Package 'ghroute'

November 29, 2023

Title GraphHopper Routing and Navigation

Version 0.2-1

Author Simon Urbanek <simon.urbanek@R-project.org>

Maintainer Simon Urbanek <simon.urbanek@R-project.org>

Description Provides functions for efficient calculation of routes and navigation using Open Street Map and public transport data. It directly embeds GraphHopper into R and is self-contained without the need for any API or service. It supports parallel multi-threaded routing of many routes in one batch.

Imports rJava (>= 1.0)

Suggests sf

License Apache License 2.0

SystemRequirements Java >= 8

BugReports https://github.com/s-u/ghroute

NeedsCompilation no

Contents

`GHRoutes` *GraphHopper Routes Class*

Description

`GHRoutes` is a class of objects representing results from the GraphHopper routing when `output="gh"` is used.

I behaves like a non-mutable list such that usual operations such as subsetting, element extraction, iteration and `length()` work as expected. However, the object should be consiered non-mutable, i.e., it is not possible to assign new values into an existing object. Although subsetting is allowed, concatenation is not.

The object itself contains Java references which enables low-level access to the underlying results using tha GraphHopper Java API.

It is the only result type which supports representation of alternative routes. If `alt=TRUE` is used then each (virtual) element contains a list of result paths of which the first one is considerend the best according to GraphHopper.

Note that this API is considered experimental and is subject to change.

Author(s)

Simon Urbanek

`router` *Street Routing Functions*

Description

`router` initializes a GraphHopper router with a specific profile and data sources.

`route` computes a route using the specified router.

`gh.profile` defines a new routing profile

Usage

```
router(osm.file, path = "graphhopper-cache", profiles = "car", open = TRUE,
       make.default = TRUE)

route(x, ...)
## S3 method for class 'matrix'
route(x, profile, times, alt=FALSE, output=c("matrix","sf","gh"),
             silent=FALSE, threads=1L, router=.default(), ...)
## Default S3 method:
route(x, start.lon, end.lat, end.lon, ...)

gh.profile(name, vehicle = name, weighting = "fastest", turn.costs = FALSE)
gh.translation(locale)
```

Arguments

`osm.file`	string, path to the Open Street Map file used for routing
`path`	string, path to the GraphHopper cache. If it doesn't exist the cache will be created when first used
`profiles`	either a character vector or a list with one or more objects returned from `gh.profile()`. This determines the properties for the routing such as available means of transport, speeds, turn penalties etc.
`open`	logical, if `TRUE` then the returned object can be used for routing. If `FALSE` then the cache is created and saved for later use, but no router is created for the current session.
`make.default`	logical, if `TRUE` then the resulting router is used as the default router for subseqeunt operations

x	object specifying the waypoints for the routes. See details for the supported input specifications.
start.lon	scalar numeric, longitude of the starting point
end.lat	scalar numeric, latitude of the end point
end.lon	scalar numeric, longitude of the end point
profile	either a string or a ghprofile object. If not specified, defaults to the first profile of the router
output	desired output format: "matrix" for a matrix describing the paths, "sf" for a data frame with a "sfc" geometry column or "gh" GraphHoppe objects.
times	optional, vector of start times for time-dependent routing (such as public transport)
silent	logical, if TRUE then warnings (such as routing errors) are suppressed
alt	logical, if TRUE then multiple alternative routes are included in the result is available, otherwise only the best route is reported in each case. Note that only the "gh" format supports multiple routes per query.
threads	integer, number of parallel threads to use for the routing. For large matrices on private machines parallel::detectCores() or similar may make sense
router	object returned from the router() function
name	string, name of the profile
vehicle	string, mode of transport
weighting	string, weighting type
turn.costs	logical, whether to use turn costs
locale	string, name of the language locale
...	parameters passed to methods

Details

router must be called at least once to initialize the routing parameters.

gh.profile defines a profile to be used in the router. Note that a new cache must be built any time the profiles are changed.

route calculates one or more routes for each source/destination pair. The matrix form requires a matrix with exactly 4 columns specifying latitude and longitude of the start and end point respectiely. Each row of the input matrix will have exactly one entry in the result.

The scalar (default) version is just a wrapper for matrix(c(start.lat, start.lon, end.lat, end.lon), 1). Note that due to R method argument consistency requirements the first argument of the default method is actually called x even though it would better be named start.lat.

gh.translation loads translation from GraphHopper for a given language locale and return the corresponding Java object. "en" is always guaranteed to exist, others depend on the GraphHopper installation.

Value

router returns a router object.

route: the value depends on the "output" argument.

The "matrix" output is a matrix with columns "lat", "lon" and "index" where "index" is the row number in the input. Note that routing errors may occur (e.g. if the endpoint is not near

any roads) in which case the corresponding row may not appear in the output. Use `output="gh"` for comprehensive error reporting.

The `"sf"` output produces a data frame with a geometry column representing the best path. It case of an error an empty line string is used. Note that this output type requires the `sf` package.

The `"gh"` output produces an object of the class `GHRoutes` which holds the corresponding Java objects and thus allows for explicit queries for different aspects of the routes including the support for alternative routes.

`gh.translation` returns a Java object of the class `com.graphhopper.util.TranslationMap`.

`gh.profile` returns a Java object of the class `com.graphhopper.config.Profile`. Note that since it is a Java object it can be modified by calling methods on it before passing it to `router`.

Note

The `router()` uses the `rJava` package to perform the routing using the GraphHopper Java classes. GraphHopper may need significant amount of RAM depending on the size of the map used. For larger maps it is recommended to increase the amount of memory available to Java by setting the `java.parameters` option *before* the first call to `router()`, e.g.:

```
options(java.parameters="-Xmx8g")
```

will allow Java to use up to 8Gb of memory. Note that the amount of Java memory cannot be changed once Java is initialized, so if you load any other R package which uses Java it may have already limited the memory usage, so either setting the option early or starting the router first is a good option.

The functions above are a very thin API over GraphHopper Java classes. It is possible to create more complex routing constraints by using the Java classes directly. For example, the `gh.profile` function returns an object of the class `com.graphhopper.config.Profile` which can be further mutated before passing it to the router. It corresponds directly to the `profiles:` section of the YAML config files in the GraphHopper documentation. By default all entries in `profiles` are also automatically added to the contraction hierarchies (see `profiles_ch:` section and `com.graphhopper.confi`

Author(s)

Simon Urbanek

Examples

```
dst <- tempfile("osm", fileext=".pbf")
cache <- tempfile("router-cache")

## download OSM map of Cyprus
download.file("https://download.geofabrik.de/europe/cyprus-latest.osm.pbf", dst, mode="wb

library(ghroute)
## create router for Cyprus with defaults car and fastest
router(dst, cache)

## compute single route between two points
rt = route(34.7592, 32.4123, 34.7996, 32.4517)

## by default the result is a lat/lon matrix
str(rt)

par(mar=rep(0,4))
```

```
plot(rt[,2], rt[,1], ty='l', asp=1/cos(32.4/180*pi))

## it is also possible to compute multiple different
## routes by supplying a start-end matrix
m = matrix(c(34.7592, 32.4123, 34.7996, 32.4517,
             34.7592, 32.4123, 34.6791, 33.044,
             34.9827, 33.7348, 35.1716, 33.3616),,4,TRUE)

## the result is a big matrix with "index" column
## refering to the input row
rts = route(m)

str(rts)

## it is possible to split the result into a list of
## matrices - each for one input row
res = rts2list(rts, nrow(m))
str(res)

plot(rts[,2], rts[,1], ty='n', asp=1/cos(32.4/180*pi))
for (o in res) lines(o[,2], o[,1])

## if you have sf package, you can get sfc instead:
have.sf <- requireNamespace("sf", quietly=TRUE)
if (have.sf) {
  rts = route(m, output="sf")
  print(rts)
  plot(rts$geometry)
}

## the maximum flexilibity is with "gh" output
rts = route(m, output="gh")

rts

rts[[1]]

## expert use - return turn-by-turn instructions by querying the Java object
## get the instructions object
ins <- rts[[1]]$path$getInstructions()
## use English locale
tr <- gh.translation("en")
## get the descriptions
sapply(seq.int(ins$size()),
       function(j) ins$get(j - 1L)$getTurnDescription(tr))

## clean up
unlink(c(dst, cache), TRUE, FALSE, FALSE)
```

`rts2list` *Split route result matrix by index*

Description

`rts2list` converts route result matrix into a list of individual matrices based on the supplied index which is typically part of the matrix.

Usage

```
rts2list(rts, nrow = max(index), index = rts[, 3])
```

Arguments

`rts`	matrix
`nrow`	total number of input rows
`index`	index mapping rows of the matrix to result rows

Details

The matrix `rts` is split into indiviudal matrices according to the `index`. It is similar to `split(rts, index)` except that the result has exactly `nrow` entries corresponding the the indices 1 .. `nrow` and index must be contiguous. Any missing entries (i.e., indices between 1 and `nrow` which are not in `index`) are filled with `NULL`.

Value

A list of exactly `nrow` elements. The entries are either `NULL` (index not present) or the corresponding slice of the matrix.

Author(s)

Simon Urbanek

Examples

```
m = matrix(1:10,, 2)
rts2list(m, 3, c(1,1,1,3,3))

## route matrices have the index as 3rd column
m = cbind(m, c(1,1,1,3,3))

## let's say the input had 5 pairs
rts2list(m, 5)
```

Index