Operating the Framework

This is a comprehensive guide on how to operate the developed ATFM simulation framework effectively. It is designed to help users understand the practical steps involved in deploying and utilizing the framework, from initial setup to executing simulation steps. The following subsections cover the installation process, the configuration of the framework, and a demonstration with an example.

Installation and Setup

This section outlines the essential steps required to install and prepare the ATFM simulation framework for operation. The installation process focuses on ensuring that the correct version of Python is installed and that all necessary dependencies are properly set up. By following these guidelines, users can quickly get the framework ready for use. The section references the official Python documentation and other relevant resources for detailed instructions on Python installation and managing dependencies.

The simulation framework was developed using Python version 3.10.9, which is the recommended version to run the simulations.

The following Python packages are also required for the framework to work properly:

- Random: A module for generating random numbers and making random selections in Python. (https://docs.python.org/3/library/random.html)

- Copy: A module providing support for shallow and deep copying operations in Python. (https://docs.python.org/3/library/copy.html)

- Logging: A module that implements an event-logging system for Python applications. (https://docs.python.org/3/library/logging.html)

- Json: A module that encodes and decodes JSON files. (https://docs.python.org/3/library/json.html)

- NumPy: A library for numerical computations in Python. The version used in the simulation framework is 1.24.2. (https://numpy.org/doc/)

- Pandas: A library for data manipulation in Python. The version used in the simulation framework is 2.2.1. (https://pandas.pydata.org/)

- Docplex.mp.model: A module from the `docplex` library, an optimization modeling library for Python provided by IBM Decision Optimization. The recommended version is 2.25.236. (https://ibmdecisionoptimization.github.io/docplex-doc/mp/docplex.mp.model.html)

The free version of this package is limited, and it is recommended to obtain an academic license to remove size limitations when running optimization simulations. Academic license details: (https://community.ibm.com/community/user/ai-datascience/blogs/xavier-nodet1/2020/07/09/cplex-free-for-students)

- Mesa: An agent-based modeling (ABM) framework developed in Python. The version used for developing the framework is 2.1. It can be installed using: pip install "git+https://github.com/projectmesa/mesa.git@v2.1.0"

Documentation: (https://mesa.readthedocs.io/en/stable/)

Simulation Framework’s File Structure

This section introduces the file structure of the air traffic flow management simulation framework. Understanding the organization of files and modules within the framework is crucial for effective navigation and usage.

Simulation Framework File Structure

|- ATFMSimulation.py

| Contains classes and functions to rule the simulation model.

|- MesaScheduler.py

| Contains classes and functions to instantiate the scheduler Mesa requires.

|- AirlineAgent.py

| Contains classes and functions of an airline agent.

|- PlatformAgent.py

| Contains classes and functions of a platform agent.

|- NetworkManagerAgent.py

| Contains classes and functions of a network manager agent.

|- Flight.py

| Contains classes for scheduled flight and actual flight.

|- TimeAssignmentStrategy.py

| Contains classes for assigning scheduled flights to times.

|- WindowStrategy.py

| Contains classes to determine the span of the window of a regulation.

|- RegulationStrategy.py

| Contains classes to determine the capacity reduction of a regulation.

|- PriorityAssignmentStrategy.py

| Contains classes to determine which flights will be prioritized.

|- DesiredTimeStrategy.py

| Contains classes to determine the desired time of a priority flight.

|- AssignMarginsStrategy.py

| Contains classes to determine the time not before and not after margins.

|- InterveneEquityStrategy.py

| Contains classes and functions to execute the equity intervention.

|- OptimizationStrategy.py

| Contains classes and functions to execute the optimization algorithm.

|- EquityMonitor.py

| Contains classes and functions to monitor the equity metrics.

|- credits\_clearing.py

| Contains classes and functions to handle the equity mechanism.

|- LoggingConfig.py

| Contains classes to configure logging.

Configuration

This section provides an overview of the configuration steps necessary to customize the air traffic flow management simulation framework according to specific operational needs. Proper configuration ensures that the framework operates efficiently and aligns with the intended simulation scenarios. The configuration process involves selecting and setting simulation parameters and agent behaviors.

To configure the ATFM simulation framework, we import the modules for the strategies and define the parameters. Then, we create the inputs required for the ATFMSimulation model. The following example illustrates how to initiate the configuration of the ATFM simulation framework.

A Python file named `setup.py` is created. In this file, we define the framework's strategies and parameters that control how the simulation operates.

Importing Modules for Setup

```python

Import modules for setup

Time assignment strategy

import TimeAssignmentStrategy

Network manager agent strategies

import WindowStrategy

import RegulationStrategy

Airline agent strategies

import PriorityAssignmentStrategy

import DesiredTimeStrategy

import AssignMarginsStrategy

Platform agent strategies

import OptimizationStrategy

import EquityMonitorStrategy

import InterveneEquityStrategy

```

Setting Simulation Parameters

```python

Set parameters

steps = 100 Number of simulation iterations

n\_times\_ = 100 Number of available time slots

starting\_seed\_ = 0

Weight map parameters

base\_value\_weight\_map\_ = 100

percentage\_reduction\_weight\_map\_ = 0.1

```

Initializing Strategies for Simulation

```python

Strategies used

Regular schedule time assignment strategy

time\_assignment\_strategy = TimeAssignmentStrategy.RandomTimeAssignmentStrategy(time\_assignment\_seed=0)

Network manager agent strategies

window\_strategy\_ = WindowStrategy.RandomWindowStrategy()

regulation\_strategy\_ = RegulationStrategy.ReducedCapacityStrategy(capacity=2)

Airline's flight prioritization strategies

priority\_assignment\_strategy = PriorityAssignmentStrategy.PriorityRandomAssignmentStrategy()

desired\_time\_strategy = DesiredTimeStrategy.DesiredTimeRandomAssignmentStrategy()

assign\_margins\_strategy = AssignMarginsStrategy.MarginsRandomAssignmentStrategy()

Platform agent strategies

optimization\_strategy\_ = OptimizationStrategy.LinearAssignmentStrategy()

equity\_monitor\_strategy\_ = EquityMonitorStrategy.CreditsClearingStrategy(initial\_credits=100)

intervene\_equity\_strategy\_ = InterveneEquityStrategy.CreditBasedInterventionStrategy(equity\_threshold=0)

```

Running the Simulation

A `main.py` file is created as the entry point for running the air traffic flow management simulation. It imports necessary modules, initializes the simulation with the configured parameters, sets up logging, and executes the simulation for a specified number of steps.

Initializing and Running the Model

```python

import ATFMSimulation

import Setup

import logging

import LoggingConfiguration

Initialize the simulation model with parameters from Setup

model = ATFMSimulation.ATFMSimulationModel(

n\_times=Setup.n\_times\_,

seed=Setup.starting\_seed\_,

time\_assignment\_strategy=Setup.time\_assignment\_strategy,

network\_manager\_agent=Setup.network\_manager\_agent\_,

list\_airline\_agents=Setup.list\_airline\_agents\_,

platform\_agent=Setup.platform\_agent\_

)

LoggingConfiguration.configure\_logging()

for i in range(Setup.steps):

model.step()

```

Output Data

The output data of the simulation framework is stored in a machine-readable format as a JSON file named `Output\_data.json`. The log file `Simulation\_log.log` is also generated, containing intermediate results for traceability.