Package 'valh'

February 12, 2026

Type Package

Title Interface Between R and the OpenStreetMap-Based Routing Service 'Valhalla'

Version 0.2.0

Description An interface between R and the 'Valhalla' API. 'Valhalla' is a routing service based on 'OpenStreetMap' data.
See <https://valhalla.github.io/valhalla/> for more information.
This package enables the computation of routes, trips, isochrones and travel distances matrices (travel time and kilometer distance).

License GPL (>= 3)

Imports jsonlite, googlePolylines, curl, utils, sf

Depends R (>= 3.5.0)

Suggests knitr, rmarkdown, tinytest

URL https://github.com/riatelab/valh

BugReports https://github.com/riatelab/valh/issues

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Timothée Giraud [cre, aut] (ORCID: <https://orcid.org/0000-0002-1932-3323>),
Matthieu Viry [aut] (ORCID: <https://orcid.org/0000-0002-0693-8556>)

Maintainer Timothée Giraud <timothee.giraud@cnrs.fr>

Repository CRAN

Date/Publication 2026-02-12 14:20:02 UTC

Contents

`valh` *Shortest Paths and Travel Time with the OpenStreetMap-Based Routing Service Valhalla*

Description

An interface between R and the Valhalla API.
Valhalla is a routing service based on OpenStreetMap data. See `https://valhalla.github.io/valhalla/` for more information.
This package enables the computation of routes, trips, isochrones and travel distances matrices.

- `vl_matrix`: Build and send Valhalla API queries to get travel time matrices between points. This function interfaces the *matrix* Valhalla service.
- `vl_route`: Build and send a Valhalla API query to get the travel geometry between two points. This function interfaces with the *route* Valhalla service.
- `vl_optimized_route`: Build and send a Valhalla API query to get the shortest travel geometry between multiple unordered points. This function interfaces the *optimized_route* Valhalla service. Use this function to resolve the travelling salesman problem.
- `vl_locate`: Build and send an Valhalla API query to get the nearest point on the street network. This function interfaces the *locate* Valhalla service.
- `vl_isochrone`: This function computes areas that are reachable within a given time span (or road distance) from a point and returns the reachable regions as polygons. These areas of equal travel time are called isochrones. This function interfaces the *isochrone & isodistance* Valhalla service.
- `vl_elevation`: Build and send a Valhalla API query to get the elevation at a set of input locations. This function interfaces with the *height* Valhalla service.
- `vl_status`: Build and send a Valhalla API query to get information on the Valhalla server (version etc.).. This function interfaces with the *status* Valhalla service.

Note

This package relies on the usage of a running Valhalla service (tested with versions 3.4.x & 3.5.x of Valhalla).

To use a custom Valhalla instance, you just need to change the `valh.server` option to the url of the instance :
`options(valh.server = "http://address.of.the.server/")`
You can also set this option in your `.Rprofile` file to make it permanent.

The package ships a sample dataset of 100 random pharmacies in Berlin (© OpenStreetMap contributors - `https://www.openstreetmap.org/copyright/en`).
The sf dataset uses the projection WGS 84 / UTM zone 34N (EPSG:32634).
The csv dataset uses WGS 84 (EPSG:4326).

Author(s)

Maintainer: Timothée Giraud `<timothee.giraud@cnrs.fr>` (ORCID)

Authors:

- Matthieu Viry (ORCID)

See Also

Useful links:

- `https://github.com/riatelab/valh`
- Report bugs at `https://github.com/riatelab/valh/issues`

`vl_elevation` *Get elevation along a route*

Description

Build and send a Valhalla API query to get the elevation at a set of input locations.
This function interfaces with the *height* service.
If `sampling_dist` is provided, the elevation is sampled at regular intervals along the input locations.

Usage

```
vl_elevation(loc, sampling_dist, server = getOption("valh.server"))
```

Arguments

`loc`	one (or multiples) point(s) at which to get elevation. `loc` can be: • a vector of coordinates (longitude and latitude, WGS 84), • a data.frame of longitudes and latitudes (WGS 84), • a matrix of longitudes and latitudes (WGS 84), • an sfc object of type POINT, • an sf object of type POINT.
`sampling_dist`	distance between each point to sample the elevation (in meters). Default is no sampling.
`server`	URL of the Valhalla server.

Value

An sf POINT object is returned with the following fields: 'distance' (the distance from the first points), 'height' (the sampled height on the DEM) and 'geometry' (the geometry of the sampled point).

Examples

```
## Not run:
# Inputs are data frames
apotheke.df <- read.csv(system.file("csv/apotheke.csv", package = "valh"))
# The first 5 points
pts <- apotheke.df[1:5, c("lon", "lat")]
# Ask for the elevation at these points
elev1 <- vl_elevation(loc = pts)

# Inputs are sf points
library(sf)
apotheke.sf <- st_read(system.file("gpkg/apotheke.gpkg", package = "valh"),
  quiet = TRUE
)
# The first 5 points
pts2 <- apotheke.sf[1:5, ]
# Ask for the elevation at these points
elev2 <- vl_elevation(loc = pts2)
# Ask for elevation between the first and the second points,
# sampling every 100 meters
elev3 <- vl_elevation(loc = apotheke.sf[1:2, ], sampling_dist = 100)
# Plot the corresponding elevation profile
plot(as.matrix(st_drop_geometry(elev3)), type = "l")

# Input is a route (sf LINESTRING) from vl_route
# Compute the route between the first and the second points
library(sf)
apotheke.sf <- st_read(system.file("gpkg/apotheke.gpkg", package = "valh"),
  quiet = TRUE
)
src <- apotheke.sf[1, ]
```

```
dst <- apotheke.sf[2, ]
route <- vl_route(src = src, dst = dst)

# Split the LINESTRING into its composing points
pts_route <- sf::st_cast(route, "POINT")
# Ask for the elevation at these points
elev4 <- vl_elevation(loc = pts_route)

# Plot the elevation profile
plot(as.matrix(st_drop_geometry(elev4)), type = "l")

## End(Not run)
```

`vl_isochrone` *Get isochrones and isodistances from a point*

Description

Build and send a Valhalla API query to get isochrones or isodistances from a point.
This function interfaces with the *Isochrone & Isodistance* service.
Note that you must provide either 'times' or 'distances' to compute the isochrones at given times or distances from the center point.

Usage

```
vl_isochrone(
  loc,
  times,
  distances,
  costing = "auto",
  costing_options = list(),
  server = getOption("valh.server")
)
```

Arguments

`loc` one point from which to compute isochrones. `loc` can be:

- a vector of coordinates (longitude and latitude, WGS 84),
- a data.frame of longitudes and latitudes (WGS 84),
- a matrix of longitudes and latitudes (WGS 84),
- an sfc object of type POINT,
- an sf object of type POINT.

`times` vector of travel times (in minutes) to compute the isochrones. The maximum number of isochrones is 4. The minimal value must be greater than 0.

`distances` vector of travel distances (in kilometers) to compute the isochrones. The maximum number of isochrones is 4. The minimal value must be greater than 0.

`costing`	costing model to use.
`costing_options`	list of options to use with the costing model (see https://valhalla.github.io/valhalla/api/turn-by-turn/api-reference/#costing-options for more details about the options available for each costing model).
`server`	URL of the Valhalla server.

Value

An sf MULTIPOLYGON object is returned with the following fields: 'metric' (the metric used, either 'time' or 'distance') and 'contour' (the value of the metric).

Examples

```
## Not run:
# Inputs are data frames
apotheke.df <- read.csv(system.file("csv/apotheke.csv", package = "valh"))

# Extract the first point and compute isochrones at 3, 6, 9 and 12 kilometers,
# using the "auto" costing model
pt1 <- apotheke.df[1, c("lon", "lat")]
iso1 <- vl_isochrone(loc = pt1, distances = c(3, 6, 9, 12), costing = "auto")

# Inputs are sf points
library(sf)
apotheke.sf <- st_read(system.file("gpkg/apotheke.gpkg", package = "valh"),
  quiet = TRUE
)
# Extract the first point and compute isochrones at 15, 30, 45 and 60 minutes
# using the "bicycle" costing model
pt2 <- apotheke.sf[1, ]
iso2 <- vl_isochrone(loc = pt2, times = c(15, 30, 45, 60), costing = "bicycle")

## End(Not run)
```

vl_locate — *Get the nearest point on the road network*

Description

This function interfaces with the *locate* Valhalla service.

Usage

```
vl_locate(
  loc,
  verbose = FALSE,
  costing = "auto",
  costing_options = list(),
  server = getOption("valh.server")
)
```

Arguments

`loc`	one (or multiples) point(s) to snap to the street network. `loc` can be: • a vector of coordinates (longitude and latitude, WGS 84), • a data.frame of longitudes and latitudes (WGS 84), • a matrix of longitudes and latitudes (WGS 84), • an sfc object of type POINT, • an sf object of type POINT.
`verbose`	logical indicating whether to return additional information.
`costing`	costing model to use.
`costing_options`	list of options to use with the costing model (see `https://valhalla.github.io/valhalla/api/turn-by-turn/api-reference/#costing-options` for more details about the options available for each costing model).
`server`	URL of the Valhalla server.

Value

If there is only one input point, return a single sf object containing the nearest point(s) on the road network. If there is more than one input point, return a list of sf objects, one for each input point.

Examples

```
## Not run:
# Inputs are data frames
apotheke.df <- read.csv(system.file("csv/apotheke.csv", package = "valh"))

loc <- apotheke.df[1, c("lon", "lat")]

# Ask for the nearest point on the road network at this point
# using "auto" costing model
on_road_1 <- vl_locate(loc = loc)

# Inputs are sf points
library(sf)
apotheke.sf <- st_read(system.file("gpkg/apotheke.gpkg", package = "valh"),
  quiet = TRUE
)

# Ask for one point
locsf1 <- apotheke.sf[1, ]
# The result is a single sf object
on_road_2 <- vl_locate(loc = locsf1)

# Ask for multiple points
```

```
locsf2 <- apotheke.sf[1:3, ]
# The result is a list of sf objects
on_road_3 <- vl_locate(loc = locsf2)

## End(Not run)
```

---

vl_matrix *Get Travel Time Matrices Between Points*

Description

Build and send Valhalla API queries to get travel time matrices between points.
This function interfaces the *matrix* Valhalla service.
Use `src` and `dst` to set different origins and destinations. Use `loc` to compute travel times or travel distances between all points.

Usage

```
vl_matrix(
  src,
  dst,
  loc,
  costing = "auto",
  costing_options = list(),
  server = getOption("valh.server")
)
```

Arguments

`src` origin points. `src` can be:

- a data.frame of longitudes and latitudes (WGS 84),
- a matrix of longitudes and latitudes (WGS 84),
- an sfc object of type POINT,
- an sf object of type POINT.

If relevant, row names are used as identifiers.

`dst` destination. `dst` can be:

- a data.frame of longitudes and latitudes (WGS 84),
- a matrix of longitudes and latitudes (WGS 84),
- an sfc object of type POINT,
- an sf object of type POINT.

If relevant, row names are used as identifiers.

`loc` points. `loc` can be:

- a data.frame of longitudes and latitudes (WGS 84),
- a matrix of longitudes and latitudes (WGS 84),

- an sfc object of type POINT,
- an sf object of type POINT.

If relevant, row names are used as identifiers.

`costing`	costing model to use.
`costing_options`	list of options to use with the costing model (see `https://valhalla.github.io/valhalla/api/turn-by-turn/api-reference/#costing-options` for more details about the options available for each costing model).
`server`	URL of the Valhalla server.

Value

The output of this function is a list composed of one or two matrices and 2 data.frames

- durations: a matrix of travel times (in minutes)
- distances: a matrix of distances (in specified units, default to kilometers)
- sources: a data.frame of the coordinates of the points actually used as starting points (EPSG:4326 - WGS84)
- destinations: a data.frame of the coordinates of the points actually used as destinations (EPSG:4326 - WGS84)

Examples

```
## Not run:
# Inputs are data frames
apotheke.df <- read.csv(system.file("csv/apotheke.csv", package = "valh"))
# Travel time matrix
distA <- vl_matrix(loc = apotheke.df[1:50, c("lon", "lat")])
# First 5 rows and columns
distA$durations[1:5, 1:5]

# Travel time matrix with different sets of origins and destinations
distA2 <- vl_matrix(
  src = apotheke.df[1:10, c("lon", "lat")],
  dst = apotheke.df[11:20, c("lon", "lat")]
)
# First 5 rows and columns
distA2$durations[1:5, 1:5]

# Inputs are sf points
library(sf)
apotheke.sf <- st_read(system.file("gpkg/apotheke.gpkg", package = "valh"),
  quiet = TRUE
)
distA3 <- vl_matrix(loc = apotheke.sf[1:10, ])
# First 5 rows and columns
distA3$durations[1:5, 1:5]

# Travel time matrix with different sets of origins and destinations
```

```
distA4 <- vl_matrix(src = apotheke.sf[1:10, ], dst = apotheke.sf[11:20, ])
# First 5 rows and columns
distA4$durations[1:5, 1:5]

## End(Not run)
```

vl_optimized_route *Get the Optimized Route Between Multiple Points*

Description

Build and send a Valhalla API query to get the optimized route (and so a solution to the Traveling Salesman Problem) between multiple points.
This function interfaces with the *optimized_route* Valhalla service.

Usage

```
vl_optimized_route(
  loc,
  end_at_start = FALSE,
  costing = "auto",
  costing_options = list(),
  server = getOption("valh.server")
)
```

Arguments

`loc` starting point and waypoints to reach along the route. `loc` can be:

- a data.frame of longitudes and latitudes (WGS 84),
- a matrix of longitudes and latitudes (WGS 84),
- an sfc object of type POINT,
- an sf object of type POINT.

The first row or element is the starting point.
Row names, if relevant, or element indexes are used as identifiers.

`end_at_start` logical indicating whether the route should end at the first point (making the trip a loop).

`costing` costing model to use.

`costing_options` list of options to use with the costing model (see https://valhalla.github.io/valhalla/api/turn-by-turn/api-reference/#costing-options for more details about the options available for each costing model).

`server` URL of the Valhalla server.

Value

a list of two elements:

- summary: a list whose elements are a summary of the trip (duration, distance, presence of tolls, highways, time restrictions and ferries),
- shape: an sf LINESTRING of the optimized route.

Examples

```
## Not run:
# Inputs are data frames
apotheke.df <- read.csv(system.file("csv/apotheke.csv", package = "valh"))
pts1 <- apotheke.df[1:6, c("lon", "lat")]

# Compute the optimized route between the first 6 points
# (starting point, 4 waypoints and final destination), by bike
trip1a <- vl_optimized_route(loc = pts1, end_at_start = FALSE, costing = "bicycle")

# Compute the optimized route between the first 6 points returning to the
# starting point, by bike
trip1b <- vl_optimized_route(loc = pts1, end_at_start = TRUE, costing = "bicycle")

# Inputs are sf points
library(sf)
apotheke.sf <- st_read(system.file("gpkg/apotheke.gpkg", package = "valh"),
  quiet = TRUE
)
pts2 <- apotheke.sf[1:6, ]
# Compute the optimized route between the first 6 points
# (starting point, 4 waypoints and final destination)
trip2a <- vl_optimized_route(loc = pts2, end_at_start = FALSE, costing = "auto")

# Compute the optimized route between the first 6 points, returning to the
# starting point
trip2b <- vl_optimized_route(loc = pts2, end_at_start = TRUE, costing = "auto")

## End(Not run)
```

`vl_route` *Get the Shortest Path Between Two Points*

Description

Build and send a Valhalla API query to get the travel geometry between two points.
This function interfaces with the *route* Valhalla service.
Use `src` and `dst` to get the shortest direct route between two points. Use `loc` to get the shortest route between two points using ordered waypoints.

Usage

```
vl_route(
  src,
  dst,
  loc,
  costing = "auto",
  costing_options = list(),
  server = getOption("valh.server")
)
```

Arguments

`src`	starting point of the route. `src` can be: • a vector of coordinates (longitude and latitude, WGS 84), • a data.frame of longitudes and latitudes (WGS 84), • a matrix of longitudes and latitudes (WGS 84), • an sfc object of type POINT, • an sf object of type POINT. If relevant, row names are used as identifiers. If `src` is a data.frame, a matrix, an sfc object or an sf object then only the first row or element is considered.
`dst`	destination of the route. `dst` can be: • a vector of coordinates (longitude and latitude, WGS 84), • a data.frame of longitudes and latitudes (WGS 84), • a matrix of longitudes and latitudes (WGS 84), • an sfc object of type POINT, • an sf object of type POINT. If relevant, row names are used as identifiers. If `dst` is a data.frame, a matrix, an sfc object or an sf object then only the first row or element is considered.
`loc`	starting point, waypoints (optional) and destination of the route. `loc` can be: • a data.frame of longitudes and latitudes (WGS 84), • a matrix of longitudes and latitudes (WGS 84), • an sfc object of type POINT, • an sf object of type POINT. The first row or element is the starting point then waypoints are used in the order they are stored in `loc` and the last row or element is the destination. If relevant, row names are used as identifiers.
`costing`	costing model to use.
`costing_options`	list of options to use with the costing model (see `https://valhalla.github.io/valhalla/api/turn-by-turn/api-reference/#costing-options` for more details about the options available for each costing model).
`server`	URL of the Valhalla server.

Value

The output of this function is an sf LINESTRING of the shortest route.
It contains 4 fields:

- starting point identifier
- destination identifier
- travel time in minutes
- travel distance in kilometers.

Examples

```
## Not run:
# Inputs are data frames
apotheke.df <- read.csv(system.file("csv/apotheke.csv", package = "valh"))
src <- apotheke.df[1, c("lon", "lat")]
dst <- apotheke.df[2, c("lon", "lat")]
# Route between the two points, using bicycle costing model
route1 <- vl_route(src = src, dst = dst, costing = "bicycle")

# Inputs are sf points
library(sf)
apotheke.sf <- st_read(system.file("gpkg/apotheke.gpkg", package = "valh"),
  quiet = TRUE
)
srcsf <- apotheke.sf[1, ]
dstsf <- apotheke.sf[2, ]
# Route between the two points, using bicycle costing model and a custom
# costing option
route2 <- vl_route(
  src = srcsf,
  dst = dstsf,
  costing = "bicycle",
  costing_options = list(cycling_speed = 19)
)

## End(Not run)
```

`vl_status` *Get Valhalla Service Status*

Description

Use this function to return information on the Valhalla server (version etc.).
This function interfaces with the *Status* Valhalla service.

Usage

```
vl_status(server = getOption("valh.server"), verbose = FALSE)
```

Arguments

`server`	URL of the Valhalla server.
`verbose`	if TRUE and if the service has it enabled, it will return additional information about the loaded tileset.

Value

A list with information on the Valhalla service is returned.

Examples

```
## Not run:
vl_status("https://valhalla1.openstreetmap.de/", verbose = FALSE)

## End(Not run)
```

Index