COMP6203 Intelligent Agents
Lab Instructions

Jan Buermann
j.buermann@soton.ac.uk

2024

Contents

Lab #2

2.1 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.)

2.1.1 Transportation Opportunities (TimeWindowTrade)

Cargo transportation opportunities the companies can bid for are represented by `class TimeWindowTrade` (see Figure 1). 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.

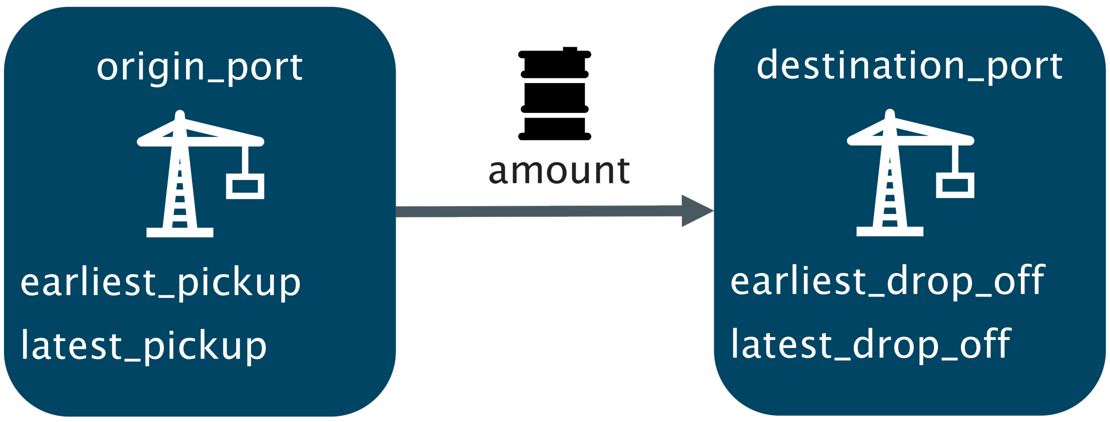

Figure 1: Illustration of a time window trade.

- `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.

2.1.2 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 2), 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.

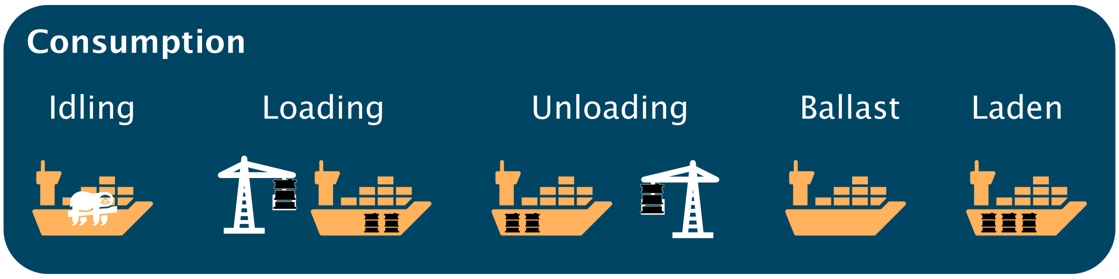

Figure 2: Vessel statuses.

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.

2.2 The Market

Periodically an auction will be run at which several transportation opportunities (see Section 2.1.1) 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.

2.3 Running a Company

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

```
class MyCompany(TradingCompany):

    def pre_inform(trades, time):
        pass

    def inform(trades, *args, **kwargs):
        pass

    def receive(contracts, auction_ledger, *args, **kwargs)
        pass

     def propose_schedules(self, trades):
        pass
```

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

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

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. This will be discussed in Lab #3.)

2.4 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.

2.5 Task 1: Creating a Schedule

Last week we created a subclass of `class TradingCompany`. This week we will extend it by writing how our company schedules the transportation of trades and later how to bid for trades. Starting with 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. In your first task you will override this function `def propose_schedules`.

Firstly, the function 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. So add the following import to your code.

```
from mable.transport_operation import ScheduleProposal
```

2.5.1 Task 1.1: A Very Simple Scheduling

Let's add a simple implantation of `def propose_schedules`. The idea is to just iterate over the list of trades and the list of vessels and check if for every trade-vessel combination the vessel can transport the cargo after finishing the current cargoes. The following functions, properties and instance variables should be sufficient to achieve this.

- `def add_transportation` Add a trade to a schedule (`class Schedule`). If the function is called specifying only the trade, the function will append the trade to the end of the schedule.
- `def fleet` Using the `fleet` property you can access your company's fleet of vessels (`class TradingCompany`).
- `def verify_schedule` Once you have constructed a schedule, you can use the function `def verify_schedule` (`class 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 copy function of `class Schedule` provides you with a deep copy of the schedule to test changes to a schedule without messing up the original schedule.

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.
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.

2.5.2 Task 1.2: Slightly Better Scheduling

Sometimes two cargo pick up locations may be close to each other and it would make more sense to pick up both cargoes before dropping them off (assuming the cargo hold is big enough). The `class Schedule` has a function `get_insertion_points` which allows you to get all valid locations that can be used for `add_transportation`. 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 2.4.

2.6 Task 2: Creating a Bid

As mentioned in Section 2.1.2, the vessel classes have a convenience methods for loading rates, loading times and fuel costs. The only additional function to allow you to calculate all times and costs is the `def get_network_distance`. This function allows you to calculate the distance between two locations, e.g. the distance between two ports. You can access the `class CompanyHeadquarters` which provides this function via the `def headquarters` property in `class TradingCompany`. i.e. your company's class.

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.

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