D3.2 Model for Recommended Routing Corridors

Project Number	
Classification	Public
Deliverable No.	D3.2
Work Package(s)	WP3
Document Version	1.0
Issue Date	December 31, 2019
Document Timescale	Project Start: June/July 1st (Poland), 2018
Final version due	2019-12-31
Compiled by	PUEB
Authors	Dominik Filipiak, Krzysztof Węcel, Milena Stróżyna, Błażej Lisiecki, Witold Abramowicz
Issue Authorisation	Sebastian Feuerstack, OFF

HANSA is funded by the MarTERA partners German Federal Ministry of Economic Affairs and Energy (BMWi), Polish National Centre for Research and Development (NCBR) and Research Council of Norway (RCN) and co-funded by the European Union.

DISTRIBUTION LIST

Copy type[1]	Company and Location	Recipient
T	HANSA Consortium	all HANSA Partners

RECORD OF REVISION

Date	Status Description	Author
2019-12-31	First draft	PUEB

Table of Contents

1 Tools

This document presents the code of the methods and tools developed in the HANSA project related to the generation of the mesh and recommended corridors as well as the web service for providing the results of these methods to external systems and applications. The code itself is available in the separate file. The code is organized in several folders, reflecting the repositories in which the code was developed.

1.1 Apache Spark Docker Image

The repository *HANSA-Docker-SparkJupyter* has already been described in Deliverable D2.2 but it was updated with new kernel for `pyspark`. The updates resulted from the needs of analyses, where new Python packages were added.

The repository contains a docker image used as an Apache Spark container within the project. It is used mainly in the process of the development of the recommended corridors and mesh generation. Heavily exploited for analysis in Jupyter notebooks.

It contains the following key files:

- **Dockerfile** The first file describes the docker image – it is based on the jupyter/pyspark-notebook image.
- **docker-compose.yml** The second file describes the service. In this file, high-level features are set, such as Apache Spark options, external libraries, or available ports.
- **kernel.json** The file defining pyspark kernel, which is used to direct Spark to use avro, not available in a default image.

1.2 HANSA API

The repository *HANSA-API* is a new development within the reported period.

The repository contains the flask-based wrapper for exposing API, along with OpenAPI documentation.

It contains the following key files:

- **Dockerfile** The docker image for wrapping the API.
- **build_docker.sh** The command to build the docker with API.
- **docker-compose.yml** The docker compose script as a shortcut for running all necessary containers.

- **app.py** Main Python file that is run within a container. It contains all documentation of the API rendered later as OpenAPI.
- **run_hansa_api.sh** The command used to start the docker with HANSA API.
- **run_hansa_neo4j.sh** The command used to start the Neo4j necessary to answer API calls concerning the mesh.
- **test_corridor.bat**, **test_mesh.bat** Sample requests using `curl`
- **test*.json** JSON files containing specific parameters for test requests

1.3 Genetic Algorithm for Waypoint Selection

The repository *HANSA-GeneticAlgorithm* is a new development within the reported period.

The repository contains the implementation of GA for maritime routing.

It contains mostly the Jupyter notebooks. The following are key files:

- **1_WaypointDiscovery.ipynb** The notebook presenting the process of waypoint discovery.
- **1a_WaypointDiscoveryPlots.ipynb** The notebook with visualizations illustrating the approach and its validation.

All other notebooks are deprecated and left in the repository for reference. For example, when describing the evolution of the approach and the optimization it was a useful source of old efforts. The detailed description of the genetic algorithm is provided in the deliverable D3.1.

1.4 CUSUM

The repository *HANSA-CUSUM* is a new development within the reported period.

The repository contains reimplementation of the CUSUM approach in a big data context, using Apache Spark. For the development, both Python and Scala are used, what is emphasized by internal structure of the repository. Scripts contained here were used to deliver data used later by one of the waypoint and mesh discovery variants. We recommend using only code from CUSUM_exec_v2.ipynb (Scala).

It contains the following key files:

- **CUSUM/scala/src/implementation/cusumExecution.scala** The main execution program in Scala, for batch processing.
- **CUSUM/scala_spark/CUSUM_exec_v2.ipynb** The Jupyter notebook demonstrating step by step implementation in scala.

- **CUSUM/python/cusum_calculations/cusum_exec.py** The main execution script in Python. Several supportive files are located in the neighboring directories.

1.5 Generation of the Mesh

The repository *HANSA-Graph* is a new development within the reported period. It was partly derived from HANSA-GeneticAlgorithm to focus purely on mesh generation, provided that waypoints are given.

The repository contains files related to generation of a mesh based on waypoints and AIS/CUSUM data. All implementation work is done in Python. Jupyter notebooks demonstrate the approach step by step and contain some remarkable results. Python scripts are used for batch processing. For example, on server in the folder visualization-30000 we have generated 62 files with meshes both for the Baltic + the North Sea area and the German Bight area, for different types of ships, including various configurations for waypoint generation (to fulfill the requirement for hyperparameter tuning).

It contains the following key Jupyter notebooks:

- **1_v3_AIS enrichment-kNN approach-Copy1-german bight.ipynb** The most up-to-date Jupyter notebook with all experiments conducted for elaboration of the most efficient method of *AIS enrichment*. The area of the German Bight is considered, hence the name.
- **2_v3_Waypoints to Mesh-Copy1-german bight.ipynb** The most up-to-date Jupyter notebook containing various approaches to generation of edges, as described in Deliverable D3.1.

There are also files prepared for batch processing that should be executed in the order given below. Please note that scripts check if the output file already exist, thus skipping unnecessary processing (hence `iteration` in name). So, they can be run multiple times when an error occurs. The side effect is that for reprocessing we need to explicitly remove the output files, e.g. folders with CSVs.

The following scripts perform the batch processing:

- **enrich-ais-iteration.py** The Python script that takes AIS data on input and produces enriched AIS data, i.e. with the closest waypoints assigned. Selection of waypoints is based on name convention agreed in the project.
- **build-graph-iteration.py** The Python scripts that takes enriched AIS data and produces meshed/-graphs. They consists of two files: nodes and edges. They can be then imported to Neo4j.
- **visualize-baltic-iteration.py** The Python script responsible for producing maps of the meshes. As there are two separate areas, this file is responsible for the Baltic + the North Sea area.
- **visualize-german-iteration.py** The Python script responsible for producing maps of the meshes. As there are two separate areas, this file is responsible for the German Bight area.

1.6 Import mesh into Neo4j

The repository *HANSA-neo4j* [`https://github.com/KIE-UEP/HANSA-neo4j`] is a new development within the reported period.

The repository contains files related to import of mesh files into a Neo4j database. It uses built-in algorithms like Dijkstra to serve API with shortest or fastest paths.

It contains the following key files:

- **run_hansa_neo4j.sh** The shell script used to start a docker with Neo4j. The running database is a prerequisite for import.
- **hansa_graph** The Python package containing routines for importing mesh into a database. To be installed with `setup.py`.

Plugin installation

To install APOC Procedures and Neo4j Graph Algorithms just copy files `apoc-3.5.0.4-all.jar` and `neo4j-graph-algorithms-3.5.7.0-standalone.jar` to Neo4j's `/plugins` directory, add line `dbms.security.procedures.unrestricted=algo.*,apoc.*` to `neo4j.conf` file and restart Neo4j server.