; an error is raised otherwise.

new_carspeeds
: A data.frame specifying the new car speed for each OSM edge id. This table must contain columns osm_id, max_speed and speed_type. The "speed_type" column is of class character and it indicates whether the values in "max_speed" should be interpreted as percentages of original speeds ("scale") or as absolute speeds ("km/h"). Alternatively, the new_carspeeds parameter can receive an sf data.frame with POLYGON geometry that indicates the new car speed for all the roads that fall within each polygon. In this case, the table must contain the columns poly_id with a unique id for each polygon, scale with the new speed scaling factors and priority, which is a number ranking which polygon should be considered in case of overlapping polygons. See more info in the scenarios vignette (vignette("scenarios", package = "r5r")).

carspeed_scale
: Numeric. The scaling factor applied to the car speed of road segments not specified in new_carspeeds. Defaults to 1, which keeps the speeds of the unlisted roads unchanged.

new_lts	A data.frame specifying the new LTS levels for each OSM edge id. The table must contain columns osm_id and lts. Alternatively, the new_lts parameter can receive an sf data.frame with LINESTRING geometry. R5 will then find the nearest road for each LINESTRING and update its LTS value accordingly.
n_threads	An integer. The number of threads to use when running the router in parallel. Defaults to Inf (all available threads).
verbose	A logical. Whether to show R5 informative messages when running the function. Defaults to FALSE (R5 error messages are still shown). TRUE shows detailed output, useful for debugging issues not caught by r5r.
progress	A logical. Whether to show a progress counter when running the router. Defaults to TRUE. Only works when verbose is FALSE. May slightly slow computation, as the counter is synchronized across threads.
drop_geometry	A logical. Whether to drop the geometry of each trip leg from the output. Defaults to FALSE (geometry kept).
osm_link_ids	A logical. Whether the output should include additional columns: osm_id_list for the OSM ids of the road segments used along the trip geometry, edge_id_list for the ids of the internal street network edges used, and board_stop_id and alight_stop_id for the GTFS ids of transit boarding and alighting stops, prefixed by the feed name (e.g. "poa_eptc:1649"). osm_id_list and edge_id_list are character columns formatted as "[id1, id2, ...]", and are "[]" for transit segments. Can only be TRUE when drop_geometry is FALSE. Defaults to FALSE. Keep in mind that osm_id_list will contain an id even if the route only uses a small stretch of the road (e.g. 5m of a 600m street segment). For more precision, use edge_id_list, which returns the exact internal edge segments used in the trip. You can inspect these edge ids and their associated properties, including OSM ids, with street_network_to_sf().
output_dir	Either NULL (the default) or a path to an existing directory. When a path is given, the function writes the results as .csv files to that directory and returns the path instead of the results. Useful in memory-constrained settings, as results are not loaded into RAM. Missing values (NA) are written as empty fields.
r5r_core	The r5r_core argument is deprecated as of r5r v2.3.0. Use the r5r_network argument instead.

Value

When drop_geometry is FALSE, the function outputs a LINESTRING sf with detailed information on the itineraries between the specified origins and destinations. When TRUE, the output is a data.table. All distances are in meters and travel times are in minutes. Each row is one segment (trip leg) of one itinerary, with the columns:

- from_id, from_lat, from_lon, to_id, to_lat, to_lon: the origin and destination of the pair;
- option: the itinerary number within the pair. Options are ordered with direct (non-transit) trips first, then by number of segments, then by duration, so option 1 is not necessarily the fastest one. With shortest_path = TRUE, each pair has a single itinerary (option 1);
- departure_time: the departure time of the itinerary ("HH:MM:SS");

- total_duration, total_distance: duration and distance of the whole itinerary;
- segment, mode, segment_duration, wait, distance, route: the segment number, its transport mode, its in-motion duration, the waiting time before it, its distance and the transit route used (empty for non-transit segments);
- total_fare, cumulative_fare: only when a fare_structure is used;
- osm_id_list, edge_id_list, board_stop_id, alight_stop_id: only when osm_link_ids is TRUE;
- geometry: only when drop_geometry is FALSE.

If output_dir is not NULL, the function returns the path specified in that parameter, in which the .csv files containing the results are saved. This function writes one file per origin-destination pair, named from_<origin id>_to_<destination id>.csv, so pairs must be unique and ids cannot contain characters that are not allowed in file names.

Transport modes

R5 allows for multiple combinations of transport modes. The options include:

- **Transit modes:** TRAM, SUBWAY, RAIL, BUS, FERRY, CABLE_CAR, GONDOLA, FUNICULAR. The option TRANSIT automatically considers all public transport modes available.
- **Non transit modes:** WALK, BICYCLE, CAR.

Level of Traffic Stress (LTS)

When cycling is enabled in R5 (by passing the value BICYCLE to either mode or mode_egress), setting max_lts will allow cycling only on streets with a given level of danger/stress. Setting max_lts to 1, for example, will allow cycling only on separated bicycle infrastructure or low-traffic streets and routing will revert to walking when traversing any links with LTS exceeding 1. Setting max_lts to 3 will allow cycling on links with LTS 1, 2 or 3. Routing also reverts to walking if the street segment is tagged as non-bikable in OSM (e.g. a staircase), independently of the specified max LTS.

The default methodology for assigning LTS values to network edges is based on commonly tagged attributes of OSM ways. See more info about LTS in the original documentation of R5 from Conveyal at https://docs.conveyal.com/learn-more/traffic-stress. In summary:

- **LTS 1**: Tolerable for children. This includes low-speed, low-volume streets, as well as those with separated bicycle facilities (such as parking-protected lanes or cycle tracks).
- **LTS 2**: Tolerable for the mainstream adult population. This includes streets where cyclists have dedicated lanes and only have to interact with traffic at formal crossing.
- **LTS 3**: Tolerable for "enthused and confident" cyclists. This includes streets which may involve close proximity to moderate- or high-speed vehicular traffic.
- **LTS 4**: Tolerable only for "strong and fearless" cyclists. This includes streets where cyclists are required to mix with moderate- to high-speed vehicular traffic.

For advanced users, you can provide custom LTS values by adding a tag <key = "lts"> to the osm.pbf file.

Datetime parsing

r5r ignores the timezone attribute of datetime objects when parsing dates and times, using the study area's timezone instead. For example, let's say you are running some calculations using Rio de Janeiro, Brazil, as your study area. The datetime `as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S")` will be parsed as May 13th, 2019, 14:00h in Rio's local time, as expected. But `as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S", tz = "Europe/Paris")` will also be parsed as the exact same date and time in Rio's local time, perhaps surprisingly, ignoring the timezone attribute.

Routing algorithm

The `pareto_frontier()` function, and `detailed_itineraries()` with `shortest_path = FALSE` or a `fare_structure`, use an R5-specific extension to the McRAPTOR routing algorithm. The implementation used in `detailed_itineraries()` allows the router to find paths that are optimal and less than optimal in terms of travel time, with some heuristics around multiple access modes, riding the same patterns, etc. The specific extension to McRAPTOR to do suboptimal path routing is not documented yet, but a detailed description of base McRAPTOR can be found in Delling et al (2015). The implementation used in `pareto_frontier()`, on the other hand, returns only the fastest trip within a given monetary cutoff, ignoring slower trips that cost the same. A detailed discussion on the algorithm can be found in Conway and Stewart (2019).

- Delling, D., Pajor, T., & Werneck, R. F. (2015). Round-based public transit routing. Transportation Science, 49(3), 591-604. doi:10.1287/trsc.2014.0534
- Conway, M. W., & Stewart, A. F. (2019). Getting Charlie off the MTA: a multiobjective optimization method to account for cost constraints in public transit accessibility metrics. International Journal of Geographical Information Science, 33(9), 1759-1787. doi:10.1080/13658816.2019.1605075

Fastest itinerary

When only the fastest itinerary is returned (`shortest_path = TRUE`) and no `fare_structure` is used, public transport is routed with R5's RAPTOR algorithm (Conway et al, 2017), the one used by `travel_time_matrix()`, which is much faster than McRAPTOR. Compared with McRAPTOR, the results can differ slightly: departures are routed every minute of `time_window` instead of at random seconds, so the itinerary returned is never more than about a minute slower and often faster; walking transfers between stops are limited by `max_walk_time` (McRAPTOR only applies it to the first and last legs); stops where the GTFS `pickup_type` or `drop_off_type` does not allow boarding or alighting are respected; and when two trips are equally fast, the route chosen may differ.

- Conway, M. W., Byrd, A., & van der Linden, M. (2017). Evidence-based transit and land use sketch planning using interactive accessibility methods on combined schedule and headway-based networks. Transportation Research Record, 2653(1), 45-53. doi:10.3141/265306

See Also

Other routing: `arrival_travel_time_matrix()`, `expanded_travel_time_matrix()`, `pareto_frontier()`, `travel_time_matrix()`

Examples

```
library(r5r)

# build transport network
data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path)

# load origin/destination points
points <- read.csv(file.path(data_path, "poa_points_of_interest.csv"))

# inputs
departure_datetime <- as.POSIXct(
  "13-05-2019 14:00:00",
  format = "%d-%m-%Y %H:%M:%S"
)

det <- detailed_itineraries(
  r5r_network,
  origins = points[10,],
  destinations = points[12,],
  mode = c("WALK", "TRANSIT"),
  departure_datetime = departure_datetime,
  max_trip_duration = 60
)
head(det)

stop_r5(r5r_network)
```

---

download_r5 *Download* R5.jar

Description

Downloads R5.jar and saves it locally, inside the package directory.

Usage

```
download_r5(
  version = NULL,
  quiet = FALSE,
  force_update = FALSE,
  temp_dir = FALSE
)
```

Arguments

version	A string. The version of R5 to be downloaded. When NULL, it defaults to the latest version.

quiet	A logical. Whether to show informative messages when downloading the file. Defaults to FALSE.
force_update	A logical. Whether to overwrite a previously downloaded R5.jar in the local directory. Defaults to FALSE.
temp_dir	A logical. Whether the file should be saved in a temporary directory. Defaults to FALSE.

Value

The path to the downloaded file.

See Also

Other Build network: build_network(), setup_r5()

Examples

```
library(r5r)

download_r5(temp_dir = TRUE)
```

---

expanded_travel_time_matrix

Calculate minute-by-minute travel times between origin destination pairs

---

Description

Detailed computation of travel time estimates between one or multiple origin destination pairs. Results show the travel time of the fastest route alternative departing each minute within a specified time window. This function can be very memory intensive for large data sets and time windows. destinations can have at most 5000 rows, a limit of R5 for detailed path information; split larger sets into chunks.

Usage

```
expanded_travel_time_matrix(
  r5r_network,
  origins,
  destinations,
  mode = "WALK",
  mode_egress = "WALK",
  departure_datetime = NULL,
  time_window = 10L,
  breakdown = FALSE,
  max_walk_time = Inf,
```

```
  max_bike_time = Inf,
  max_car_time = Inf,
  max_trip_duration = 120L,
  walk_speed = 3.6,
  bike_speed = 12,
  max_rides = 3,
  max_lts = 2,
  new_carspeeds = NULL,
  carspeed_scale = 1,
  new_lts = NULL,
  draws_per_minute = 5L,
  seed = NULL,
  n_threads = Inf,
  verbose = FALSE,
  progress = TRUE,
  output_dir = NULL,
  r5r_core = deprecated()
)
```

Arguments

r5r_network	A routable transport network created with build_network().
origins, destinations	Either a POINT sf object with WGS84 CRS, or a data.frame containing the columns id, lon and lat.
mode	A character vector. The transport modes allowed for access, transfer and vehicle legs of the trips. Defaults to WALK. See details for other options.
mode_egress	A character vector. The transport mode used after egress from the last public transport. It can be either WALK, BICYCLE or CAR. Defaults to WALK. Ignored when public transport is not used.
departure_datetime	A POSIXct object. Only affects public transport legs; when routing with public transport, it must fall within the service period of the GTFS feeds (calendar.txt; see check_transit_availability()). Required for public transport; optional otherwise. See details for how datetimes are parsed.
time_window	An integer. The time window in minutes for which r5r will calculate multiple travel time matrices departing each minute. Defaults to 10 minutes. The output has one row per departure minute (and per Monte Carlo draw, see draws_per_minute). See vignette("time_window", package = "r5r").
breakdown	A logical. Whether to include detailed information about each trip in the output. If FALSE (the default), the output lists the total time between each origin-destination pair and the routes used to complete the trip for each minute of the specified time window. If TRUE, the output also includes the access, waiting, in-vehicle, transfer and egress time of each trip, and its number of public transport rides. For trips made only by walking, cycling or driving (routes equal to [WALK], [BICYCLE] or [CAR]), these components are all 0 and total_time holds the whole trip.

max_walk_time An integer. The maximum walking time (in minutes) to access and egress the transit network, make transfers, or complete walk-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In walk-only trips, the lower of max_walk_time and max_trip_duration applies.

max_bike_time An integer. The maximum cycling time (in minutes) to access and egress the transit network, make transfers, or complete bicycle-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In bicycle-only trips, the lower of max_bike_time and max_trip_duration applies.

max_car_time An integer. The maximum driving time (in minutes) to access and egress the transit network, or to complete car-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In car-only trips, the lower of max_car_time and max_trip_duration applies.

max_trip_duration
An integer. The maximum trip duration in minutes. Defaults to 120.

walk_speed A numeric. Average walk speed in km/h. Defaults to 3.6.

bike_speed A numeric. Average cycling speed in km/h. Defaults to 12.

max_rides An integer. The maximum number of public transport rides allowed in the same trip. Defaults to 3.

max_lts An integer between 1 and 4. The maximum level of traffic stress that cyclists will tolerate. A value of 1 means cyclists will only travel through the quietest streets, while a value of 4 indicates cyclists can travel through any road. Defaults to 2. See details.

new_carspeeds A data.frame specifying the new car speed for each OSM edge id. This table must contain columns osm_id, max_speed and speed_type. The "speed_type" column is of class character and it indicates whether the values in "max_speed" should be interpreted as percentages of original speeds ("scale") or as absolute speeds ("km/h"). Alternatively, the new_carspeeds parameter can receive an sf data.frame with POLYGON geometry that indicates the new car speed for all the roads that fall within each polygon. In this case, the table must contain the columns poly_id with a unique id for each polygon, scale with the new speed scaling factors and priority, which is a number ranking which polygon should be considered in case of overlapping polygons. See more info in the scenarios vignette (vignette("scenarios", package = "r5r")).

carspeed_scale Numeric. The scaling factor applied to the car speed of road segments not specified in new_carspeeds. Defaults to 1, which keeps the speeds of the unlisted roads unchanged.

new_lts A data.frame specifying the new LTS levels for each OSM edge id. The table must contain columns osm_id and lts. Alternatively, the new_lts parameter can receive an sf data.frame with LINESTRING geometry. R5 will then find the nearest road for each LINESTRING and update its LTS value accordingly.

draws_per_minute
: An integer. The number of Monte Carlo draws to perform per minute of time_window. Defaults to 5. This would mean 300 draws in a 60-minute time window, for example. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a frequencies.txt table; otherwise, r5r still routes a departure every minute of time_window, in a deterministic way. In functions that accept a fare_structure, passing one switches to the McRaptor algorithm, which instead routes time_window * draws_per_minute departure times sampled at random within the window (different for each origin, and the same in every call from the same origin with the same seed). This parameter then affects the results with or without frequencies. R5's McRaptor cannot sample many departure times per minute over long windows, so such combinations raise an error (e.g. 8 draws per minute in a 10-minute window, or 5 in a window of 150 minutes or more), as does a total of exactly 200 draws.

seed
: An integer or NULL. The seed of the Monte Carlo draws of the departure offsets of frequency-based transit routes and, with the McRaptor algorithm, of the departure times sampled within time_window. Defaults to NULL, in which case the draws are random and results may vary slightly between runs. With an integer, the same inputs always give the same results, regardless of n_threads. Results are still a Monte Carlo sample, so a different seed may give slightly different results: report the seed used and check how sensitive the results are to it. Note that set.seed() has no effect on these draws. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a frequencies.txt table. In functions that accept a fare_structure, passing one switches to the McRaptor algorithm, and seed then affects the results with or without frequencies.

n_threads
: An integer. The number of threads to use when running the router in parallel. Defaults to Inf (all available threads).

verbose
: A logical. Whether to show R5 informative messages when running the function. Defaults to FALSE (R5 error messages are still shown). TRUE shows detailed output, useful for debugging issues not caught by r5r.

progress
: A logical. Whether to show a progress counter when running the router. Defaults to TRUE. Only works when verbose is FALSE. May slightly slow computation, as the counter is synchronized across threads.

output_dir
: Either NULL (the default) or a path to an existing directory. When a path is given, the function writes the results as .csv files to that directory and returns the path instead of the results. Useful in memory-constrained settings, as results are not loaded into RAM. Missing values (NA) are written as empty fields.

r5r_core
: The r5r_core argument is deprecated as of r5r v2.3.0. Use the r5r_network argument instead.

Value

A data.table with travel time estimates (in minutes) and the routes used in each trip between origin and destination pairs, for each minute of the specified time window. Each set of origin, destination and departure minute can appear up to N times, where N is the number of Monte Carlo draws set by draws_per_minute (this only applies when the GTFS feeds include a frequencies

table; otherwise a single draw is performed). A pair is completely absent from the final output if no trips could be completed in any of the minutes of the time window. If for a single pair trips could be completed in some of the minutes of the time window, but not for all of them, the minutes in which trips couldn't be completed will have NA travel time and routes used. A pair may also appear with NA in every minute, when its only trips are slightly longer than max_trip_duration (e.g. walk-only trips, whose time to walk between the point and the street network is added after the limit is applied).

The output has the columns from_id, to_id, departure_time, draw_number, routes and total_time, plus access_time, wait_time, ride_time, transfer_time, egress_time and n_rides when breakdown = TRUE. routes lists the public transport routes used, separated by |, or the street mode (e.g. [WALK]) for trips that do not use public transport. draw_number only labels the draws of a departure minute; their order is arbitrary.

Travel times are reported with a precision of 0.1 minute, and trips longer than max_trip_duration get NA. travel_time_matrix() rounds travel times down to whole minutes, so it can report trips that are up to 0.9 minute over max_trip_duration which are NA here. With walk-, bike- or car-only mode, travel times are whole minutes.

If output_dir is not NULL, the function returns the path specified in that parameter, in which the .csv files containing the results are saved.

Transport modes

R5 allows for multiple combinations of transport modes. The options include:

- **Transit modes:** TRAM, SUBWAY, RAIL, BUS, FERRY, CABLE_CAR, GONDOLA, FUNICULAR. The option TRANSIT automatically considers all public transport modes available.
- **Non transit modes:** WALK, BICYCLE, CAR.

Level of Traffic Stress (LTS)

When cycling is enabled in R5 (by passing the value BICYCLE to either mode or mode_egress), setting max_lts will allow cycling only on streets with a given level of danger/stress. Setting max_lts to 1, for example, will allow cycling only on separated bicycle infrastructure or low-traffic streets and routing will revert to walking when traversing any links with LTS exceeding 1. Setting max_lts to 3 will allow cycling on links with LTS 1, 2 or 3. Routing also reverts to walking if the street segment is tagged as non-bikable in OSM (e.g. a staircase), independently of the specified max LTS.

The default methodology for assigning LTS values to network edges is based on commonly tagged attributes of OSM ways. See more info about LTS in the original documentation of R5 from Conveyal at https://docs.conveyal.com/learn-more/traffic-stress. In summary:

- **LTS 1**: Tolerable for children. This includes low-speed, low-volume streets, as well as those with separated bicycle facilities (such as parking-protected lanes or cycle tracks).
- **LTS 2**: Tolerable for the mainstream adult population. This includes streets where cyclists have dedicated lanes and only have to interact with traffic at formal crossing.
- **LTS 3**: Tolerable for "enthused and confident" cyclists. This includes streets which may involve close proximity to moderate- or high-speed vehicular traffic.
- **LTS 4**: Tolerable only for "strong and fearless" cyclists. This includes streets where cyclists are required to mix with moderate- to high-speed vehicular traffic.

For advanced users, you can provide custom LTS values by adding a tag `<key = "lts">` to the `osm.pbf` file.

Datetime parsing

`r5r` ignores the timezone attribute of datetime objects when parsing dates and times, using the study area's timezone instead. For example, let's say you are running some calculations using Rio de Janeiro, Brazil, as your study area. The datetime `as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S")` will be parsed as May 13th, 2019, 14:00h in Rio's local time, as expected. But `as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S", tz = "Europe/Paris")` will also be parsed as the exact same date and time in Rio's local time, perhaps surprisingly, ignoring the timezone attribute.

Routing algorithm

The `travel_time_matrix()`, `expanded_travel_time_matrix()`, `arrival_travel_time_matrix()` and `accessibility()` functions use an R5-specific extension to the RAPTOR routing algorithm (see Conway et al., 2017). This RAPTOR extension uses a systematic sample of one departure per minute over the time window set by the user in the 'time_window' parameter. A detailed description of base RAPTOR can be found in Delling et al (2015). However, whenever the user includes transit fares inputs to these functions, they automatically switch to use an R5-specific extension to the McRAPTOR routing algorithm. `detailed_itineraries()` also uses RAPTOR when it only returns the fastest itinerary (`shortest_path = TRUE`) and no fares are used.

- Conway, M. W., Byrd, A., & van der Linden, M. (2017). Evidence-based transit and land use sketch planning using interactive accessibility methods on combined schedule and headway-based networks. Transportation Research Record, 2653(1), 45-53. doi:10.3141/265306
- Delling, D., Pajor, T., & Werneck, R. F. (2015). Round-based public transit routing. Transportation Science, 49(3), 591-604. doi:10.1287/trsc.2014.0534

See Also

Other routing: `arrival_travel_time_matrix()`, `detailed_itineraries()`, `pareto_frontier()`, `travel_time_matrix()`

Examples

```
library(r5r)

# build transport network
data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path)

# load origin/destination points
points <- read.csv(file.path(data_path, "poa_points_of_interest.csv"))

departure_datetime <- as.POSIXct(
  "13-05-2019 14:00:00",
  format = "%d-%m-%Y %H:%M:%S"
)
```

```
# by default only returns the total time between each pair in each minute of
# the specified time window
ettm <- expanded_travel_time_matrix(
  r5r_network,
  origins = points,
  destinations = points,
  mode = c("WALK", "TRANSIT"),
  time_window = 20,
  departure_datetime = departure_datetime,
  max_trip_duration = 60
)
head(ettm)

# when breakdown = TRUE the output contains much more information
ettm <- expanded_travel_time_matrix(
  r5r_network,
  origins = points,
  destinations = points,
  mode = c("WALK", "TRANSIT"),
  time_window = 20,
  departure_datetime = departure_datetime,
  max_trip_duration = 60,
  breakdown = TRUE
)
head(ettm)

stop_r5(r5r_network)
```

`find_snap` *Find snapped locations of input points on street network*

Description

Finds the snapped location of points on R5 network. Snapping is an important step of the routing process, which is when the origins and destinations specified by the user are actually positioned on the network created by R5. The snapping process in R5 is composed of two rounds. First, it tries to snap the points within a radius of 300 meters from themselves. If the first round is unsuccessful, then R5 expands the search to the radius specified (by default 1.6km). If yet again it is unsuccessful, then the unsnapped points won't be used during the routing process. The snapped location of each point depends on the transport mode set by the user, because some network edges are not available to specific modes (e.g. a pedestrian-only street cannot be used to snap car trips).

Usage

```
find_snap(
  r5r_network,
  points,
  radius = 1600,
```

```
  mode = "WALK",
  r5r_core = deprecated()
)
```

Arguments

r5r_network	A routable transport network created with build_network().
points	Either a POINT sf object with WGS84 CRS, or a data.frame containing the columns id, lon and lat.
radius	Numeric. The maximum radius in meters within which to snap. Defaults to 1600m.
mode	A string. Which mode to consider when trying to snap the points to the network. Defaults to WALK, also allows BICYCLE and CAR.
r5r_core	The r5r_core argument is deprecated as of r5r v2.3.0. Use the r5r_network argument instead.

Value

A data.table with the original points, their respective snapped coordinates on the street network and the Euclidean distance (in meters) between the original points and their snapped location. Points that could not be snapped show NA coordinates and found = FALSE.

See Also

Other network functions: street_network_bbox(), street_network_to_sf(), transit_network_to_sf()

Examples

```
library(r5r)

path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path = path)
points <- read.csv(file.path(path, "poa_hexgrid.csv"))

snap_df <- find_snap(
  r5r_network,
  points = points,
  radius = 2000,
  mode = "WALK"
  )

stop_r5(r5r_network)
```

`get_gtfs_errors` *Get GTFS errors encountered in network building*

Description

Returns a data frame of the GTFS errors R5 encountered when building the network. If the build failed and there is no network object, pass the path where the network is stored as `r5r_network` instead.

Usage

```
get_gtfs_errors(r5r_network)
```

Arguments

`r5r_network`	the R5R network object, or a path to the location where the network is stored (useful if network build failed).

Value

A `data.frame`

See Also

Other support functions: `exists_tiff()`, `fileurl_from_metadata()`, `start_r5r_java()`, `stop_r5()`, `tempdir_unique()`, `travel_time_surface()`, `validate_bad_osm_ids()`

Examples

```
library(r5r)

# directory with street network and gtfs files
data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path)

get_gtfs_errors(r5r_network)
```

isochrone *Estimate isochrones from a given location*

Description

Fast computation of isochrones from a given location. The function can return either polygon-based or line-based isochrones. Polygon-based isochrones are generated from a travel time surface: travel times from each origin to the centres of a regular grid of Web Mercator pixels (see `zoom`), from which the isochrone polygons are interpolated with the marching squares algorithm. Line-based isochrones are based on travel times from each origin to the centroids of all segments in the transport network.

Usage

```
isochrone(
  r5r_network,
  origins,
  mode = "transit",
  mode_egress = "walk",
  cutoffs = c(0, 15, 30),
  zoom = 10,
  departure_datetime = NULL,
  polygon_output = TRUE,
  time_window = 10L,
  max_walk_time = Inf,
  max_bike_time = Inf,
  max_car_time = Inf,
  max_trip_duration = 120L,
  walk_speed = 3.6,
  bike_speed = 12,
  max_rides = 3,
  max_lts = 2,
  draws_per_minute = 5L,
  seed = NULL,
  percentiles = NULL,
  n_threads = Inf,
  verbose = FALSE,
  progress = TRUE,
  sample_size = deprecated(),
  r5r_core = deprecated()
)
```

Arguments

`r5r_network`	A routable transport network created with `build_network()`.
`origins`	Either a `POINT sf` object with WGS84 CRS, or a `data.frame` containing the columns `id`, `lon` and `lat`.

mode — A character vector. The transport modes allowed for access, transfer and vehicle legs of the trips. Defaults to TRANSIT. See details for other options.

mode_egress — A character vector. The transport mode used after egress from the last public transport. It can be either WALK, BICYCLE or CAR. Defaults to WALK. Ignored when public transport is not used.

cutoffs — A numeric vector. The travel times, in minutes, that delimit the isochrones. Defaults to c(0, 15, 30). Values are sorted and duplicates are removed; at least one value must be greater than 0.

zoom — A number between 9 and 12. The Web Mercator zoom level of the travel time grid from which polygon isochrones are interpolated (only used when polygon_output = TRUE). Higher values give more detailed isochrones but take longer to compute. Defaults to 10 (cells of about 153 meters at the Equator). For how grid cells are defined, see the R5 documentation.

departure_datetime — A POSIXct object. Only affects public transport legs; when routing with public transport, it must fall within the service period of the GTFS feeds (calendar.txt; see check_transit_availability()). Required for public transport; optional otherwise. See details for how datetimes are parsed.

polygon_output — A Logical. If TRUE, the function outputs polygon-based isochrones (the default) based on travel times from each origin to a regular grid of points (see parameter zoom). If FALSE, the function outputs line-based isochrones based on travel times from each origin to the centroids of all segments in the transport network.

time_window — An integer. The time window in minutes. Departures are simulated every minute from departure_datetime until time_window minutes later, and travel times are summarized over these departures using percentiles (the median by default). With a fare_structure, and always in pareto_frontier(), departures are instead sampled at random within the window: draws_per_minute per minute on average (one in pareto_frontier()), different for each origin and the same in every call from the same origin with the same seed. Defaults to 10. See vignette("time_window", package = "r5r").

max_walk_time — An integer. The maximum walking time (in minutes) to access and egress the transit network, make transfers, or complete walk-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In walk-only trips, the lower of max_walk_time and max_trip_duration applies.

max_bike_time — An integer. The maximum cycling time (in minutes) to access and egress the transit network, make transfers, or complete bicycle-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In bicycle-only trips, the lower of max_bike_time and max_trip_duration applies.

max_car_time — An integer. The maximum driving time (in minutes) to access and egress the transit network, or to complete car-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In car-only trips, the lower of max_car_time and max_trip_duration applies.

max_trip_duration
: Ignored. The maximum trip duration is set internally from `max(cutoffs)`.

walk_speed
: A numeric. Average walk speed in km/h. Defaults to 3.6.

bike_speed
: A numeric. Average cycling speed in km/h. Defaults to 12.

max_rides
: An integer. The maximum number of public transport rides allowed in the same trip. Defaults to 3.

max_lts
: An integer between 1 and 4. The maximum level of traffic stress that cyclists will tolerate. A value of 1 means cyclists will only travel through the quietest streets, while a value of 4 indicates cyclists can travel through any road. Defaults to 2. See details.

draws_per_minute
: An integer. The number of Monte Carlo draws to perform per minute of `time_window`. Defaults to 5. This would mean 300 draws in a 60-minute time window, for example. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a `frequencies.txt` table; otherwise, r5r still routes a departure every minute of `time_window`, in a deterministic way. In functions that accept a `fare_structure`, passing one switches to the McRaptor algorithm, which instead routes `time_window * draws_per_minute` departure times sampled at random within the window (different for each origin, and the same in every call from the same origin with the same `seed`). This parameter then affects the results with or without frequencies. R5's McRaptor cannot sample many departure times per minute over long windows, so such combinations raise an error (e.g. 8 draws per minute in a 10-minute window, or 5 in a window of 150 minutes or more), as does a total of exactly 200 draws.

seed
: An integer or `NULL`. The seed of the Monte Carlo draws of the departure offsets of frequency-based transit routes and, with the McRaptor algorithm, of the departure times sampled within `time_window`. Defaults to `NULL`, in which case the draws are random and results may vary slightly between runs. With an integer, the same inputs always give the same results, regardless of `n_threads`. Results are still a Monte Carlo sample, so a different seed may give slightly different results: report the seed used and check how sensitive the results are to it. Note that `set.seed()` has no effect on these draws. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a `frequencies.txt` table. In functions that accept a `fare_structure`, passing one switches to the McRaptor algorithm, and `seed` then affects the results with or without frequencies.

percentiles
: An integer vector (max length of 5). The travel time percentiles within `time_window` used to build the isochrones, one set of polygons per percentile. Defaults to 50 (the median travel time). Only used when `polygon_output = TRUE`; must be `NULL` for line-based isochrones.

n_threads
: An integer. The number of threads to use when running the router in parallel. Defaults to `Inf` (all available threads).

verbose
: A logical. Whether to show `R5` informative messages when running the function. Defaults to `FALSE` (`R5` error messages are still shown). `TRUE` shows detailed output, useful for debugging issues not caught by `r5r`.

progress	A logical. Whether to show a progress counter when running the router. Defaults to TRUE. Only works when verbose is FALSE. May slightly slow computation, as the counter is synchronized across threads.
sample_size	deprecated, no longer has any effect.
r5r_core	The r5r_core argument is deprecated as of r5r v2.3.0. Use the r5r_network argument instead.

Value

A "sf" "data.frame". With polygon_output = TRUE, one POLYGON or MULTIPOLYGON per origin, percentile and cutoff, with columns id (origin id), isochrone (cutoff in minutes), percentile (a string such as "p50") and polygons. Each polygon covers the whole area reached from 0 up to its cutoff, so polygons of larger cutoffs contain those of smaller ones. With polygon_output = FALSE, one LINESTRING per street segment reached, with columns id (origin id), edge_index, osm_id, isochrone (the smallest cutoff at or above the segment's travel time, i.e. bands are intervals), travel_time_p50 and geometry.

Transport modes

R5 allows for multiple combinations of transport modes. The options include:

- **Transit modes:** TRAM, SUBWAY, RAIL, BUS, FERRY, CABLE_CAR, GONDOLA, FUNICULAR. The option TRANSIT automatically considers all public transport modes available.
- **Non transit modes:** WALK, BICYCLE, CAR.

Level of Traffic Stress (LTS)

When cycling is enabled in R5 (by passing the value BICYCLE to either mode or mode_egress), setting max_lts will allow cycling only on streets with a given level of danger/stress. Setting max_lts to 1, for example, will allow cycling only on separated bicycle infrastructure or low-traffic streets and routing will revert to walking when traversing any links with LTS exceeding 1. Setting max_lts to 3 will allow cycling on links with LTS 1, 2 or 3. Routing also reverts to walking if the street segment is tagged as non-bikable in OSM (e.g. a staircase), independently of the specified max LTS.

The default methodology for assigning LTS values to network edges is based on commonly tagged attributes of OSM ways. See more info about LTS in the original documentation of R5 from Conveyal at https://docs.conveyal.com/learn-more/traffic-stress. In summary:

- **LTS 1**: Tolerable for children. This includes low-speed, low-volume streets, as well as those with separated bicycle facilities (such as parking-protected lanes or cycle tracks).
- **LTS 2**: Tolerable for the mainstream adult population. This includes streets where cyclists have dedicated lanes and only have to interact with traffic at formal crossing.
- **LTS 3**: Tolerable for "enthused and confident" cyclists. This includes streets which may involve close proximity to moderate- or high-speed vehicular traffic.
- **LTS 4**: Tolerable only for "strong and fearless" cyclists. This includes streets where cyclists are required to mix with moderate- to high-speed vehicular traffic.

For advanced users, you can provide custom LTS values by adding a tag <key = "lts"> to the osm.pbf file.

Datetime parsing

r5r ignores the timezone attribute of datetime objects when parsing dates and times, using the study area's timezone instead. For example, let's say you are running some calculations using Rio de Janeiro, Brazil, as your study area. The datetime `as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S")` will be parsed as May 13th, 2019, 14:00h in Rio's local time, as expected. But `as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S", tz = "Europe/Paris")` will also be parsed as the exact same date and time in Rio's local time, perhaps surprisingly, ignoring the timezone attribute.

Routing algorithm

The travel_time_matrix(), expanded_travel_time_matrix(), arrival_travel_time_matrix() and accessibility() functions use an R5-specific extension to the RAPTOR routing algorithm (see Conway et al., 2017). This RAPTOR extension uses a systematic sample of one departure per minute over the time window set by the user in the 'time_window' parameter. A detailed description of base RAPTOR can be found in Delling et al (2015). However, whenever the user includes transit fares inputs to these functions, they automatically switch to use an R5-specific extension to the McRAPTOR routing algorithm. detailed_itineraries() also uses RAPTOR when it only returns the fastest itinerary (shortest_path = TRUE) and no fares are used.

- Conway, M. W., Byrd, A., & van der Linden, M. (2017). Evidence-based transit and land use sketch planning using interactive accessibility methods on combined schedule and headway-based networks. Transportation Research Record, 2653(1), 45-53. doi:10.3141/265306
- Delling, D., Pajor, T., & Werneck, R. F. (2015). Round-based public transit routing. Transportation Science, 49(3), 591-604. doi:10.1287/trsc.2014.0534

Examples

```
old_options <- options(java.parameters = "-Xmx2G")
library(r5r)
library(ggplot2)

# build transport network
data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path = data_path)

# load origin/point of interest
points <- read.csv(file.path(data_path, "poa_points_of_interest.csv"))
origin <- points[2,]

departure_datetime <- as.POSIXct(
 "13-05-2019 14:00:00",
 format = "%d-%m-%Y %H:%M:%S"
)

# estimate polygon-based isochrone from origin
iso_poly <- isochrone(
  r5r_network,
  origins = origin,
  mode = "walk",
```

```
  polygon_output = TRUE,
  departure_datetime = departure_datetime,
  cutoffs = seq(0, 120, 30)
  )

head(iso_poly)


# estimate line-based isochrone from origin
iso_lines <- isochrone(
  r5r_network,
  origins = origin,
  mode = "walk",
  polygon_output = FALSE,
  departure_datetime = departure_datetime,
  cutoffs = seq(0, 100, 25)
  )

head(iso_lines)


# plot colors
colors <- c('#ffe0a5','#ffcb69','#ffa600','#ff7c43','#f95d6a',
            '#d45087','#a05195','#665191','#2f4b7c','#003f5c')

# polygons
ggplot() +
  geom_sf(data=iso_poly, aes(fill=factor(isochrone))) +
  scale_fill_manual(values = colors) +
  theme_minimal()

# lines
ggplot() +
  geom_sf(data=iso_lines, aes(color=factor(isochrone))) +
  scale_color_manual(values = colors) +
  theme_minimal()

stop_r5(r5r_network)
options(old_options)
```

`pareto_frontier` *Calculate travel time and monetary cost Pareto frontier*

Description

Fast computation of travel time and monetary cost Pareto frontier between origin and destination pairs.

Usage

```
pareto_frontier(
  r5r_network,
  origins,
  destinations,
  mode = c("WALK", "TRANSIT"),
  mode_egress = "WALK",
  departure_datetime = NULL,
  time_window = 10L,
  seed = NULL,
  percentiles = 50L,
  max_walk_time = Inf,
  max_bike_time = Inf,
  max_car_time = Inf,
  max_trip_duration = 120L,
  fare_structure,
  fare_cutoffs,
  walk_speed = 3.6,
  bike_speed = 12,
  max_rides = 3,
  max_lts = 2,
  n_threads = Inf,
  verbose = FALSE,
  progress = TRUE,
  output_dir = NULL,
  r5r_core = deprecated()
)
```

Arguments

r5r_network	A routable transport network created with build_network().
origins, destinations	Either a POINT sf object with WGS84 CRS, or a data.frame containing the columns id, lon and lat.
mode	A character vector. The transport modes allowed for access, transfer and vehicle legs of the trips. Defaults to WALK. See details for other options.
mode_egress	A character vector. The transport mode used after egress from the last public transport. It can be either WALK, BICYCLE or CAR. Defaults to WALK. Ignored when public transport is not used.
departure_datetime	A POSIXct object. Only affects public transport legs; when routing with public transport, it must fall within the service period of the GTFS feeds (calendar.txt; see check_transit_availability()). Required for public transport; optional otherwise. See details for how datetimes are parsed.
time_window	An integer. The time window in minutes. Departures are simulated every minute from departure_datetime until time_window minutes later, and travel times

are summarized over these departures using `percentiles` (the median by default). With a `fare_structure`, and always in `pareto_frontier()`, departures are instead sampled at random within the window: `draws_per_minute` per minute on average (one in `pareto_frontier()`), different for each origin and the same in every call from the same origin with the same `seed`. Defaults to 10. See `vignette("time_window", package = "r5r")`.

`seed` An integer or `NULL`. The seed of the Monte Carlo draws of the departure offsets of frequency-based transit routes and, with the McRaptor algorithm, of the departure times sampled within `time_window`. Defaults to `NULL`, in which case the draws are random and results may vary slightly between runs. With an integer, the same inputs always give the same results, regardless of `n_threads`. Results are still a Monte Carlo sample, so a different seed may give slightly different results: report the seed used and check how sensitive the results are to it. Note that `set.seed()` has no effect on these draws. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a `frequencies.txt` table. This function always uses the McRaptor algorithm, so `seed` affects the results with or without frequencies.

`percentiles` An integer vector (max length of 5). Specifies the percentile to use when returning travel time estimates within the given time window. This parameter is applied to the travel time estimates only (e.g. if the 25th percentile is specified, and the output between A and B is 15 minutes and 10 dollars, 25% of all trips cheaper than 10 dollars taken between these points are shorter than 15 minutes). Defaults to 50, returning the median travel time. If a vector with length bigger than 1 is passed, the output contains an additional column that specifies the percentile of each travel time and monetary cost combination. Due to upstream restrictions, only 5 percentiles can be specified at a time. See the R5 documentation at https://docs.conveyal.com/analysis/methodology#accounting-for-variability.

`max_walk_time` An integer. The maximum walking time (in minutes) to access and egress the transit network, make transfers, or complete walk-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to `Inf` (no limit besides `max_trip_duration`). In walk-only trips, the lower of `max_walk_time` and `max_trip_duration` applies.

`max_bike_time` An integer. The maximum cycling time (in minutes) to access and egress the transit network, make transfers, or complete bicycle-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to `Inf` (no limit besides `max_trip_duration`). In bicycle-only trips, the lower of `max_bike_time` and `max_trip_duration` applies.

`max_car_time` An integer. The maximum driving time (in minutes) to access and egress the transit network, or to complete car-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to `Inf` (no limit besides `max_trip_duration`). In car-only trips, the lower of `max_car_time` and `max_trip_duration` applies.

`max_trip_duration`
An integer. The maximum trip duration in minutes. Defaults to 120.

fare_structure	A fare structure object, following the convention set in setup_fare_structure(). This object describes how transit fares should be calculated. See vignette("fare_structure", package = "r5r") for its structure. Required: without it there are no fares to trade off against travel time.
fare_cutoffs	A numeric vector, required. The monetary cutoffs (each greater than or equal to 0) used to build the Pareto frontier. Usually these are all possible fares in your fare_structure, including sums of fares for trips with transfers. Coarse cutoffs merge different trips: with routes costing $3 and $4 and a single cutoff of 5, the output shows the fastest trips costing up to $5 but not which route they used; c(3, 4) separates them.
walk_speed	A numeric. Average walk speed in km/h. Defaults to 3.6.
bike_speed	A numeric. Average cycling speed in km/h. Defaults to 12.
max_rides	An integer. The maximum number of public transport rides allowed in the same trip. Defaults to 3.
max_lts	An integer between 1 and 4. The maximum level of traffic stress that cyclists will tolerate. A value of 1 means cyclists will only travel through the quietest streets, while a value of 4 indicates cyclists can travel through any road. Defaults to 2. See details.
n_threads	An integer. The number of threads to use when running the router in parallel. Defaults to Inf (all available threads).
verbose	A logical. Whether to show R5 informative messages when running the function. Defaults to FALSE (R5 error messages are still shown). TRUE shows detailed output, useful for debugging issues not caught by r5r.
progress	A logical. Whether to show a progress counter when running the router. Defaults to TRUE. Only works when verbose is FALSE. May slightly slow computation, as the counter is synchronized across threads.
output_dir	Either NULL (the default) or a path to an existing directory. When a path is given, the function writes the results as .csv files to that directory and returns the path instead of the results. Useful in memory-constrained settings, as results are not loaded into RAM. Missing values (NA) are written as empty fields.
r5r_core	The r5r_core argument is deprecated as of r5r v2.3.0. Use the r5r_network argument instead.

Value

A data.table with the travel time and monetary cost Pareto frontier between the specified origins and destinations. An additional column identifying the travel time percentile is present if more than one value was passed to percentiles. Origin and destination pairs whose trips couldn't be completed within the maximum travel time using less money than the specified monetary cutoffs are not returned in the data.table. If output_dir is not NULL, the function returns the path specified in that parameter, in which the .csv files containing the results are saved.

Transport modes

R5 allows for multiple combinations of transport modes. The options include:

- **Transit modes:** TRAM, SUBWAY, RAIL, BUS, FERRY, CABLE_CAR, GONDOLA, FUNICULAR. The option TRANSIT automatically considers all public transport modes available.
- **Non transit modes:** WALK, BICYCLE, CAR.

Level of Traffic Stress (LTS)

When cycling is enabled in R5 (by passing the value BICYCLE to either mode or mode_egress), setting max_lts will allow cycling only on streets with a given level of danger/stress. Setting max_lts to 1, for example, will allow cycling only on separated bicycle infrastructure or low-traffic streets and routing will revert to walking when traversing any links with LTS exceeding 1. Setting max_lts to 3 will allow cycling on links with LTS 1, 2 or 3. Routing also reverts to walking if the street segment is tagged as non-bikable in OSM (e.g. a staircase), independently of the specified max LTS.

The default methodology for assigning LTS values to network edges is based on commonly tagged attributes of OSM ways. See more info about LTS in the original documentation of R5 from Conveyal at https://docs.conveyal.com/learn-more/traffic-stress. In summary:

- **LTS 1**: Tolerable for children. This includes low-speed, low-volume streets, as well as those with separated bicycle facilities (such as parking-protected lanes or cycle tracks).
- **LTS 2**: Tolerable for the mainstream adult population. This includes streets where cyclists have dedicated lanes and only have to interact with traffic at formal crossing.
- **LTS 3**: Tolerable for "enthused and confident" cyclists. This includes streets which may involve close proximity to moderate- or high-speed vehicular traffic.
- **LTS 4**: Tolerable only for "strong and fearless" cyclists. This includes streets where cyclists are required to mix with moderate- to high-speed vehicular traffic.

For advanced users, you can provide custom LTS values by adding a tag <key = "lts"> to the osm.pbf file.

Datetime parsing

r5r ignores the timezone attribute of datetime objects when parsing dates and times, using the study area's timezone instead. For example, let's say you are running some calculations using Rio de Janeiro, Brazil, as your study area. The datetime as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S") will be parsed as May 13th, 2019, 14:00h in Rio's local time, as expected. But as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S", tz = "Europe/Paris") will also be parsed as the exact same date and time in Rio's local time, perhaps surprisingly, ignoring the timezone attribute.

Routing algorithm

The pareto_frontier() function, and detailed_itineraries() with shortest_path = FALSE or a fare_structure, use an R5-specific extension to the McRAPTOR routing algorithm. The implementation used in detailed_itineraries() allows the router to find paths that are optimal and less than optimal in terms of travel time, with some heuristics around multiple access modes, riding the same patterns, etc. The specific extension to McRAPTOR to do suboptimal path routing is not documented yet, but a detailed description of base McRAPTOR can be found in Delling et al (2015). The implementation used in pareto_frontier(), on the other hand, returns only the

fastest trip within a given monetary cutoff, ignoring slower trips that cost the same. A detailed discussion on the algorithm can be found in Conway and Stewart (2019).

- Delling, D., Pajor, T., & Werneck, R. F. (2015). Round-based public transit routing. Transportation Science, 49(3), 591-604. doi:10.1287/trsc.2014.0534
- Conway, M. W., & Stewart, A. F. (2019). Getting Charlie off the MTA: a multiobjective optimization method to account for cost constraints in public transit accessibility metrics. International Journal of Geographical Information Science, 33(9), 1759-1787. doi:10.1080/13658816.2019.1605075

See Also

Other routing: `arrival_travel_time_matrix()`, `detailed_itineraries()`, `expanded_travel_time_matrix()`, `travel_time_matrix()`

Examples

```
library(r5r)

# build transport network
data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path = data_path)

# load origin/destination points
points <- read.csv(file.path(data_path, "poa_hexgrid.csv"))[1:5,]

# load fare structure object
fare_structure_path <- system.file(
  "extdata/poa/fares/fares_poa.zip",
  package = "r5r"
)
fare_structure <- read_fare_structure(fare_structure_path)

departure_datetime <- as.POSIXct(
  "13-05-2019 14:00:00",
  format = "%d-%m-%Y %H:%M:%S"
)

pf <- pareto_frontier(
  r5r_network,
  origins = points,
  destinations = points,
  mode = c("WALK", "TRANSIT"),
  departure_datetime = departure_datetime,
  fare_structure = fare_structure,
  fare_cutoffs = c(4.5, 4.8, 9, 9.3, 9.6)
)
head(pf)

stop_r5(r5r_network)
```

r5r_cache *Manage cached files from the r5r package*

Description

Manage cached files from the r5r package

Usage

```
r5r_cache(list_files = TRUE, delete_file = NULL)
```

Arguments

list_files	Logical. Whether to print a message with the address of r5r JAR files cached locally. Defaults to TRUE.
delete_file	String. The file name (basename) of a JAR file cached locally that should be deleted. Defaults to NULL, so that no file is deleted. If delete_file = "all", then all cached files are deleted.

Value

A message indicating which files exist and/or which ones have been deleted from local cache directory.

Examples

```
# download r5 JAR
r5r::download_r5()

# list all files cached
r5r_cache(list_files = TRUE)

# delete r5 JAR
r5r_cache(delete_file = 'r5-v7.0')
```

r5r_sitrep *Generate an r5r situation report to help debug errors*

Description

The function reports a list with the following information:

- The package version of {r5r} in use.
- The installed version of R5.jar.
- The Java version in use.
- The amount of memory set to Java through the java.parameters option.
- The user's Session Info.

Usage

```
r5r_sitrep()
```

Value

A `list` with information of the versions of the r5r package, Java and R5 Jar in use, the memory set to Java and user's Session Info.

Examples

```
r5r_sitrep()
```

---

read_fare_structure *Read a fare structure object from a file*

Description

Read a fare structure object from a file

Usage

```
read_fare_structure(file_path, encoding = "UTF-8")
```

Arguments

`file_path`	A path pointing to a fare structure with a `.zip` extension.
`encoding`	A string. Passed to `data.table::fread()`, defaults to `"UTF-8"`. Other possible options are `"unknown"` and `"Latin-1"`. This is not used to re-encode the input, but to enable handling encoded strings in their native encoding.

Value

A fare structure object.

See Also

Other fare structure: `setup_fare_structure()`, `write_fare_structure()`

Examples

```
path <- system.file("extdata/poa/fares/fares_poa.zip", package = "r5r")
fare_structure <- read_fare_structure(path)
```

`setup_fare_structure` *Setup a fare structure to calculate the monetary costs of trips*

Description

Creates a basic fare structure that describes how transit fares should be calculated in `travel_time_matrix()`, `expanded_travel_time_matrix()`, `accessibility()` and `pareto_frontier()`. This fare structure can be manually edited and adjusted to the existing rules in your study area, as long as they stick to some basic premises. See the fare-structure vignette.

Usage

```
setup_fare_structure(
  r5r_network,
  base_fare,
  by = "MODE",
  debug_path = NULL,
  debug_info = NULL,
  r5r_core = deprecated()
)
```

Arguments

`r5r_network`	A routable transport network created with `build_network()`.
`base_fare`	A numeric. A base value used to populate the fare structure.
`by`	A string. Describes how `fare_types` (a classification we created to assign fares to different routes) are distributed among routes. Possible values are `MODE`, `AGENCY` and `GENERIC`. `MODE` is used when the mode is what determines the price of a route (e.g. if all the buses of a given city cost $5). `AGENCY` is used when the agency that operates each route is what determines its price (i.e. when two different routes/modes operated by a single agency cost the same; note that you can also use `AGENCY_NAME`, if the agency_ids listed in your GTFS cannot be easily interpreted). `GENERIC` is used when all the routes cost the same. This classification can later be edited to better suit your needs (when, for example, two types of buses cost the same, but one offers discounts after riding the subway and the other one doesn't), but this parameter may save you some work.
`debug_path`	Either a path to a `.csv` file or `NULL`. When `NULL` (the default), fare debugging capabilities are disabled - i.e. there's no way to check if the fare calculation is correct. When a path is provided, `r5r` saves different itineraries and their respective fares to the specified file. How each itinerary is described is controlled by `debug_info`.
`debug_info`	Either a string (when `debug_path` is a path) or `NULL` (the default). Ignored if `debug_path` is `NULL`. One of `MODE`, `ROUTE` or `MODE_ROUTE`, setting how each itinerary is written. E.g. a subway leg on route 001 followed by a bus leg on route 007 is written as `SUBWAY|BUS`, `001|007` or `SUBWAY 001|BUS 007`, respectively. The debug output lists all itineraries `R5` checked when calculating

routes, not only those returned by `travel_time_matrix()`, `accessibility()` and `pareto_frontier()`. This slows computation, but the larger sample of itineraries helps find issues in the fare structure.

`r5r_core` The `r5r_core` argument is deprecated as of r5r v2.3.0. Use the `r5r_network` argument instead.

Value

A fare structure object.

See Also

Other fare structure: `read_fare_structure()`, `write_fare_structure()`

Examples

```
library(r5r)

data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path)

fare_structure <- setup_fare_structure(r5r_network, base_fare = 5)

# to debug fare calculation
fare_structure <- setup_fare_structure(
  r5r_network,
  base_fare = 5,
  debug_path = "fare_debug.csv",
  debug_info = "MODE"
)

fare_structure$debug_settings

# debugging can be manually turned off by setting output_file to ""
fare_structure$debug_settings <- ""
```

`setup_r5` *Create a transport network used for routing in R5 (deprecated)*

Description

[Deprecated]

`setup_r5()` was renamed to `build_network()` to create a more consistent API. `setup_r5()` **is being deprecated** after *r5r* v2.3.0 and will be **removed in a future release**. Use `build_network()` instead.

Usage

```
setup_r5(
  data_path,
  verbose = FALSE,
  temp_dir = FALSE,
  elevation = "TOBLER",
  overwrite = FALSE
)
```

Arguments

data_path	A string pointing to the directory where data inputs are stored and where the built network.dat will be saved.
verbose	A logical. Whether to show R5 informative messages when running the function. Defaults to FALSE (R5 error messages are still shown). TRUE shows detailed output, useful for debugging issues not caught by r5r.
temp_dir	A logical. Whether the network.dat file should be saved to a temporary directory. Defaults to FALSE.
elevation	A string. The name of the impedance function to be used to calculate impedance for walking and cycling based on street slopes. Available options include TOBLER (Default) and MINETTI, or NONE to ignore elevation. R5 loads elevation data from .tif files saved inside the data_path directory. Elevation raster must be in WGS 84 (EPSG:4326) coordinate reference system. See more info in the Details section below.
overwrite	A logical. Whether to overwrite an existing network.dat or to use a cached file. Defaults to FALSE (i.e. use a cached network).

Value

An r5r_network object, used by r5r's routing and support functions.

Elevation

More information about the TOBLER and MINETTI options to calculate the effects of elevation on travel times can be found in the references below:

- Campbell, M. J., et al (2019). Using crowdsourced fitness tracker data to model the relationship between slope and travel rates. Applied geography, 106, 93-107. doi:10.1016/j.apgeog.2019.03.008.
- Minetti, A. E., et al (2002). Energy cost of walking and running at extreme uphill and downhill slopes. Journal of applied physiology. doi:10.1152/japplphysiol.01177.2001.
- Tobler, W. (1993). Three presentations on geographical analysis and modeling: Non-isotropic geographic modeling speculations on the geometry of geography global spatial analysis. Technical Report. National center for geographic information and analysis. 93 (1). https://escholarship.org/uc/item/05r820mz.

See Also

Other Build network: build_network(), download_r5()

Examples

```
library(r5r)

# directory with street network and gtfs files
data_path <- system.file("extdata/poa", package = "r5r")

# `setup_r5()` has been deprecated, please switch to `build_network()`
r5r_network <- build_network(data_path)
```

---

stop_r5 *Stop running r5r network*

Description

Stops the given r5r_network objects, or all r5r_network objects in the calling environment if none is supplied.

Usage

```
stop_r5(...)
```

Arguments

...	r5r_network objects currently running. By default, if no r5r network is supplied all running networks are stopped.

Value

No return value, called for side effects.

See Also

Other support functions: exists_tiff(), fileurl_from_metadata(), get_gtfs_errors(), start_r5r_java(), tempdir_unique(), travel_time_surface(), validate_bad_osm_ids()

Examples

```
library(r5r)

path <- system.file("extdata/poa", package = "r5r")

r5r_network <- build_network(path)

stop_r5(r5r_network)
```

`street_network_bbox` *Extract the geographic bounding box of the transport network*

Description

Extracts the geographic bounding box of the street network layer from a routable transport network built with `build_network()`. It is a fast and memory-efficient alternative to `sf::st_bbox(street_network_to_sf(r5r_ne`

Usage

```
street_network_bbox(
  r5r_network,
  output = c("polygon", "bbox", "vector"),
  r5r_core = deprecated()
)
```

Arguments

`r5r_network`	A routable transport network created with `build_network()`.
`output`	A character string specifying the desired output format. One of `"polygon"` (the default), `"bbox"`, or `"vector"`.
`r5r_core`	The `r5r_core` argument is deprecated as of r5r v2.3.0. Use the `r5r_network` argument instead.

Value

By default (`output = "polygon"`), an `sf` object with a single `POLYGON` geometry. If `output = "bbox"`, an `sf bbox` object. If `output = "vector"`, a named numeric vector with `xmin`, `ymin`, `xmax`, `ymax` coordinates. All outputs use the WGS84 coordinate reference system (EPSG: 4326).

See Also

Other network functions: `find_snap()`, `street_network_to_sf()`, `transit_network_to_sf()`

Examples

```
library(r5r)
library(sf)

data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path)

# Get the network's bounding box as an sf polygon (default)
poly <- street_network_bbox(r5r_network, output = "polygon")
plot(poly)

# Get an sf bbox object (order is xmin, ymin, xmax, ymax)
box <- street_network_bbox(r5r_network , output = "bbox")
```

```
box

# Get a simple named vector (order now also xmin, ymin, xmax, ymax)
vec <- street_network_bbox(r5r_network , output = "vector")
vec

stop_r5(r5r_network)
```

`street_network_to_sf` *Extract OpenStreetMap network in sf format*

Description

Extracts the OpenStreetMap network in `sf` format from a routable transport network built with `build_network()`.

Usage

```
street_network_to_sf(r5r_network, r5r_core = deprecated())
```

Arguments

`r5r_network`	A routable transport network created with `build_network()`.
`r5r_core`	The `r5r_core` argument is deprecated as of r5r v2.3.0. Use the `r5r_network` argument instead.

Value

A list with two components of a street network in sf format: vertices (POINT) and edges (LINESTRING).

See Also

Other network functions: `find_snap()`, `street_network_bbox()`, `transit_network_to_sf()`

Examples

```
library(r5r)

# build transport network
path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(path)

# extract street network from r5r_network
street_net <- street_network_to_sf(r5r_network)

stop_r5(r5r_network)
```

`transit_network_to_sf` *Extract transit network in sf format*

Description

Extracts the transit network in `sf` format from a routable transport network built with `build_network()`.

Usage

```
transit_network_to_sf(r5r_network, r5r_core = deprecated())
```

Arguments

`r5r_network`	A routable transport network created with `build_network()`.
`r5r_core`	The `r5r_core` argument is deprecated as of r5r v2.3.0. Use the `r5r_network` argument instead.

Value

A list with two components of a transit network in `sf` format: route shapes (`LINESTRING`) and transit stops (`POINT`). The same `route_id`/`short_name` might appear with different geometries. This occurs when the same route is associated to more than one `shape_ids` in the GTFS feed used to create the transit network. Some transit stops might be returned with geometry `POINT EMPTY` (i.e. missing spatial coordinates). This may occur when a transit stop is not snapped to the road network, possibly because the GTFS feed used to create the transit network covers an area larger than the `.osm.pbf` input data.

See Also

Other network functions: `find_snap()`, `street_network_bbox()`, `street_network_to_sf()`

Examples

```
library(r5r)

# build transport network
path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(path)

# extract transit network from r5r_network
transit_net <- transit_network_to_sf(r5r_network)

stop_r5(r5r_network)
```

`travel_time_matrix` *Calculate travel time matrix between origin destination pairs considering a departure time*

Description

Fast computation of travel time estimates between one or multiple origin destination pairs. This function considers a departure time set by the user. If you want to calculate travel times considering a time of arrival, have a look at the `arrival_travel_time_matrix()` function.

Usage

```
travel_time_matrix(
  r5r_network,
  origins,
  destinations,
  mode = "WALK",
  mode_egress = "WALK",
  departure_datetime = NULL,
  time_window = 10L,
  percentiles = 50L,
  max_walk_time = Inf,
  max_bike_time = Inf,
  max_car_time = Inf,
  max_trip_duration = 120L,
  walk_speed = 3.6,
  bike_speed = 12,
  max_rides = 3,
  max_lts = 2,
  fare_structure = NULL,
  max_fare = Inf,
  new_carspeeds = NULL,
  carspeed_scale = 1,
  new_lts = NULL,
  draws_per_minute = 5L,
  seed = NULL,
  n_threads = Inf,
  verbose = FALSE,
  progress = TRUE,
  output_dir = NULL,
  r5r_core = deprecated()
)
```

Arguments

`r5r_network` A routable transport network created with `build_network()`.

origins, destinations
: Either a POINT sf object with WGS84 CRS, or a data.frame containing the columns id, lon and lat.

mode
: A character vector. The transport modes allowed for access, transfer and vehicle legs of the trips. Defaults to WALK. See details for other options.

mode_egress
: A character vector. The transport mode used after egress from the last public transport. It can be either WALK, BICYCLE or CAR. Defaults to WALK. Ignored when public transport is not used.

departure_datetime
: A POSIXct object. Only affects public transport legs; when routing with public transport, it must fall within the service period of the GTFS feeds (calendar.txt; see check_transit_availability()). Required for public transport; optional otherwise. See details for how datetimes are parsed.

time_window
: An integer. The time window in minutes. Departures are simulated every minute from departure_datetime until time_window minutes later, and travel times are summarized over these departures using percentiles (the median by default). With a fare_structure, and always in pareto_frontier(), departures are instead sampled at random within the window: draws_per_minute per minute on average (one in pareto_frontier()), different for each origin and the same in every call from the same origin with the same seed. Defaults to 10. See vignette("time_window", package = "r5r").

percentiles
: An integer vector (max length of 5). Specifies the percentile to use when returning travel time estimates within the given time window. For example, if the 25th travel time percentile between A and B is 15 minutes, 25% of all trips taken between these points within the specified time window are shorter than 15 minutes. Defaults to 50, returning the median travel time. If a vector with length bigger than 1 is passed, the output contains an additional column for each percentile specifying the percentile travel time estimate. Due to upstream restrictions, only 5 percentiles can be specified at a time. See the R5 documentation at https://docs.conveyal.com/analysis/methodology#accounting-for-variability.

max_walk_time
: An integer. The maximum walking time (in minutes) to access and egress the transit network, make transfers, or complete walk-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In walk-only trips, the lower of max_walk_time and max_trip_duration applies.

max_bike_time
: An integer. The maximum cycling time (in minutes) to access and egress the transit network, make transfers, or complete bicycle-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In bicycle-only trips, the lower of max_bike_time and max_trip_duration applies.

max_car_time
: An integer. The maximum driving time (in minutes) to access and egress the transit network, or to complete car-only trips. Applies to each leg separately (e.g. 15 allows up to 15 minutes to reach transit and another 15 after leaving it). Defaults to Inf (no limit besides max_trip_duration). In car-only trips, the lower of max_car_time and max_trip_duration applies.

max_trip_duration
: An integer. The maximum trip duration in minutes. Defaults to 120.

walk_speed
: A numeric. Average walk speed in km/h. Defaults to 3.6.

bike_speed
: A numeric. Average cycling speed in km/h. Defaults to 12.

max_rides
: An integer. The maximum number of public transport rides allowed in the same trip. Defaults to 3.

max_lts
: An integer between 1 and 4. The maximum level of traffic stress that cyclists will tolerate. A value of 1 means cyclists will only travel through the quietest streets, while a value of 4 indicates cyclists can travel through any road. Defaults to 2. See details.

fare_structure
: A fare structure object, following the convention set in `setup_fare_structure()`. This object describes how transit fares should be calculated. See `vignette("fare_structure", package = "r5r")` for its structure.

max_fare
: A number. The maximum value that trips can cost when calculating the fastest journey between each origin and destination pair. Defaults to `Inf` (no limit). A finite value requires a `fare_structure`; an error is raised otherwise.

new_carspeeds
: A `data.frame` specifying the new car speed for each OSM edge id. This table must contain columns `osm_id`, `max_speed` and `speed_type`. The `"speed_type"` column is of class character and it indicates whether the values in `"max_speed"` should be interpreted as percentages of original speeds (`"scale"`) or as absolute speeds (`"km/h"`). Alternatively, the `new_carspeeds` parameter can receive an `sf data.frame` with POLYGON geometry that indicates the new car speed for all the roads that fall within each polygon. In this case, the table must contain the columns `poly_id` with a unique id for each polygon, `scale` with the new speed scaling factors and `priority`, which is a number ranking which polygon should be considered in case of overlapping polygons. See more info in the scenarios vignette (`vignette("scenarios", package = "r5r")`).

carspeed_scale
: Numeric. The scaling factor applied to the car speed of road segments not specified in `new_carspeeds`. Defaults to 1, which keeps the speeds of the unlisted roads unchanged.

new_lts
: A `data.frame` specifying the new LTS levels for each OSM edge id. The table must contain columns `osm_id` and `lts`. Alternatively, the `new_lts` parameter can receive an `sf data.frame` with LINESTRING geometry. R5 will then find the nearest road for each LINESTRING and update its LTS value accordingly.

draws_per_minute
: An integer. The number of Monte Carlo draws to perform per minute of `time_window`. Defaults to 5. This would mean 300 draws in a 60-minute time window, for example. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a `frequencies.txt` table; otherwise, r5r still routes a departure every minute of `time_window`, in a deterministic way. In functions that accept a `fare_structure`, passing one switches to the McRaptor algorithm, which instead routes `time_window * draws_per_minute` departure times sampled at random within the window (different for each origin, and the same in every call from the same origin with the same `seed`). This parameter then affects the results with or without frequencies. R5's McRaptor cannot sample many departure times per minute over long windows, so such combinations

raise an error (e.g. 8 draws per minute in a 10-minute window, or 5 in a window of 150 minutes or more), as does a total of exactly 200 draws.

`seed` An integer or `NULL`. The seed of the Monte Carlo draws of the departure offsets of frequency-based transit routes and, with the McRaptor algorithm, of the departure times sampled within `time_window`. Defaults to `NULL`, in which case the draws are random and results may vary slightly between runs. With an integer, the same inputs always give the same results, regardless of `n_threads`. Results are still a Monte Carlo sample, so a different seed may give slightly different results: report the seed used and check how sensitive the results are to it. Note that `set.seed()` has no effect on these draws. With the default routing algorithm, this parameter only affects the results when the GTFS feeds contain a `frequencies.txt` table. In functions that accept a `fare_structure`, passing one switches to the McRaptor algorithm, and `seed` then affects the results with or without frequencies.

`n_threads` An integer. The number of threads to use when running the router in parallel. Defaults to `Inf` (all available threads).

`verbose` A logical. Whether to show `R5` informative messages when running the function. Defaults to `FALSE` (`R5` error messages are still shown). `TRUE` shows detailed output, useful for debugging issues not caught by `r5r`.

`progress` A logical. Whether to show a progress counter when running the router. Defaults to `TRUE`. Only works when `verbose` is `FALSE`. May slightly slow computation, as the counter is synchronized across threads.

`output_dir` Either `NULL` (the default) or a path to an existing directory. When a path is given, the function writes the results as `.csv` files to that directory and returns the path instead of the results. Useful in memory-constrained settings, as results are not loaded into RAM. Missing values (`NA`) are written as empty fields.

`r5r_core` The `r5r_core` argument is deprecated as of r5r v2.3.0. Use the `r5r_network` argument instead.

Value

A `data.table` with travel time estimates (in minutes) between origin and destination pairs. Pairs whose trips couldn't be completed within the maximum travel time and/or whose origin is too far from the street network are not returned in the `data.table`. If `output_dir` is not `NULL`, the function returns the path specified in that parameter, in which the `.csv` files containing the results are saved.

Transport modes

`R5` allows for multiple combinations of transport modes. The options include:

- **Transit modes:** `TRAM`, `SUBWAY`, `RAIL`, `BUS`, `FERRY`, `CABLE_CAR`, `GONDOLA`, `FUNICULAR`. The option `TRANSIT` automatically considers all public transport modes available.
- **Non transit modes:** `WALK`, `BICYCLE`, `CAR`.

Level of Traffic Stress (LTS)

When cycling is enabled in `R5` (by passing the value `BICYCLE` to either `mode` or `mode_egress`), setting `max_lts` will allow cycling only on streets with a given level of danger/stress. Setting

max_lts to 1, for example, will allow cycling only on separated bicycle infrastructure or low-traffic streets and routing will revert to walking when traversing any links with LTS exceeding 1. Setting max_lts to 3 will allow cycling on links with LTS 1, 2 or 3. Routing also reverts to walking if the street segment is tagged as non-bikable in OSM (e.g. a staircase), independently of the specified max LTS.

The default methodology for assigning LTS values to network edges is based on commonly tagged attributes of OSM ways. See more info about LTS in the original documentation of R5 from Conveyal at https://docs.conveyal.com/learn-more/traffic-stress. In summary:

- **LTS 1**: Tolerable for children. This includes low-speed, low-volume streets, as well as those with separated bicycle facilities (such as parking-protected lanes or cycle tracks).
- **LTS 2**: Tolerable for the mainstream adult population. This includes streets where cyclists have dedicated lanes and only have to interact with traffic at formal crossing.
- **LTS 3**: Tolerable for "enthused and confident" cyclists. This includes streets which may involve close proximity to moderate- or high-speed vehicular traffic.
- **LTS 4**: Tolerable only for "strong and fearless" cyclists. This includes streets where cyclists are required to mix with moderate- to high-speed vehicular traffic.

For advanced users, you can provide custom LTS values by adding a tag <key = "lts"> to the osm.pbf file.

Datetime parsing

r5r ignores the timezone attribute of datetime objects when parsing dates and times, using the study area's timezone instead. For example, let's say you are running some calculations using Rio de Janeiro, Brazil, as your study area. The datetime as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S") will be parsed as May 13th, 2019, 14:00h in Rio's local time, as expected. But as.POSIXct("13-05-2019 14:00:00", format = "%d-%m-%Y %H:%M:%S", tz = "Europe/Paris") will also be parsed as the exact same date and time in Rio's local time, perhaps surprisingly, ignoring the timezone attribute.

Routing algorithm

The travel_time_matrix(), expanded_travel_time_matrix(), arrival_travel_time_matrix() and accessibility() functions use an R5-specific extension to the RAPTOR routing algorithm (see Conway et al., 2017). This RAPTOR extension uses a systematic sample of one departure per minute over the time window set by the user in the 'time_window' parameter. A detailed description of base RAPTOR can be found in Delling et al (2015). However, whenever the user includes transit fares inputs to these functions, they automatically switch to use an R5-specific extension to the McRAPTOR routing algorithm. detailed_itineraries() also uses RAPTOR when it only returns the fastest itinerary (shortest_path = TRUE) and no fares are used.

- Conway, M. W., Byrd, A., & van der Linden, M. (2017). Evidence-based transit and land use sketch planning using interactive accessibility methods on combined schedule and headway-based networks. Transportation Research Record, 2653(1), 45-53. doi:10.3141/265306
- Delling, D., Pajor, T., & Werneck, R. F. (2015). Round-based public transit routing. Transportation Science, 49(3), 591-604. doi:10.1287/trsc.2014.0534

See Also

Other routing: arrival_travel_time_matrix(), detailed_itineraries(), expanded_travel_time_matrix(), pareto_frontier()

Examples

```
library(r5r)

# build transport network
data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path)

# load origin/destination points
points <- read.csv(file.path(data_path, "poa_points_of_interest.csv"))

departure_datetime <- as.POSIXct(
  "13-05-2019 14:00:00",
  format = "%d-%m-%Y %H:%M:%S"
)

ttm <- travel_time_matrix(
  r5r_network,
  origins = points,
  destinations = points,
  mode = c("WALK", "TRANSIT"),
  departure_datetime = departure_datetime,
  max_trip_duration = 60
)
head(ttm)

# using a larger time window
ttm <- travel_time_matrix(
  r5r_network,
  origins = points,
  destinations = points,
  mode = c("WALK", "TRANSIT"),
  departure_datetime = departure_datetime,
  time_window = 30,
  max_trip_duration = 60
)
head(ttm)

# selecting different percentiles
ttm <- travel_time_matrix(
  r5r_network,
  origins = points,
  destinations = points,
  mode = c("WALK", "TRANSIT"),
  departure_datetime = departure_datetime,
  time_window = 30,
  percentiles = c(25, 50, 75),
  max_trip_duration = 60
```

```
)
head(ttm)

# use a fare structure and set a max fare to take monetary constraints into
# account
fare_structure <- read_fare_structure(
  file.path(data_path, "fares/fares_poa.zip")
)
ttm <- travel_time_matrix(
  r5r_network,
  origins = points,
  destinations = points,
  mode = c("WALK", "TRANSIT"),
  departure_datetime = departure_datetime,
  fare_structure = fare_structure,
  max_fare = 5,
  max_trip_duration = 60,
)
head(ttm)

stop_r5(r5r_network)
```

`write_fare_structure` *Write a fare structure object to disk*

Description

Writes a fare structure object to disk. Fare structure is saved as a collection of `.csv` files inside a `.zip` file.

Usage

```
write_fare_structure(fare_structure, file_path)
```

Arguments

`fare_structure`	A fare structure object, following the convention set in `setup_fare_structure()`. This object describes how transit fares should be calculated. See `vignette("fare_structure", package = "r5r")` for its structure.
`file_path`	A path to a `.zip` file. Where the fare structure should be written to.

Value

The path passed to `file_path`, invisibly.

See Also

Other fare structure: `read_fare_structure()`, `setup_fare_structure()`

Examples

```
library(r5r)

data_path <- system.file("extdata/poa", package = "r5r")
r5r_network <- build_network(data_path)

fare_structure <- setup_fare_structure(r5r_network, base_fare = 5)

tmpfile <- tempfile("sample_fare_structure", fileext = ".zip")
write_fare_structure(fare_structure, tmpfile)
```

Index

∗ **Build network**
  build_network, 16
  download_r5, 26
  setup_r5, 50
∗ **Cache data**
  r5r_cache, 47
∗ **Isochrone**
  isochrone, 36
∗ **accessibility**
  accessibility, 3
∗ **fare structure**
  read_fare_structure, 48
  setup_fare_structure, 49
  write_fare_structure, 62
∗ **network functions**
  find_snap, 33
  street_network_bbox, 53
  street_network_to_sf, 54
  transit_network_to_sf, 55
∗ **routing**
  arrival_travel_time_matrix, 10
  detailed_itineraries, 19
  expanded_travel_time_matrix, 27
  pareto_frontier, 41
  travel_time_matrix, 56
∗ **support functions**
  get_gtfs_errors, 35
  stop_r5, 52

accessibility, 3
accessibility(), *9*, *15*, *32*, *40*, *49, 50*, *60*
arrival_travel_time_matrix, 10
arrival_travel_time_matrix(), *9*, *15*, *25*, *32*, *40*, *46*, *56*, *60, 61*

build_network, 16
build_network(), *3*, *11*, *18*, *20*, *27, 28*, *34*, *36*, *42*, *49, 50*, *52–56*

check_transit_availability, 18
check_transit_availability(), *4*, *12*, *21*, *28*, *37*, *42*, *57*

data.table::fread(), *48*
detailed_itineraries, 19
detailed_itineraries(), *9*, *15*, *25*, *32*, *40*, *45, 46*, *60, 61*
download_r5, 26
download_r5(), *17*, *52*

exists_tiff(), *35*, *52*
expanded_travel_time_matrix, 27
expanded_travel_time_matrix(), *9, 10*, *14, 15*, *25*, *32*, *40*, *46*, *49*, *60, 61*

fileurl_from_metadata(), *35*, *52*
find_snap, 33
find_snap(), *7*, *53–55*

get_gtfs_errors, 35
get_gtfs_errors(), *52*

isochrone, 36

pareto_frontier, 41
pareto_frontier(), *4*, *15*, *25*, *32*, *37*, *43*, *45*, *49, 50*, *57*, *61*

r5r_cache, 47
r5r_sitrep, 47
read_fare_structure, 48
read_fare_structure(), *50*, *62*

setup_fare_structure, 49
setup_fare_structure(), *5*, *22*, *44*, *48*, *58*, *62*
setup_r5, 50
setup_r5(), *17*, *27*
start_r5r_java(), *35*, *52*
stop_r5, 52
stop_r5(), *35*

street_network_bbox, 53
street_network_bbox(), *34*, *54, 55*
street_network_to_sf, 54
street_network_to_sf(), *23*, *34*, *53*, *55*

tempdir_unique(), *35*, *52*
transit_network_to_sf, 55
transit_network_to_sf(), *34*, *53, 54*
travel_time_matrix, 56
travel_time_matrix(), *9, 10*, *15*, *25*, *32*, *40*, *46*, *49, 50*, *60*
travel_time_surface(), *35*, *52*

validate_bad_osm_ids(), *35*, *52*

write_fare_structure, 62
write_fare_structure(), *48*, *50*