Maritime Shipping Competition (MSC) Getting Started Guide

MSC Team

2024

Contents

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.

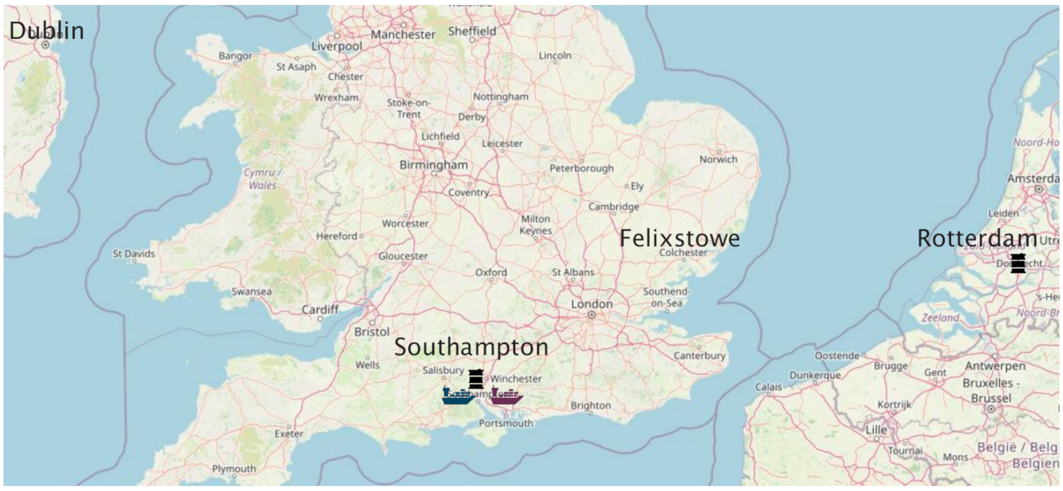

Figure 1: A maritime example.

This example highlights the general setting of the tramp trade maritime shipping which is the basis of the Maritime Shipping Competition (MSC). This guide will help you to understand how this setting is reflected in our simulator MABLE and what are the important functions to 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 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.

2 A first look at Mable

Maritime Agent-Based Logistics Emulator (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 competition 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.

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.

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.

2.2 MABLE API

MABLE's API is accessible under MABLE API.

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`.

To get start install MABLE and the auxiliary files:

- Install Mable
  - Install MABLE via PyPI
  - Install MABLE manually
    - Download the simulator wheel via the GitHub repository.
    - Install MABLE in your local python, e.g.

```
pip install MABLE-<X.Y.Z>-py3-none-any.whl
```

- Download the resources file `mable_resources.zip` LINK

3 Running a Maritime Setting

To implement your own shipping company you will have to populate a subclass of `mable.cargo_bidding.TradingCompany`. The following example highlights setting up the simulator frame if you want to start with a complete frame see Section 13

3.1 Simple setup

Getting started with a running example is fairly straightforward with the functions provided in MABLE. Consider the following frame.

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

class Companyn(TradingCompany):
    pass

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

The parts are the following.

1. Starting with the imports.
   a) The import `TradingCompany` from `mable.cargo_bidding` loads the shipping company base class
   b) The import `environment` and `fleets` from `mable.examples` allow you to quickly set up a default trading environment and predefined fleets of vessels.
2. The shipping company class needs to be a subclass of `TradingCompany`.
3. 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 (".").

4. A sample can be created with `mable.examples.fleets.example_fleet_1()`. For a more conplex fleet see `mixed_fleet` (see Section 13).

5. A company is added using the `specifications_builder` and the `add_company` function.
   - To define a company the class, the fleet and the name of the company need to be specify.
   - The function expects a `DataClass` object which is automatically provided via `MyCompany.Data`. Hence, company specifications can be created via `MyCompany.Data(MyCompany, fleet, "My Shipping Corp Ltd.")`

6. 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.

7. Finally, a simulation is run by calling the simulation's `run` function, i.e. `sim.run()`.

3.2 Run the maritime Setting

A setting as describe in Section 3.1 can be run to start a 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 |
+---------+----------+
```

4 The World

The world or environment in MABLE is a shipping network of connected ports. Each port has a name and a location consisting out of a latitude and longitude. (While you won't have to interact with ports directly, ports are represented by `class LatLongPort` with latitude and longitude in decimal degrees.)

4.1 Transportation Opportunities (TimeWindowTrade)

Cargo transportation opportunities the companies can bid for are represented by `class TimeWindowTrade` (see Figure 2). Each of these has specifies a port from which the cargo has to be picked up, `origin_port`, and a port at which the cargo has to be dropped off, `destination_port`. Moreover, the trade specifies what `amount` of which type (`cargo_type`) has to be transported.

There will be only one `cargo_type` which is `"Oil"`. However, you can still use the property whenever you need to pass the type to a different function.

Additionally, the trade may specify up to four time points which indicate restrictions on the time when the cargo can be picked up and dropped off.

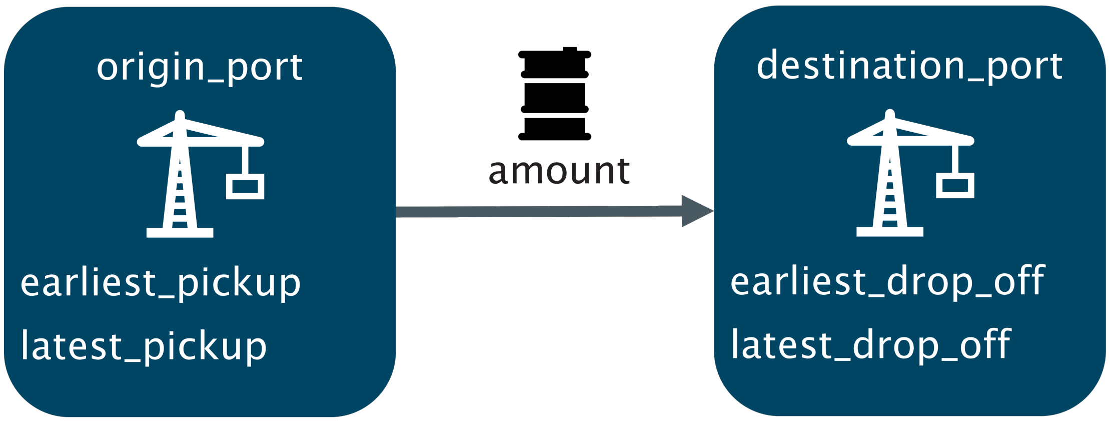

Figure 2: Illustration of a time window trade.

- `earliest_pickup` When the vessel arrives earlier it will have to wait.
- `latest_pickup` When the vessel arrives later it will not be able to pick up the cargo, i.e. fulfil the job.
- `earliest_drop_off` When the vessel arrives earlier it will have to wait.
- `latest_drop_off` When the vessel arrives later it will not be able to drop off the cargo, i.e. fulfil the job.

If a time is not set, i.e. no restrictions on arrival, this is indicated by None for the respective time.

You can ignore all other instance variables of `class TimeWindowTrade`: `probability` and `status`. All trades will happen.

4.2 Vessels (VesselWithEngine)

Cargoes are transported using vessels (ships) which have a cargo hold to store the cargo and and engine that burns fuel. Each company has a number of vessels whose cargo hold size and fuel consumption might be different. The cargo hold size restricts the amount the vessel can transport. The fuel consumption depends on what the vessel is doing but even when the vessel is idling the vessel consumes fuel. There are five different states or operations of what the vessel could be doing (see Figure 3), each have their own rate of consumption which is an amount of fuel per time:

- Idling, i.e. doing none of the other operations.
- Loading, i.e. loading cargo within a port.
- Unloading, i.e. unloading cargo within a port.
- Laden, i.e. moving between ports with cargo in the hold.
- Ballast, i.e. moving between ports with no cargo in the hold.

The class you will have to interact with in MABLE is the `class VesselWithEngine` which models a cargo hold and an engine that consumes fuel. VesselWithEngine (including via its ancestors[1]) has a lot of convenience methods to calculate capacity, loading rate, times for loading, consumptions and costs.

[1] `WorldVessel`, `SimpleVessel` and `Vessel`

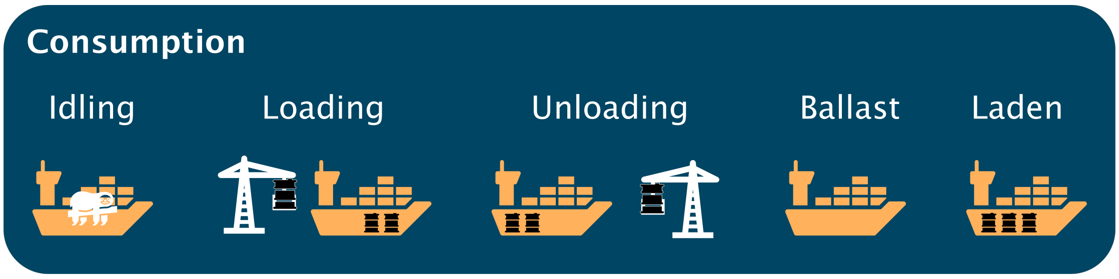

Figure 3: Vessel statuses.

5 The Market

Periodically an auction will be run at which several transportation opportunities (see Section 4.1) will be on offer. All companies can bid to transport any or all of the available trades by offering a price per trade they would like to get paid for the transportation of the particular trade. The market will collect all bids and for each trade will run a second cost reverse auction to determine the winner and the payment to the winning company. Cargoes that have been awarded to a company have to be transported by that company. Otherwise, the company will be fined.

The market will also inform campaniles of upcoming auctions beforehand. Just before an auction - the current auction - the market will inform all companies what trades will be available at the auction that will follow in the next period after the current auction.

6 Running a Company

For the companies to interact with the market there is an interface of three functions: `pre_inform`, `inform` and `receive`.

The market will use the `pre_inform` function to inform the companies of upcoming trades at a later time. The call will contain the trades that will be auctioned off at that time as well as the time when this will happen. Following this, the market will hold an auction for the trades in the current auction round. The market will use the `inform` function to tell the companies which trades are auctioned off. The expectation is that the companies return a list of all bids they want to place as a list of `class Bid`. The market will use all bids from all companies to determine the winners for all trades in the current auction round. Subsequently, the market will call `receive` to inform every company of the trades they won in the auction. (The call will also contain the information about all auction outcomes which may be of interest to opponent modelling.)

If you make changes to `receive` make sure that you always call `apply_schedules` with the schedules you want your vessel to perform. Otherwise, your company's transportations will not be scheduled and your company will get fined for not fulfilling its contracts.

7 Schedules

A schedule is the plan of operation for a single vessel which captures the order of when the cargo of a trade gets picked up and when the cargo of the trades gets dropped off. You can think of a schedule as a simple sequence of pick-up and drop-off events. For example a vessel that is supposed to transport two trades, $trade_1$ and $trade_2$, might transport both of them in order or pick up both of them first and then drop them off. This results in six possible combinations.

- [Pick up $trade_1$, Drop off $trade_1$, Pick up $trade_2$, Drop off $trade_2$]
- [Pick up $trade_2$, Drop off $trade_2$, Pick up $trade_1$, Drop off $trade_1$]
- [Pick up $trade_1$, Pick up $trade_2$, Drop off $trade_1$, Drop off $trade_2$]
- [Pick up $trade_1$, Pick up $trade_2$, Drop off $trade_2$, Drop off $trade_1$]
- [Pick up $trade_2$, Pick up $trade_1$, Drop off $trade_1$, Drop off $trade_2$]
- [Pick up $trade_2$, Pick up $trade_1$, Drop off $trade_2$, Drop off $trade_1$]

The `class Schedule` provides functionality to plan schedules, including adding to and verifying schedules.

8 Creating Schedules

For the scheduling of trades, the default implementation of `inform` calls the `propose_schedules` function of the company object and expects the function to return a list of trades that can be transported, the proposed schedules to do this and the estimated costs for each cargo. You can override this function `def propose_schedules` but be aware that the `propose_schedules` specifies the return value, consisting of the list of trades, the proposed schedules and the estimated costs, to be wrapped in an instance of the `class ScheduleProposal` data class.

8.1 Building a Schedule

To build a schedule the following functions already provided by MABLE to help you.

- `def fleet` Your company class has a `fleet` property which gives you a list of your company's fleet of vessels.

- `def schedule` Every vessel has a `schedule` property to get a copy of its current schedule. Do not use `Schedule` directly to create new schedules. They will not be set up correctly and you will encounter errors.

- `def add_transportation` `add_transportation` allows you to add a trade to a schedule. If the function is called specifying only the trade, the function will append the trade to the end of the schedule.

- `def verify_schedule` Once you have constructed a schedule, you can use the function `verify_schedule` to verify that the schedule can be fulfilled. Fulfilment, is based on two factors, the ability to deliver on time and the condition that the cargo hold size is not violated.
  - All cargoes are delivered on time if all cargoes are picked up and dropped off within the time windows specified by the trades. This can also be tested individually by using the `def verify_schedule_time` function.
  - The cargo hold size is not violated if the vessel is never overloaded, i.e. the cargo is not exceeding the capacity of the cargo hold, never underloaded, i.e. trying to unload cargo when the vessel is empty. This can also be tested individually by using the `def verify_schedule_cargo` function.

- `def copy` For convenience the `Schedule` has a `copy` function which gives you a deep copy of the schedule, i.e. you can change the copy without affecting the old schedule. Hence, you can create different alternatives without make new variants of the schedule without messing up the original schedule. You might not need this function right now but remember it for later!

- The `Schedule` has a function `get_insertion_points` which allows you to get all valid locations that can be used for the `location_pick_up` and `location_drop_off` of the `add_transportation` function. You could use this to determine if instead of just appending a trade to the schedule you could do the pick up and drop off earlier similar to some of the combinations in Section 7. Hence, do the following.

`def inform` will simply take the list of scheduled trades and for now forward them with a zero bid. Its implementation looks like this.

```
        def inform(self, trades, *args, **kwargs):
            proposed_scheduling =
             ↪  self.propose_schedules(trades)
            scheduled_trades =
             ↪  proposed_scheduling.scheduled_trades
            bids = [Bid(amount=0, trade=one_trade) for
             ↪  one_trade in scheduled_trades]
            return bids
```

The `class Schedule` defines a range of other functions to interact with the schedule which are used by the simulation: `def next`, `def pop`, `def get`, `def __getitem__` (the dunder to allow the syntactic sugar access via square brackets, i.e. `some_schedule[i]`). You should **not** use these! `Schedule` does a lot of 'under-the-hood' work to stablish, maintain and verify schedules which could be messed up when you do not use these function in the right way.

9 Creating Bids

Once a schedule has been proposed transportation costs can be calculates to inform the creation of bids. From a schedule you may determine how long it would take to:

- Reach the loading port (origin port),
- Load the cargo,
- Travel from the origin port to the destination port,
- Unload the cargo,
- as well as potential idling times.

Once you know that you can calculate the fuel consumption and use this as an estimate of the costs. A simple way of bidding can be achieved by modifying `propose_schedules`. If you build up a dictionary that stores for each trade the calculated cost which is returned with the `ScheduleProposal`, the default implementation of inform will automatically bid these costs.

The following example shows a simple scheduling strategy which blanks to calculate costs.

```python
    def propose_schedules(self, trades):
        schedules = {}
        scheduled_trades = []
        costs = {}
        i = 0
        while i < len(trades):
            current_trade = trades[i]
            is_assigned = False
            j = 0
            while j < len(self._fleet) and not is_assigned:
                current_vessel = self._fleet[j]
                current_vessel_schedule = schedules.get(current_vessel,
                ↪   current_vessel.schedule)
                new_schedule = current_vessel_schedule.copy()
                new_schedule.add_transportation(current_trade)
                if new_schedule.verify_schedule():
                    # TODO calculate costs
                    costs[current_trade] = 0  # TODO Add cost
                    schedules[current_vessel] = new_schedule
                    scheduled_trades.append(current_trade)
                    is_assigned = True
                j += 1
            i += 1
        return ScheduleProposal(schedules, scheduled_trades, costs)
```

To calculate costs, MABLE provides further useful functions.

- Do determine times and fuel consumptions there are a lot of convenience functions in the vessel classes, as mentioned in Section 4.2.
- `def get_network_distance` Your company class has a property `def headquarters` which allows you to access its `class CompanyHeadquarters`. This class provides the `get_network_distance` to calculate the distance between two ports.

`def inform` will simply take the list of scheduled trades and the costs and will bid for those trades using the costs as the bid value. Its implementation looks like this.

```
def inform(self, trades, *args, **kwargs):
    proposed_scheduling =
    ↪ self.propose_schedules(trades)
    scheduled_trades =
    ↪ proposed_scheduling.scheduled_trades
    trades_and_costs = [
        (x, proposed_scheduling.costs[x]) if x
        ↪ in proposed_scheduling.costs
        else 0
        for x in scheduled_trades]
    bids = [Bid(amount=cost, trade=one_trade)
    ↪ for one_trade, cost in
    ↪ trades_and_costs]
    return bids
```

The `def receive` function will simply apply `def propose_schedules` to the won trades and uses these schedules. For your own agent you might want to have a different procedure to schedule these cargoes.

10 Running Tournaments

If you need to adjust how many auctions run and how many cargoes are auctioned off to test different settings with different numbers of vessels and companies, have a look at `def get_specification_builder` and its parameters.

11 Future Auctions and Cargoes

Before a cargo auction, your company will be informed of the subsequent auction's cargoes (see also Section 5). As introduced in Section 6, this will happen via a call to the `pre_inform`. The function has no default implementation in `class TradingCompany` meaning that your company will do nothing if you do not override the function.

Remember that there is no certainty that you will get the trade in this round or that even if you win the contract you will also get the following one.

12 Competitors' Fleets

The previous sections should give you an idea of trades, including future ones, and your companies vessels. In the same way that so far your company has made decisions based on the vessels and trades your competitors will do the same. Often the cheapest vessel to use may be the closest vessel to the origin port because there is less additional cost of moving the vessel there first.

You can use the `def get_companies` function of your headquarters to get a list of all companies. Note that this list also includes your own company.

13 A Playground Sandbox

To make development of your competition agent easier you can use the following provided files which can be downloaded from the GitHub repository.

```
main_competition_playground.py
groupn.py
```

The entry point scrip `main_competition_playground.py` runs your python code and `groupn.py` can be used to add your agent code.

- Within `groupn.py` there is a class called `Companyn` which is where you should **add your code**.

If you look into `main_competition_playground.py` you should see in the function `def build_specification` that your company and two further companies are added to the simulation. The company `class MyArchEnemy` and `class TheScheduler` are very simple agents.

There are a number of options to adjust so that you can test different scenarios.

- `number_of_month` allows you to specify how long the simulation will run. Every month there will be one auction.
- `trades_per_auction` allows you to specify the number of trades that are being auctioned in one of the monthly auctions.
- `def mixed_fleet` is used to create fleets.
  - It generates fleets with a specified number of three types of vessels called *Suezmax*, *Aframax* and *VLCC*.

   - The function expects three arguments: `num_suezmax`, `num_aframax` and `num_vlcc` to specify the number of the respective types of vessels in the fleet, i.e.

     ```
     mixed_fleet(num_suezmax=2, num_aframax=3, num_vlcc=0)
     ```

     will create a fleet with 5 vessels of which 2 will be *Suezmax*, 3 will be *Aframax* and none will be *VLCC*.

- Both `MyArchEnemy` and `TheScheduler` have a constructor argument `profit_factor` which is used to modify the cost to bid calculation of the agent, i.e. the agent will bid `cost * profit_factor`.

- Also take note of the `global_agent_timeout` parameter of the environment creation function `def generate_simulation`. It is set to 60 by default, which means your company has 60 seconds to complete `pre_inform`, `inform`, `receive`. If your company does not finish within that time its operation will be cancelled and your company will not bid or schedule the trades and thus looses its chance to transport cargoes or will have to pay a penalty.