halem Documentation

Release latest

yourname

Aug 30, 2022

CONTENTS

HALEM is a python package for optimizing shipping routes. This package provides an algorithm for optimizing the route for a given hydrodynamic model.

Welcome to HALEM documentation! Please check the contents below for information on installation, getting started and actual example code. If you want to dive straight into the code you can check out our **`GitHub`_** page.

This package is the result of a graduation study. For the MsC theses See: <Thesis Route optimization in dynamic currents J.P. van Halem.pdf>.

CHAPTER

ONE

CONTENTS

1.1 HALEM

This page lists all functions and classes available in the halem module.

1.1.1 Module contents

class halem.**BaseRoadmap**(*number_of_neighbor_layers*, *vship*, *WD_min*, *WVPI*, *repeat=False*, *WWL=20*, *LWL=80*, *ukc=1.5*, *optimization_type=None*, *nodes_index=None*, **args*, ***kwargs*)

Bases: ABC, *NodeReduction*

Absctract Base class for the Roadmap.

Pre-processing function for the HALEM optimizations. In this fucntion the hydrodynamic model and the vessel properties are transformed into weights for the Time dependend Dijkstra function.

number_of_neighbor_layers: number of neigbouring layers for which edges are
created. increasing this number results in a higher directional resolution.

vship: (N (rows) * M (columns)) numpy array that indicates the sailing velocity
in deep water. For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity. For the optimization type cost and co2 N must be larger or equal to 2.

WD_min: numpy array with the draft of the vessel.
Numpy array has the shape of the number of discretisations in the dynamic sailing velocity

WVPI: Numpy array with the total weight of the vessel.

WWL: Width over Water Line of the vessel in meters

LWL: Length over Water Line of the vessel in meters

ukc: Minimal needed under keel clearance in meters.

repeat: Indicator if the roadmap can be repeated (True / False)
True for hydrodynamic models based on a tidal analysis

optimization_type: list of optimization types.
Excluding one or more not needed optimization types can significantly decrease the size of the preprocessing file

nodes_index: Numpy array that contains the indices of the nodes of the reduced
hydrodynamic model. nodes_index is the output of Roadmap.nodes_index. This option allows you to skip the node reduction step if this is already done.

calc_weights_time(*edge*, *i*, *j*, *vship*, *WD_min*, *WVPI*, *self_f*, *compute_cost*, *compute_co2*, *number_of_neighbor_layers*)

Function that retruns the weight of an arc

static compute_co2(*travel_time*, *speed*)

Default cost function for co2.

static compute_cost(*travel_time*, *speed*)

Default cost function for price.

static fifo_maker(*y*, *N1*)

Makes a FIFO time series from a Non-FIFO time series y: Time series N1: Mask file of the time series

abstract load()

load_hydrodynamic()

static nodes_on_land(*nodes*, *u*, *v*, *WD*)

Standard function that returns itself

parse()

halem.**HALEM_co2**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the least pollutant route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.**HALEM_cost**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the cheapest route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.**HALEM_func**(*start*, *stop*, *t0*, *vmax*, *Roadmap*, *costfunction*)

Base of the functions HALEM_time, HALEM_cost, HALEM_space, HALEM_co2. This function takes the pre-processing file, start location, stop location, departure time, and sailing velocity and returns the optimized route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties. costfunction Costfunction of the route optimization.

Roadmap.weight_time returns fastest route Roadmap.weight_space returns shortest route Roadmap.weight_cost returns cheapest route Roadmap.weight_co2 retruns least pollutant route

halem.**HALEM_space**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the shortest route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.**HALEM_time**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the fastest route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.**plot_timeseries**(*path*, *time*, *Roadmap*, *Color='r'*, *range_CP=5*)

This function can plot the time series for the route and shows a contourplot of the unsaiable areas of that route.

path: lon, lat coordinates of the route.
This is in the format of the output from halem.HALEM_func[0]

time: time series of the path.
This is in the format of the output from halem.HALEM_func[1]

Roadmap: Roadmap that is used to calculate the route. Color: Color of the plot of the time series.

Type sting, with matplotlib color

1.2 Contributing

Welcome to HALEM contributor's guide.

This document focuses on getting any potential contributor familiarized with the development processes, but other kinds of contributions are also appreciated.

If you are new to using git or have never collaborated in a project previously, please have a look at contribution-guide.org. Other resources are also listed in the excellent guide created by FreeCodeCamp[1].

Please notice, all users and contributors are expected to be **open, considerate, reasonable, and respectful**. When in doubt, Python Software Foundation's Code of Conduct is a good reference in terms of behavior guidelines.

1.2.1 Issue Reports

If you experience bugs or general issues with HALEM, please have a look on the issue tracker. If you don't see anything useful there, please feel free to fire an issue report.

Tip: Please don't forget to include the closed issues in your search. Sometimes a solution was already reported, and the problem is considered **solved**.

New issue reports should include information about your programming environment (e.g., operating system, Python version) and steps to reproduce the problem. Please try also to simplify the reproduction steps to a very minimal example that still illustrates the problem you are facing. By removing other factors, you help us to identify the root cause of the issue.

1.2.2 Documentation Improvements

You can help improve HALEM docs by making them more readable and coherent, or by adding missing information and correcting mistakes.

HALEM documentation uses Sphinx as its main documentation compiler. This means that the docs are kept in the same repository as the project code, and that any documentation update is done in the same way was a code contribution.

When working on documentation changes in your local machine, you can compile them using `tox`:

```
tox -e docs
```

and use Python's built-in web server for a preview in your web browser (`http://localhost:8000`):

```
python3 -m http.server --directory 'docs/_build/html'
```

[1] Even though, these resources focus on open source projects and communities, the general ideas behind collaborating with other developers to collectively create software are general and can be applied to all sorts of environments, including private companies and proprietary code bases.

1.2.3 Code Contributions

Submit an issue

Before you work on any non-trivial code contribution it's best to first create a report in the issue tracker to start a discussion on the subject. This often provides additional considerations and avoids unnecessary work.

Create an environment

Before you start coding, we recommend creating an isolated virtual environment to avoid any problems with your installed Python packages. This can easily be done via either `virtualenv`:

```
virtualenv <PATH TO VENV>
source <PATH TO VENV>/bin/activate
```

or Miniconda:

```
conda create -n halem python=3 six virtualenv pytest pytest-cov
conda activate halem
```

Clone the repository

1. Create an user account on GitHub if you do not already have one.
2. Fork the project repository: click on the *Fork* button near the top of the page. This creates a copy of the code under your account on GitHub.
3. Clone this copy to your local disk:

   ```
   git clone git@github.com:YourLogin/halem.git
   cd halem
   ```

4. You should run:

   ```
   pip install -U pip setuptools -e .
   ```

   to be able to import the package under development in the Python REPL.

5. Install `pre-commit`:

   ```
   pip install pre-commit
   pre-commit install
   ```

   `HALEM` comes with a lot of hooks configured to automatically help the developer to check the code being written.

Implement your changes

1. Create a branch to hold your changes:

```
git checkout -b my-feature
```

   and start making changes. Never work on the main branch!

2. Start your work on this branch. Don't forget to add docstrings to new functions, modules and classes, especially if they are part of public APIs.

3. Add yourself to the list of contributors in `AUTHORS.rst`.

4. When you're done editing, do:

```
git add <MODIFIED FILES>
git commit
```

   to record your changes in git.

   Please make sure to see the validation messages from `pre-commit` and fix any eventual issues. This should automatically use flake8/black to check/fix the code style in a way that is compatible with the project.

   Important: Don't forget to add unit tests and documentation in case your contribution adds an additional feature and is not just a bugfix.

   Moreover, writing a descriptive commit message is highly recommended. In case of doubt, you can check the commit history with:

```
git log --graph --decorate --pretty=oneline --abbrev-commit --all
```

   to look for recurring communication patterns.

   ---

5. Please check that your changes don't break any unit tests with:

```
tox
```

   (after having installed `tox` with `pip install tox` or `pipx`).

   You can also use `tox` to run several other pre-configured tasks in the repository. Try `tox -av` to see a list of the available checks.

Submit your contribution

1. If everything works fine, push your local branch to GitHub with:

```
git push -u origin my-feature
```

2. Go to the web page of your fork and click "Create pull request" to send your changes for review.

Troubleshooting

The following tips can be used when facing problems to build or test the package:

1. Make sure to fetch all the tags from the upstream repository. The command `git describe --abbrev=0 --tags` should return the version you are expecting. If you are trying to run CI scripts in a fork repository, make sure to push all the tags. You can also try to remove all the egg files or the complete egg folder, i.e., `.eggs`, as well as the `*.egg-info` folders in the `src` folder or potentially in the root of your project.

2. Sometimes `tox` misses out when new dependencies are added, especially to `setup.cfg` and `docs/requirements.txt`. If you find any problems with missing dependencies when running a command with `tox`, try to recreate the `tox` environment using the `-r` flag. For example, instead of:

   ```
   tox -e docs
   ```

   Try running:

   ```
   tox -r -e docs
   ```

3. Make sure to have a reliable `tox` installation that uses the correct Python version (e.g., 3.7+). When in doubt you can run:

   ```
   tox --version
   # OR
   which tox
   ```

   If you have trouble and are seeing weird errors upon running `tox`, you can also try to create a dedicated virtual environment with a `tox` binary freshly installed. For example:

   ```
   virtualenv .venv
   source .venv/bin/activate
   .venv/bin/pip install tox
   .venv/bin/tox -e all
   ```

4. Pytest can drop you in an interactive session in the case an error occurs. In order to do that you need to pass a `--pdb` option (for example by running `tox -- -k <NAME OF THE FALLING TEST> --pdb`). You can also setup breakpoints manually instead of using the `--pdb` option.

1.2.4 Maintainer tasks

Releases

If you are part of the group of maintainers and have correct user permissions on PyPI, the following steps can be used to release a new version for `HALEM`:

1. Make sure all unit tests are successful.
2. Tag the current commit on the main branch with a release tag, e.g., `v1.2.3`.
3. Push the new tag to the upstream repository, e.g., `git push upstream v1.2.3`
4. Clean up the `dist` and `build` folders with `tox -e clean` (or `rm -rf dist build`) to avoid confusion with old builds and Sphinx docs.
5. Run `tox -e build` and check that the files in `dist` have the correct version (no `.dirty` or git hash) according to the git tag. Also check the sizes of the distributions, if they are too big (e.g., > 500KB), unwanted clutter may have been accidentally included.

6. Run `tox -e publish -- --repository pypi` and check that everything was uploaded to PyPI correctly.

1.3 License

1.4 Credits

1.4.1 Development Lead

- Pieter van Halem

1.4.2 Collaboration

A special thanks to the following devellopers for contibuting to this package:

- Gerben de Boer
- Mark van Koningsveld
- Joris den Uijl
- Fedor Baart

1.5 History

1.5.1 1.0.0 (2022-08-31)

- Refactored app
- Added BaseRoadmap ABC
- Added Notebooks

1.5.2 0.3.1 (2019-08-08)

- Eight tag on GitHub
- clean-up
- Graduation version

1.5.3 0.3.0 (2019-08-08)

- Seventh tag on GitHub
- Released to Zenodo
- Documentation available on halem.readthedocs.io

1.5.4 0.2.0 (2019-06-27)

- Sixth tag on GitHub
- First mayor update

1.5.5 v0.1.0 (2019-05-21)

- Fifth tag on GitHub

1.5.6 v0.1.3 (2019-05-24)

- Fourth tag on GitHub

1.5.7 v0.1.2 (2019-05-22)

- Third tag on GitHub

1.5.8 v0.1.1 (2019-05-21)

- second tag on GitHub

1.5.9 v0.1.0 (2019-05-21)

- First tag on GitHub
- First release to PyPI

1.6 halem

1.6.1 halem package

Submodules

halem.functions module

halem.functions.**HALEM_co2**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the least pollutant route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.functions.**HALEM_cost**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the cheapest route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.functions.**HALEM_func**(*start*, *stop*, *t0*, *vmax*, *Roadmap*, *costfunction*)

Base of the functions HALEM_time, HALEM_cost, HALEM_space, HALEM_co2. This function takes the pre-processing file, start location, stop location, departure time, and sailing velocity and returns the optimized route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties. costfunction Costfunction of the route optimization.

Roadmap.weight_time returns fastest route Roadmap.weight_space returns shortest route Roadmap.weight_cost returns cheapest route Roadmap.weight_co2 retruns least pollutant route

halem.functions.**HALEM_space**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the shortest route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.functions.**HALEM_time**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the fastest route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.functions.**costfunction_spaceseries**(*edge*, *V_max*, *WD_min*, *flow*, *WVPI*, *L*, *tria*)

Function that returns the time series of the weights of a specifiv edge.

edge: (int) cosidered edge. edge: index of the location node
in Roadmap.nodes

V_max: Shipping velocity in deep water in meters per second WD_min: minimal needed draft in meters flow: Class that contains the hydrodynamic conditions WVPI: Weight of the vessel in tf L: (int) number of neighbouring layers. tria: triangulation of the nodes (output of scipy.spatial.Delaunay(nodes)

halem.functions.**costfunction_timeseries**(*edge*, *V_max*, *WD_min*, *flow*, *WVPI*, *L*, *tria*)

Function that returns the time series of the weights of a specific edge.

edge: (int) cosidered edge. edge: index of the location node
in Roadmap.nodes

V_max: Shipping velocity in deep water in meters per second WD_min: minimal needed draft in meters flow: Class that contains the hydrodynamic conditions WVPI: Weight of the vessel in tf L: (int) number of neighbouring layers. tria: triangulation of the nodes (output of scipy.spatial.Delaunay(nodes)

halem.functions.**find_neighbors**(*pindex*, *triang*)

Function that can find the neighbours of a Delauney mesh.

pindex: Index of the considered node. triang: Triangulation generated with scipy.spatial.Delaunay()

halem.functions.**find_neighbors2**(*index*, *triang*, *depth*)

Function that can find the neighbours of a Delauney mesh, for multiple layers of neighbours.

pindex: Index of the considered node. triang: Triangulation generated with scipy.spatial.Delaunay() Depth: Number of neigbouring layers (nb)

halem.functions.**haversine**(*coord1*, *coord2*)

use the Haversine function to determine the distance between two points in the WGS84 coordinate system. Returns the distance between the two points in meters. Source: https://janakiev.com/blog/gps-points-distance-python/

coord1: (lat, lon) coordinates of first point coord2: (lat, lon) coordinates of second point

halem.functions.**inbetweenpoints**(*start*, *stop*, *LL*, *tria*)

This node returns the nodes of influence for a specific arc. This function retruns the start and stop node plus the nodes in between the start and stop node. This function makes sure the route does not jump over hydrodynamic features when the neightbouring layers are higher than one.

start: (int) index of the start node stop: (int) index of the destination node LL: (int) number of neighbouring layers. tria: triangulation of the nodes (output of scipy.spatial.Delaunay(nodes)

halem.functions.**plot_timeseries**(*path*, *time*, *Roadmap*, *Color='r'*, *range_CP=5*)

This function can plot the time series for the route and shows a contourplot of the unsaiable areas of that route.

path: lon, lat coordinates of the route.
This is in the format of the output from halem.HALEM_func[0]

time: time series of the path.
This is in the format of the output from halem.HALEM_func[1]

Roadmap: Roadmap that is used to calculate the route. Color: Color of the plot of the time series.

Type sting, with matplotlib color

halem.functions.**squat**(*h*, *T*, *V_max*, *LWL*, *WWL*, *ukc*, *WVPI*)

Function for reducing the sailing velocity in deep water to the sailing velocity in shallow unconfined waters.

h: Array of the water depth in meters V_max: Sailing velocity in deep water in meters per second WWL: Width over Water Line of the vessel in meters LWL: Length over Water Line of the vessel in meters ukc: Minimal needed under keel clearance in meters. T: numpy array with the draft of the vessel. Numpy

array has the shape of the number of discretisations in the dynamic sailing velocity in meters

WVPI: total weight of the the vessel in tf

V: Array of sailing velocities reduced for squat,
corresponding to the input arrat h.

halem.path_finder module

class halem.path_finder.**PathFinder**(*start*, *stop*, *Roadmap*, *t0*, *graph_functions*)

Bases: `object`

This class contains the code for calculating the optimal route from the Roadmap

start: start location (lat, lon) stop: destination location (lat, lon) Roadmap: Preprocessing file graph_functions: class that selects the correct weights from the Roadmap.

dijsktra(*Roadmap*, *initial*, *end*, *t0*, *graph_functions*)

find_k_repeat(*t*, *ts*)

find_k_time(*t*, *ts*)

find_startstop(*start*, *nodes*)

halem.roadmap module

class halem.roadmap.**BaseRoadmap**(*number_of_neighbor_layers*, *vship*, *WD_min*, *WVPI*, *repeat=False*, *WWL=20*, *LWL=80*, *ukc=1.5*, *optimization_type=None*, *nodes_index=None*, **args*, ***kwargs*)

Bases: ABC, *NodeReduction*

Absctract Base class for the Roadmap.

Pre-processing function for the HALEM optimizations. In this fucntion the hydrodynamic model and the vessel properties are transformed into weights for the Time dependend Dijkstra function.

number_of_neighbor_layers: number of neigbouring layers for which edges are
created. increasing this number results in a higher directional resolution.

vship: (N (rows) * M (columns)) numpy array that indicates the sailing velocity
in deep water. For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity. For the optimization type cost and co2 N must be larger or equal to 2.

WD_min: numpy array with the draft of the vessel.
Numpy array has the shape of the number of discretisations in the dynamic sailing velocity

WVPI: Numpy array with the total weight of the vessel.

WWL: Width over Water Line of the vessel in meters

LWL: Length over Water Line of the vessel in meters

ukc: Minimal needed under keel clearance in meters.

repeat: Indicator if the roadmap can be repeated (True / False)
True for hydrodynamic models based on a tidal analysis

optimization_type: list of optimization types.
Excluding one or more not needed optimization types can significantly decrease the size of the preprocessing file

nodes_index: Numpy array that contains the indices of the nodes of the reduced
hydrodynamic model. nodes_index is the output of Roadmap.nodes_index. This option allows you to skip the node reduction step if this is already done.

calc_weights_time(*edge*, *i*, *j*, *vship*, *WD_min*, *WVPI*, *self_f*, *compute_cost*, *compute_co2*, *number_of_neighbor_layers*)

Function that retruns the weight of an arc

static compute_co2(*travel_time*, *speed*)

Default cost function for co2.

static compute_cost(*travel_time*, *speed*)

Default cost function for price.

static fifo_maker(*y*, *N1*)

Makes a FIFO time series from a Non-FIFO time series y: Time series N1: Mask file of the time series

abstract load()

load_hydrodynamic()

static nodes_on_land(*nodes*, *u*, *v*, *WD*)

Standard function that returns itself

parse()

class halem.roadmap.**Graph**

Bases: object

class that contains the nodes, arcs, and weights for the time-dependent, directional, weighted, and Non-FIFO graph of the route optimization problem. This class is used multiple times in the halem.mesh_maker.GraphFlowModel() function

add_edge(*from_node*, *to_node*, *weight*)

class halem.roadmap.**NodeReduction**(*dx_min*, *blend*, *nl*, **args*, ***kwargs*)

Bases: object

This class can reduce the number of gridpoints of the hydrodynamic model. This is done based on the vorticity and the magnitude of the flow. The nodes are pruned based on a length scale. The formula for this length scale is: LS / min = (1+|×u|)^c+(1)(1+|u|)^m. With: LS = resulting length scale, = blend factor between the curl and the magnitude method, min = minimal length scale, c = non linearity parameter for the method with the curl of the flow, m = non linearity parameter for the method with the magnitude of the flow, and u = the velocity vector of the flow.

flow: class that contains the hydrodynamic properties.

class must have the following instances. u: numpy array with shape (N, M) v: numpy array with shape (N, M) WD: numpy array with shape (N, M) nodes: numpy array with shape (N, 2) (lat, lon) t: numpy array with shape M (seconds since 01-01-1970 00:00:00) tria: triangulation of the nodes (output of scipy.spatial.Delaunay) in which N is the number of nodes of the hydrodynamic model, and M is the number of time steps of the hydrodynamic model

dx_min: float, minimal spatial resolution.

Parameter of the lengt scale function concerning the node reduction

blend: blend factor between the verticity and magnitude of the flow.

Parameter of the lengt scale function concerning the node reduction

nl: float (nl_c, nl_m)

Non linearity factor consisting out of two numbers nl_c non-linearity factor for the corticity, nl_m non-linearity factor for the magnitude of the flow. Parameter of the lengt scale function concerning the node reduction

number_of_neighbor_layers: number of neigbouring layers for which edges are

created. increasing this number results in a higher directional resolution.

static closest_node(*node*, *nodes*, *node_list*)

Finds the closest node for a subset of nodes in a set of node.

based on WGS84 coordinates.

node: considered node nodes: indices of the subset node_list: total list of the nodes

curl_func(*node*)

Determine the curl of the grid.

get_nodes()

Reduce the number of gridpoints of the hydrodynamic model.

length_scale(*node*)

Determine the lengthscale of the grid.

slope(*xs*, *ys*, *zs*)

Function for the slope of a plane in x and y direction. Used to calculate the curl of the flow for the node reduction step

Module contents

class halem.**BaseRoadmap**(*number_of_neighbor_layers*, *vship*, *WD_min*, *WVPI*, *repeat=False*, *WWL=20*, *LWL=80*, *ukc=1.5*, *optimization_type=None*, *nodes_index=None*, **args*, ***kwargs*)

Bases: ABC, *NodeReduction*

Absctract Base class for the Roadmap.

Pre-processing function for the HALEM optimizations. In this fucntion the hydrodynamic model and the vessel properties are transformed into weights for the Time dependend Dijkstra function.

number_of_neighbor_layers: number of neigbouring layers for which edges are
created. increasing this number results in a higher directional resolution.

vship: (N (rows) * M (columns)) numpy array that indicates the sailing velocity
in deep water. For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity. For the optimization type cost and co2 N must be larger or equal to 2.

WD_min: numpy array with the draft of the vessel.
Numpy array has the shape of the number of discretisations in the dynamic sailing velocity

WVPI: Numpy array with the total weight of the vessel.

WWL: Width over Water Line of the vessel in meters

LWL: Length over Water Line of the vessel in meters

ukc: Minimal needed under keel clearance in meters.

repeat: Indicator if the roadmap can be repeated (True / False)
True for hydrodynamic models based on a tidal analysis

optimization_type: list of optimization types.
Excluding one or more not needed optimization types can significantly decrease the size of the preprocessing file

nodes_index: Numpy array that contains the indices of the nodes of the reduced
hydrodynamic model. nodes_index is the output of Roadmap.nodes_index. This option allows you to skip the node reduction step if this is already done.

calc_weights_time(*edge*, *i*, *j*, *vship*, *WD_min*, *WVPI*, *self_f*, *compute_cost*, *compute_co2*, *number_of_neighbor_layers*)

Function that retruns the weight of an arc

static compute_co2(*travel_time*, *speed*)

Default cost function for co2.

static compute_cost(*travel_time*, *speed*)

Default cost function for price.

static fifo_maker(*y*, *N1*)

Makes a FIFO time series from a Non-FIFO time series y: Time series N1: Mask file of the time series

abstract load()

load_hydrodynamic()

static nodes_on_land(*nodes*, *u*, *v*, *WD*)

Standard function that returns itself

parse()

halem.**HALEM_co2**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the least pollutant route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.**HALEM_cost**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the cheapest route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.**HALEM_func**(*start*, *stop*, *t0*, *vmax*, *Roadmap*, *costfunction*)

Base of the functions HALEM_time, HALEM_cost, HALEM_space, HALEM_co2. This function takes the pre-processing file, start location, stop location, departure time, and sailing velocity and returns the optimized route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties. costfunction Costfunction of the route optimization.

Roadmap.weight_time returns fastest route Roadmap.weight_space returns shortest route Roadmap.weight_cost returns cheapest route Roadmap.weight_co2 retruns least pollutant route

halem.**HALEM_space**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the shortest route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.**HALEM_time**(*start*, *stop*, *t0*, *vmax*, *Roadmap*)

Implementation of the function HALEM_func() for the fastest route.

start: (lon, lat) coordinates of the start location stop: (lon, lat) coordinates of the destination location t0: string that indcates the departure time

('day'/'month'/'year' 'hour':'minute':'seconds')

vmax: (N (rows) * M (columns)) numpy array that indicates the sailing velocity in deep water.

For which N is the number of discretisations in the load factor, and M is the number of discretisations in the dynamic sailing velocity

For the optimization type cost and co2 N must be larger or equal to 2.

Roadmap: Preprocessing file that contains the hydrodynamic properties.

halem.**plot_timeseries**(*path*, *time*, *Roadmap*, *Color='r'*, *range_CP=5*)

This function can plot the time series for the route and shows a contourplot of the unsaiable areas of that route.

path: lon, lat coordinates of the route.
This is in the format of the output from halem.HALEM_func[0]

time: time series of the path.
This is in the format of the output from halem.HALEM_func[1]

Roadmap: Roadmap that is used to calculate the route. Color: Color of the plot of the time series.

Type sting, with matplotlib color

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX