COMP6203 Intelligent Agents
Lab Instructions

Jan Buermann
edited by Jessica Newman (jln1g19@soton.ac.uk)

2025

Contents

Lab #1

1.1 Introduction

Imagine you are running a maritime shipping company that transports oil. You have a fleet of tankers around the world and you are looking for customers who want their oil to be transported from one port to another. Currently, the possible opportunities to transport cargoes, i.e. the oil, are as presented in Figure 1.

The refinery at Fawley (near Southampton) is looking for the cheapest shipper to transport two cargoes: one to Dublin and one to Rotterdam. You and your main competitor both have a vessel in Southampton and you now need to decide what price you would offer the refinery for the transportation of either cargo.

One option is to calculate how much it will cost you to transport either cargo and offer that as your price. Mind that, if you offer to transport both and you will get the contract for both you will have to come back to Southampton once you have dropped of the first.

Another consideration is that you know that soon another company will be looking for a shipping company to transport cargo from Rotterdam to Felixstowe. Hence, if you would get the contract to transport the cargo from Fawley to Rotterdam and the contract to transport the cargo from Rotterdam to Felixstowe, you can directly continue with transporting cargoes without travelling empty from one port to another.

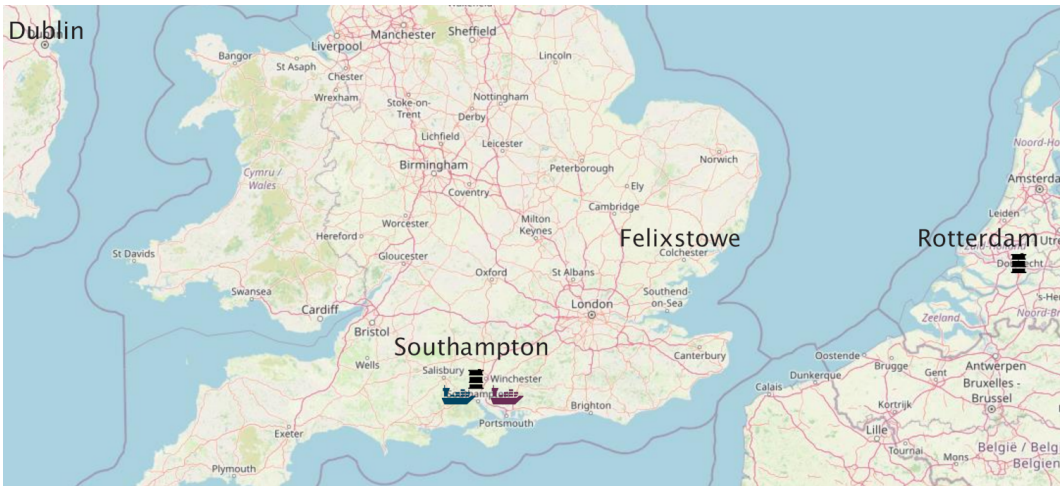

Figure 1: A maritime example.

This example highlights the general setting of the tramp trade maritime shipping which we will consider in the labs as well as in the coursework. The labs will guide you to understand how this setting is reflected in our simulator MABLE and develop a simple shipping company agent that will bid for cargo opportunities and schedules the

transportation of those cargoes they win the contracts for. In any setting including the competition at the end of the coursework, all shipping companies will have the option to bid for cargo opportunities at cargo auctions at regular intervals. The shipping companies will have to decide how much to bid at these reverse second-price auctions considering their already existing transport commitments, the requirement to transport all won contracts, some knowledge of future auctions and knowledge of the competitors' fleets.

1.2 Task 1: A first look at Mable

MABLE is an agent-based maritime cargo transportation simulator written in python which only needs a few steps to be set up. The simulator allows to specify a setting with ports, a cargo market with one homogeneous continuous cargo, e.g. oil, and shipping companies. Once such a setting is specified and run, the simulator generates random cargo transportation opportunities that shipping companies can bid for to transport and then have to be transported from one port to another. For the labs and the coursework the world with the ports and the cargo market are already specified and all you need to do is define the behaviour of a shipping company.

1.2.1 Good to know about MABLE

There are a few points about MABLE that it is good to be aware of.

Firstly, MABLE was developed following the object-oriented programming paradigm, meaning that everything is an object, e.g. vessels, ports, cargoes. If you are new to the concept or unsure about how this looks like in python there are courses on LinkedIn learning.

Secondly, MABLE is an event based simulator, meaning that time progresses in steps by events like a cargo auction happens, a vessel reaches a port or a vessel has transferred cargo. This will not affect your work as you will only need to specify in what order you vessels transport cargoes and MABLE will automatically translate that into events.

1.2.2 MABLE API

MABLE's API is accessible under MABLE API.

1.2.3 Setting up MABLE

MABLE requires at least Python version 3.11. For guidance on downloading and installing Python, go to `https://wiki.python.org/moin/BeginnersGuide/Download`.

MABLE is indexed on PyPi, so if you are running a version of Python >=3.11 you should be able to install MABLE using pip with the following command:

```
pip install mable
```

If this doesn't work you can also download MABLE manually:

- Download the simulator wheel `mable-0.0.13.post3-py3-none-any.whl`
- Install MABLE in your local python, e.g.

  ```
  pip install mable-0.0.13.post3-py3-none-any.whl
  ```

To run MABLE, you will need to download the resources file `mable_resources.zip` (alternate download link) and place this in the working directory of your python project.

1.3 Task 2: Running a Maritime Setting

To implement your own shipping company you will have to populate a subclass of `mable.cargo_bidding.TradingCompany`. We will start with a simple setup with an empty shipping company that just does the default behaviour.

1.3.1 Simple setup

Getting started with a running example is fairly straightforward with the provided example functions. We will be creating the following frame.

```
from mable.cargo_bidding import TradingCompany
from mable.examples import environment, fleets

class MyCompany(TradingCompany):
    pass

if __name__ == '__main__':
    specifications_builder = environment.
    ↪  get_specification_builder(environment_files_path=".")
    fleet = fleets.example_fleet_1()
    specifications_builder.add_company(MyCompany.Data(MyCompany, fleet,
    ↪  "My Shipping Corp Ltd."))
    sim = environment.generate_simulation(specifications_builder)
    sim.run()
```

The steps for that are the following.

1. Create a new python file.
2. Start with the imports.
   a) Import `TradingCompany` from `mable.cargo_bidding` to load the shipping company base class

b) Import `environment` and `fleets` from `mable.examples`. These allow you to quickly set up a default trading environment and predefined fleets of vessels.

3. Create the shipping company class by creating a class with a name of your choosing ,e.g. `MyCompany`, that is a subclass of `TradingCompany`. We will leave the class definition empty, using the the default implementation of the `TradingCompany` class (The default implementation can be seen here - don't worry if you don't understand the functions defined in here, as we will be going over them in future labs!)

4. A simulation is created via adding details to a *specifications_builder*. `mable.examples.environment.get_specification_builder()` returns an instance of such a builder and takes care of further setup like linking the auxiliary files. The function has one parameter `environment_files_path` which allows the specification of the directory of the resources file (the `mable_resources.zip`). The default is the current directory where the script will be executed (".").

5. Generate a sample fleet with `mable.examples.fleets.example_fleet_1()`

6. Use the `specifications_builder` and the `add_company` function to add your company.
   - To define a company you need to specify the class, the fleet and the name of the company.
   - The function expects a `DataClass` object which is automatically provided via `MyCompany.Data`. Hence you can create the company specification via `MyCompany.Data(MyCompany, fleet, "My Shipping Corp Ltd.")`

7. This specifies a simulation which can be created for the *specifications_builder* using the `environment.generate_simulation` function which returns a simulation object. Hence,

   ```
   sim = environment.generate_simulation(specifications_builder)
   ```

   provides the simulation.

8. Finally, you can run the simulation by calling the simulation's `run` function, i.e. `sim.run()`.

1.3.2 Run the maritime Setting

Once you have created a simple setup as describe in Section 1.3.1 you can run the file to start the simulation. While the script is running it should print to the console events that are happening. The run should finish with the log message `---Run Finished---`. The run will produce a file that contains information about the companies' operation and the outcomes of the cargo auctions, called `metrics_competition_<number>.json` where <*number*> is a random id. To get a quick overview of the income, MABLE comes with a little script to print this information to the console. Simply call

```
mable overview metrics_competition_<number>.json
```

which should produce an output like the following.

```
Company My Shipping Corp Ltd.
+---------+---------+
|   Name  |   Value |
+---------+---------+
|   Cost  |    1000 |
| Penalty |       0 |
| Revenue |    1100 |
+---------+---------+
|  Income |     100 |
+---------+---------+
```

1.3.3 What is the code doing?

Take a look through the MABLE documentation here. See if you can find some of the classes we used in the example code, and consider what their function might be.

A python *.whl* file is actually a ZIP-format archive with a specially formatted file name. This means you can see the source code for MABLE by unzipping the `mable-0.0.13.post3-py3-none-any.whl` file available here, or by looking at the source files in the MABLE github repository. If you are unsure of how a certain class or method works, it can be very helpful to look at the source code!