**Academic Year**

2024 - 2025

**Software engineering for real-time electricity market bidding**

Sameer Baruwal

Bachelor’s thesis to obtain the degree of Bachelor of Science in

**Electronics and ICT Engineering Technology**

Promotor

Benoit De Vrieze

Copromotor

Prof. dr. Peter Hellinckx

Inhoud

[1 Abstract 3](#_Toc198044412)

[2 Introduction 3](#_Toc198044413)

[3 Background 4](#_Toc198044414)

[3.1 The electricity market 4](#_Toc198044415)

[3.2 The ASSUME Framework 4](#_Toc198044416)

[3.3 Profiling & Optimization 5](#_Toc198044417)

[4 Methodology 6](#_Toc198044418)

[4.1 Initial simulation 6](#_Toc198044419)

[4.2 Profiling 6](#_Toc198044420)

[4.2.1 cProfile & yappi 7](#_Toc198044421)

[4.2.2 Tracemalloc 7](#_Toc198044422)

[5 Results 7](#_Toc198044423)

[5.1 Simulation results 7](#_Toc198044424)

[5.2 Profiling 8](#_Toc198044425)

[6 Conclusion 9](#_Toc198044426)

[7 Acknowledgements 9](#_Toc198044427)

[8 References 9](#_Toc198044428)

# Abstract

This study’s main goal is to simulate ten thousand residential agents on a model of the Belgian electricity market.

# Introduction

This research delves into the ASSUME framework to simulate an energy market with 10 000 residential agents. The framework is an easy-to-use market simulation toolbox with integrated reinforcement learning methods, though these methods are not used.It was chosen so that we did not need to develop a market from scratch, which can be quite complex. The main goal of this project is to simulate 10 000 residential agents on the Belgian electricity market in five minutes. To do this, we need to optimize the ASSUME framework efficiently and decrease its simulation time. Therefore, we need to find the key performance bottlenecks by means of profiling. The research questions are:

* How can we optimize the ASSUME framework efficiently to simulate 10 000 agents?
* What are the key performance bottlenecks?

This research displays a futuristic scenario where residential consumers, instead of buying electricity from a company like Engie, can buy it directly off the market. This can be done using a sort of aggregator… **which is?** Like an AI machine that buys electricity at a low cost. This is also the purpose of this research to simulate what it is like when residents buy electricity directly from the market. This helps us understand the scenario before a real-life implementation can be realized. It links with modelling for sustainability… **why is it sustainable?**

The next section will discuss the literature background to this research project. It will discuss the working mechanism of the electricity market, how the ASSUME framework models that market and finally what optimization is and how it is (not) implemented into the framework. The section that follows discusses the ways that the project was set up. This includes the initial set up of a simulation environment that models the Belgian market, the analysis of the bottlenecks with various profiling techniques and the optimization methods that were considered to overcome the bottlenecks. The results section will be graphical section where the results of the simulation, profiling, optimization and a comparison between the before and after will be discussed. Finally, we conclude this paper by summarizing the project, answering the research question, discussing the limitations and potential improvements.

# Background

## The electricity market

The electricity market is very different from the publicly known markets such as the stock market. The main difference lies in the nature of the commodity being traded and how it is used. The electricity market trades electricity, which is an instantaneous and non-storable commodity while the stock market trades ownership of companies in the form of shares.

The main electricity market in Belgium is the Epexspot. The members of the market submit orders for buying and/or selling power, which are registered in an orderbook. These orders reflect the demand and supply of the market at a specific moment in time. Based on the orderbook, a market clearing price is calculated which will be explained in a later paragraph. Once the trade is complete, the transaction is cleared and settled. Clearing ensures the proper fulfillment of each contract registered on the market. The clearing entity, which in the case of the Epexspot is the ECC, steps in and becomes the contractual partner for both the buyer and the seller to ensure the fulfillment of each trade and to mitigate counterparty risk. By matching supply and demand, the market ensures transparent and reliable prices and the market operators such as the ECC (via TSOs) make sure the electricity is delivered and paid for.

The Epexspot operates in two markets, the Day-Ahead and the Intraday market, that fulfill their own purpose. The day ahead market operates through a blind auction that takes place once a day, all year around where all the hours of the following day are traded. The orders are logged in by the members before the orderbook closes. The market established a demand curve based on the buy-orders and a supply curve based on the sell order, both for each hour of the following day. The market clearing price (MCP), which reflects the demand and supply, lies at the intersection of both curves. **Maybe a graph of the MCP?** The MCP, that is determined for each delivery period, applies to all buyers and sellers. All buyers who submitted volumes at a price higher than the MCP are executed for these volumes and pay the MCP, and all sellers who submitted volumes priced lower than the MCP are executed for these volumes and receive the MCP.

The Intraday market offers the possibility to trade even more in the short term. On the Intraday continuous market, participants trade 24 hours a day, with delivery on the same day. As soon as a buy- and sell-orders match, the trade is executed. Electricity can be traded for up to 5 minutes before delivery and through hourly, half-hourly or quarter-hourly contracts. As this allows for a high level of flexibility, members use the Intraday market to make last-minute adjustments and to balance their positions closer to real time.

The price formation process on the Epexspot Day-Ahead market follows the merit order principle. This principle guarantees the lowest possible prices to satisfy demand on the power market, as the generation with the lowest costs (or the willingness to sell at the lowest price) is dispatched first. The most expensive unit that must be activated to meet the demand sets the price, the market clearing price. The least expensive units are usually renewables like wind and solar while the most expensive are gas and coal plants.

## The ASSUME Framework

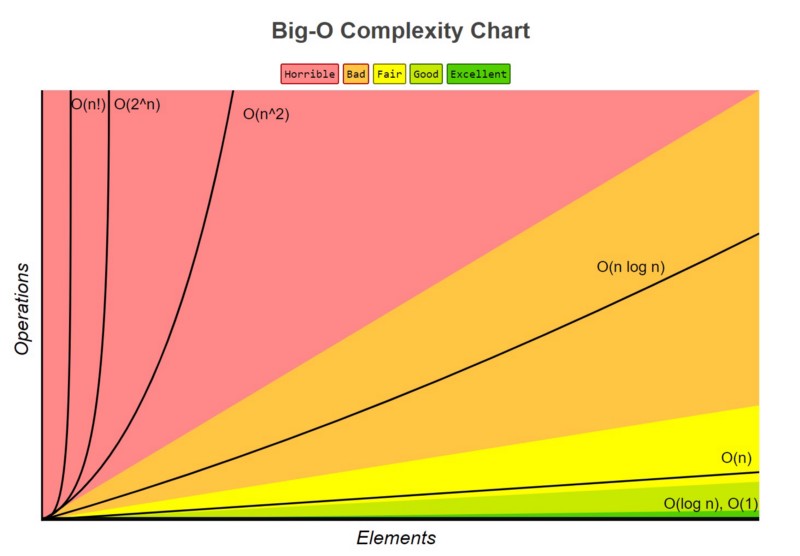
The ASSUME framework works with a World entity. Here, you can add multiple markets, producers and consumers to closely simulate a real-world example. In our case however, we work with a single day ahead market, five producers and ten thousand consumers to approximate the Belgian electricity market.

A producer or consumer is defined by the Unit class with the following properties: id, unit\_type, unit\_operator\_id, unit\_params and forecaster. The id is used to avoid duplicates and the unit\_type is used to differentiate whether it is a producer, ‘power\_plant’, or a consumer, ‘demand’. There are seven unit\_types defined, and the simulation will throw an error if you choose one that is not defined. The unit\_operator\_id acts as a RoleAgent from the mango framework where the units defined under this operator act as roles. Without going into much detail, the operator (RoleAgent) is activated and in turns activates all the units (roles). The unit\_params defines the market that the units belong to, the bidding strategy, the minimum or maximum power and the price at which they bid of offer. **Maybe a little better and more concise**

## Profiling & Optimization

Optimization is the process of modifying a software system to make some aspects of it work more efficiently or use fewer resources. In general, a program can be optimized so that it executes more quickly, uses less memory or even draws less power. Optimization often comes with trade-offs, where enhancing one metric may come at the expense of another. A very common one is the space-time trade-off where you make the trade between program runtime and memory usage. The goal is to find a balance with the resources that you have and the objective you are trying to reach. To find what exactly needs to be optimized, we must find the performance bottlenecks, the parts of the code that consume the most time and resources, by doing a performance analysis by ways of profiling.

Profiling is an analysis technique used to measure and analyse a program’s performance while it is running. There are two types of profilers: statistical and deterministic. Statistical profilers periodically check what the program is doing by sampling the call stack, they produce low overhead and will not slow down the program much, but they are not very precise in terms of timing and call count. Deterministic profilers on the other hand are very precise because they log every single function call and return, but this also means that they produce a lot of overhead.

An understanding of the Big O notation is essential as it will be used throughout this paper when analysing the computational performance of the algorithms. The big O notation is a way of describing how fast an algorithm grows as the size of the input grows. It is not an indication of how long something takes, but of how the performance will scale, especially in the worst-case scenario. For example, if something is O(n), that means if you double the data, the time it takes roughly doubles too. We say that the performance scales linearly. The figure below shows different notations and how they scale. We try to stay away from the red and in the worst-case scenario, end p in the orange.

*Figure 1: Big-O complexity chart, source: freecodecamp*

# Methodology

The work is split into three sequential steps:

1. Initial simulation
2. Profiling
3. Optimization

These three steps will be explained in depth in the following paragraphs

## Initial simulation

We initialize a simulation with five powerplants: nuclear, natural gas, wind, biofuel and solar. They represent about 95 % of the power generation in Belgium. We initialize their emission factor, maximum power output, efficiency and cost. The values were gathered from various sources and calculations and represent not the exact but an average value. Then, we initialize the consumers that consists of residentials units that have a demand value corresponding to the consumption data of an anonymous digital meter gathered from the Fluvius database. Finally, an agent0 was created to represent the demand of the rest of the consumers with data from the Elia open database. Fluvius is the DSO of Belgium and all the data that is used is from the year 2022 because we only had the meter data from that year. The consumer’s demand is the exact amount of power consumed according to the historical data; forecasts do not play a role in the simulation. The supply is defined by the maximum power output and is fixed. If the supply cannot meet the demand, there is a shortage that cannot be accounted for.

The ASSUME framework provides the possibility of using csv files to defines units, the market is defined using a yaml file. We store all these files in a certain folder and give the path name and the world entity as parameters into the simulation function of the framework. The output is saved in an SQLite database at the path defines by the user, given as a parameter when initializing the World entity. The database consists of 7 tables: demand\_meta, kpis, market\_dispatch, market\_meta, market\_orders, power\_plant\_meta and unit\_dispatch. The meta tables just list all the units and markets with their properties. The market\_orders table lists all the orders, bids and offers, of all the units for each hour. The market\_dispatch shows a table with the dispatch of power. It shows for each unit how much it had produces and consumed with positive indicating production. **Unit\_dispatch?** The KPI table shows the average price per MWh, the total volume produced/consumed for that simulation period and the total cost, which is the multiplication of the previous two values.

## Profiling

The profiling was done using the following time and memory profilers.

* cProfile & yappi (time)
* tracemalloc & pyinstrument (memory)

Each profiler will be explained with reasoning on why it was or was not chosen to do the final profiling.

### cProfile & yappi

They are both deterministic profilers. cProfile is a built-in python module while yappi had to be installed first. The main difference is that yappi support multithreaded, asyncio and gevent profiling. It can also track either WALL time and CPU time while cProfile only tracks CPU time. The ASSUME framework is built on top of the Mango framework which in turn in built on top of asyncio. Asyncio is module that allows concurrent programming, though it is not really concurrent. Using the await/async method, a user can make a function wait that would otherwise be blocking, and allow other function to run during the waiting period. This **note** shows the main issue that cProfile faces when profiling asyncio code. **Should it be explain (quite extensive) or is a note enough?** Hence, we chose yappi as out time profiler. The module pstats was used to format the profiling statistics into reports and snakviz was used to visualize them.

### Tracemalloc

There were not many in depth memory profilers to choose from. An alternative the memory\_profiler but this is a line profiler that only showed how much memory a single line consumed without going to its subcalls. Tracemalloc gives us a plethora of option to see how memory is allocated.

# Results

This section presents the results of the simulation framework in three stages. First, we describe the outcome of the initial simulation and how it scales with more agents. Second, we analyse the results of the yappi profiler and assess the bottlenecks in the framework. Finally, we outline the possible optimization that could be used to improve runtime and scalability using the Big O notation.

## Simulation results

The runtime of a simulation described in the methodology using nine residential agents and one agent0 is about 200 s. The simulation was run from two up to a hundred agents to see how the simulation scales and it gave us the following result.

*Figure 2: Graph representation of the simulation runtime in function of the number of agents*

We can conclude from the graph that the simulation scales linearly with a base runtime of about 143 seconds. This means that the system scales in a predictable and stable way and is not experiencing slowdowns but that has yet to be tested by running the simulation with more agents. The equation is for the trendline is:

With x the amount of agents and y the simulation runtime.

That means that for every added agent, the runtime increases with 5.61 seconds. If this trendline continues, we would get the following time for the simulation of ten thousand agents.

This is about 187 times slower than our goal of a simulation runtime of 5 minutes.

The same thing was done using a memory profiler and the following results were obtained. **Graph of memory scalability maybe?**

## Profiling

What is common is both time and memory profilers is that they produce significant amount of overhead. The simulation runtime doubles when using yappi and almost quadruples when using tracemalloc.

# Conclusion

# Acknowledgements

# References

<https://www.freecodecamp.org/news/all-you-need-to-know-about-big-o-notation-to-crack-your-next-coding-interview-9d575e7eec4/>

<https://kwhprice.eu/en/belgium>

<https://www.creg.be/en/highlights-creg-2022-annual-report>

<https://lowcarbonpower.org/region/Belgium>

<https://en.wikipedia.org/wiki/Electricity_sector_in_Belgium>

Harder, N., Qussous, R., & Weidlich, A. (2023). Fit for purpose: Modeling wholesale electricity markets realistically with multi-agent deep reinforcement learning. Energy and AI, 14, 100295. doi: 10.1016/j.egyai.2023.100295

Basics of the Power Market | EPEX SPOT. (2025, May 02). Retrieved from <https://www.epexspot.com/en/basicspowermarket>

Total load by all grid users: <https://opendata.elia.be/explore/dataset/ods001/table/>

(for the "agent 0")

Day ahead forecast of all generation per type: <https://opendata.elia.be/explore/dataset/ods034/information/>

Actual generation per type: <https://opendata.elia.be/explore/dataset/ods033/information/>

Smart meter profiles: <https://opendata.fluvius.be/explore/dataset/1_50-verbruiksprofielen-dm-elek-kwartierwaarden-voor-een-volledig-jaar/information/>

<https://www.epexspot.com/en/market-results>

<https://www.epexspot.com/en/basicspowermarket>

<https://www.next-kraftwerke.com/knowledge/day-ahead-trading-electricity>

<https://www.next-kraftwerke.com/knowledge/intraday-trading>

<https://assume-project.de/#research>

<https://www.sciencedirect.com/science/article/pii/S2666546823000678?via%3Dihub>

[https://docs.python.org/3/library/timeit.html](https://eur01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fdocs.python.org%2F3%2Flibrary%2Ftimeit.html&data=05%7C02%7CSameer.Baruwal%40student.uantwerpen.be%7Cf03beb0371cb425ca27f08dd65541ebc%7C792e08fb2d544a8eaf72202548136ef6%7C0%7C0%7C638778134131775603%7CUnknown%7CTWFpbGZsb3d8eyJFbXB0eU1hcGkiOnRydWUsIlYiOiIwLjAuMDAwMCIsIlAiOiJXaW4zMiIsIkFOIjoiTWFpbCIsIldUIjoyfQ%3D%3D%7C0%7C%7C%7C&sdata=C57XPsl9osXPqRih2tfTR4uFLSSvXZWFGR0%2BweVvcLs%3D&reserved=0)

<https://docs.python.org/3/library/profile.html>

<https://docs.python.org/3/library/tracemalloc.html>

<https://en.wikipedia.org/wiki/Program_optimization>

<https://www.ranorex.com/blog/what-is-code-profiling-and-how-to-choose-the-right-tool/>

<https://www.machinelearningplus.com/python/cprofile-how-to-profile-your-python-code/>

<https://realpython.com/async-io-python/>