Power Laws: Optimizing Demand-side Strategies

Second price solution

guillermobarbadillo@gmail.com

Introduction

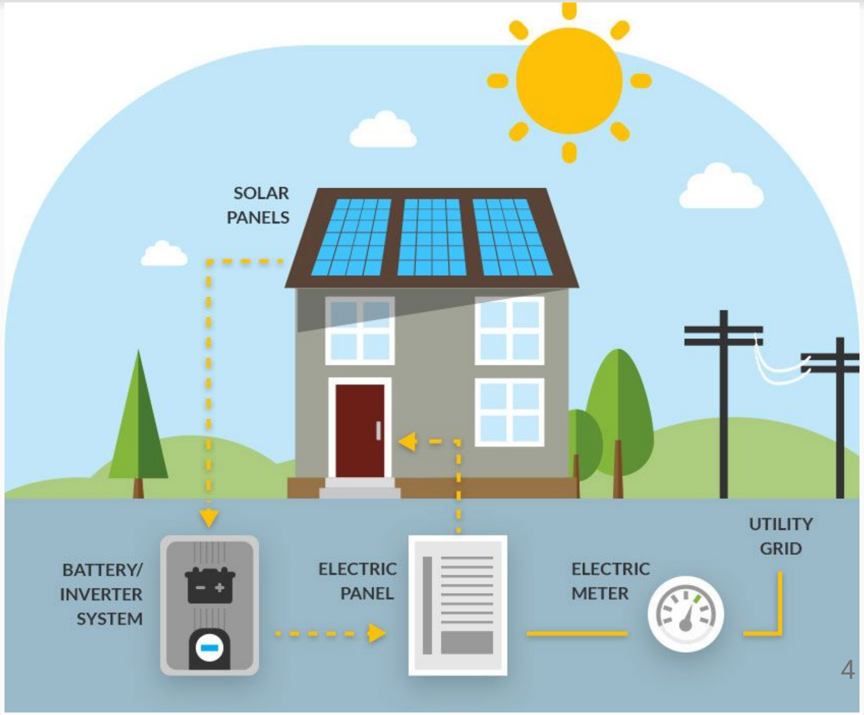
Power Laws: Optimizing Demand-side Strategies

This is one of three challenges in the “Power Laws” series being run simultaneously by **Schneider Electric**

They are hosted on **drivendata.org**

Power Laws: Optimizing Demand-side Strategies

Your goal in this competition is to build an algorithm that controls a battery charging system and spends the least amount of money over a simulation period.



Resources

- Train and public test datasets for 11 different sites (csv files). About 2.8GB of data.
- A simulation that runs on docker and computes the score for our algorithm

The private test dataset is not known and it has not been published.

Submission

At the end of the challenge we have to provide a python script called battery_controller.py that implements a BatteryController class that will be used by the simulation.

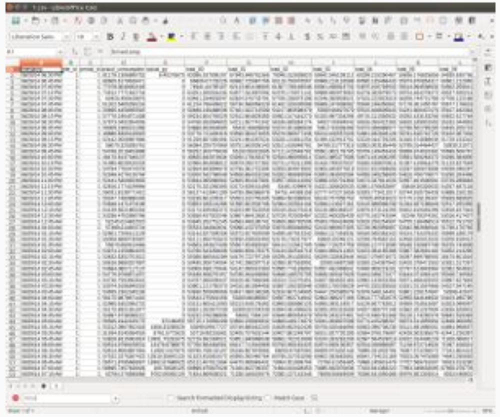
A folder called assets may be used with models or data that the controller uses.

The simulation should be able to run 240 simulations (2400 days) in less than 30 minutes. A dummy controller takes 10 minutes, so that leaves a small margin.

Data exploration

Data

- Data every 15 minutes
- 11 different sites
- The duration of the train data is about 400 days and 100 days for the public test set
- The data is divided into smaller periods of time. 10 days for the test set
- 2.2GB for train and 0.5GB of data for the public test
- Csv files



Features

- Energy demand and photovoltaic production at each timestep
- Forecast for the following 96 timesteps (1 day)
  - Price of selling and buying energy
  - Energy demand
  - Photovoltaic production

And we are also given the properties of the battery: capacity, charge and discharge power, charge and discharge efficiency

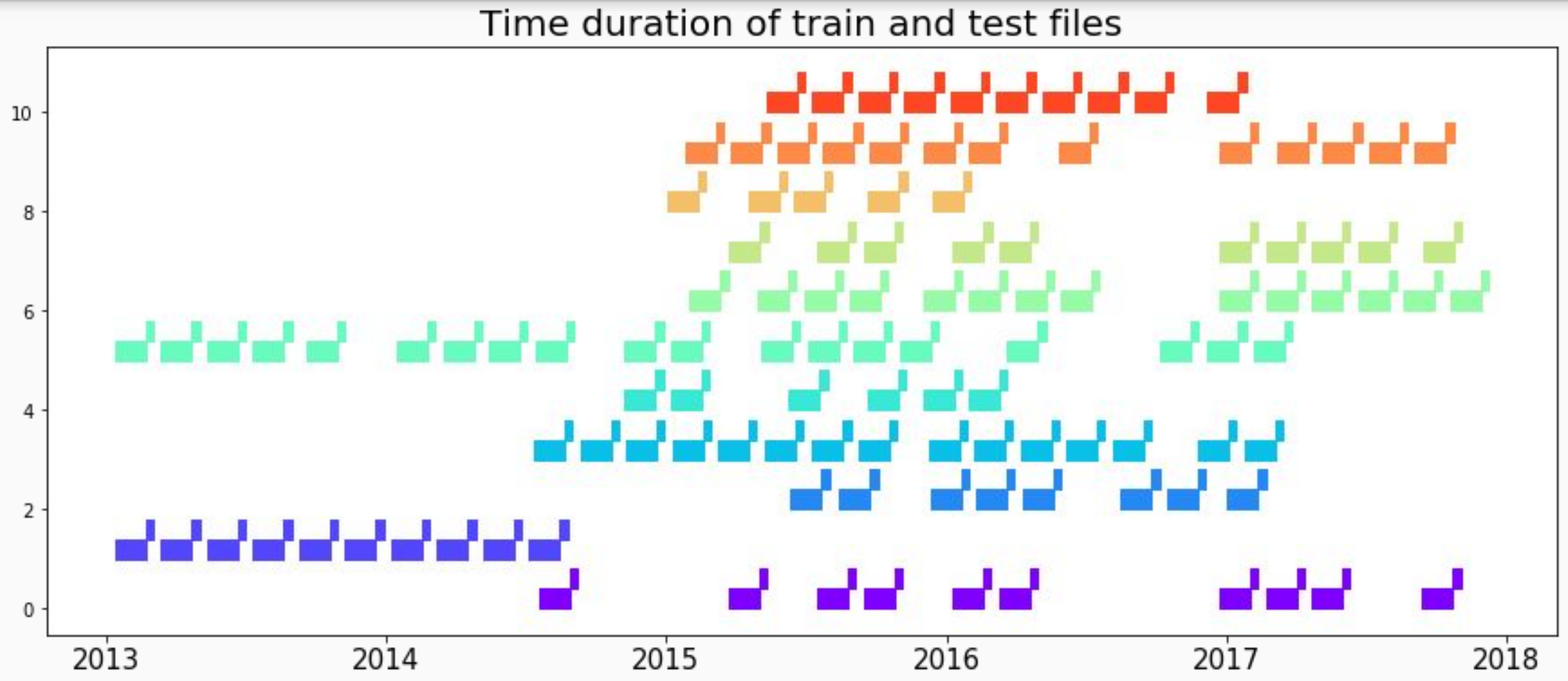
Data exploration

Understanding the data is crucial for designing a good model/algorithm. As a consequence of the No free lunch theorem we have to adapt our solution to the data in order to get the maximum performance.

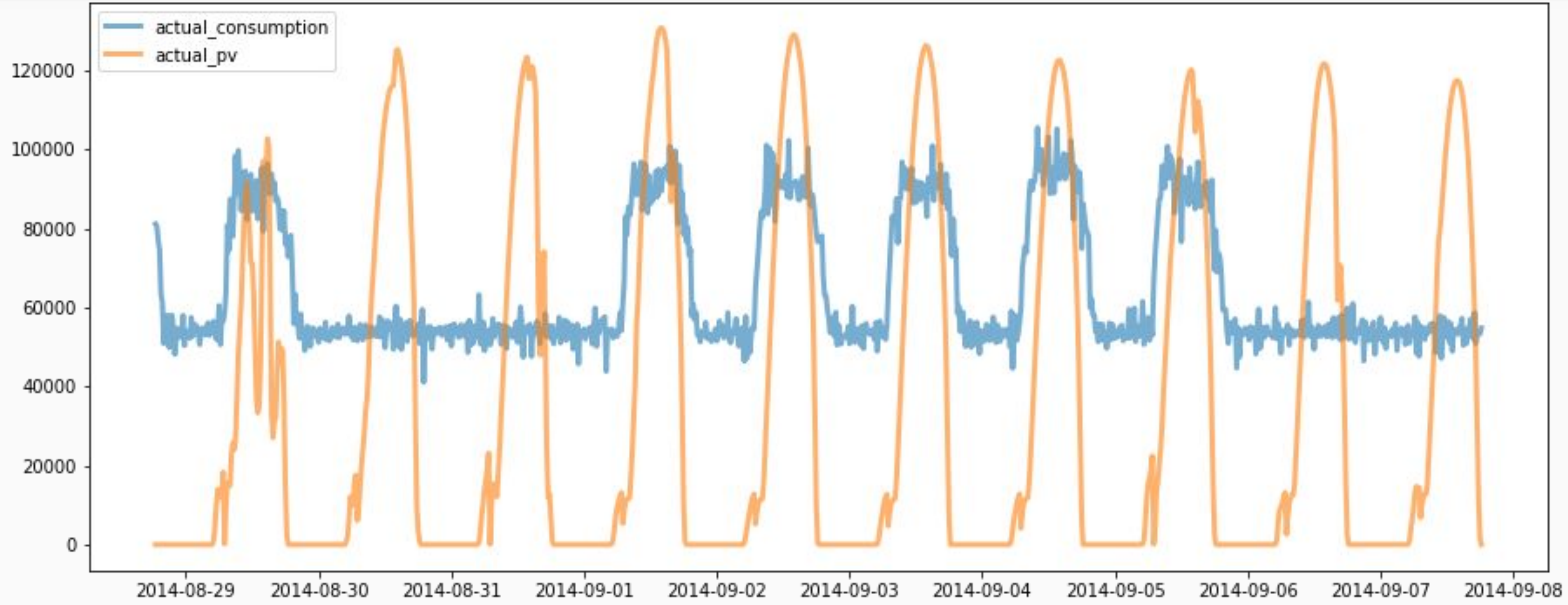
Jupyter notebooks are are great tool for loading, analyzing and visualizing data.

On the following slides I will show some visualizations that help to get insights into the data

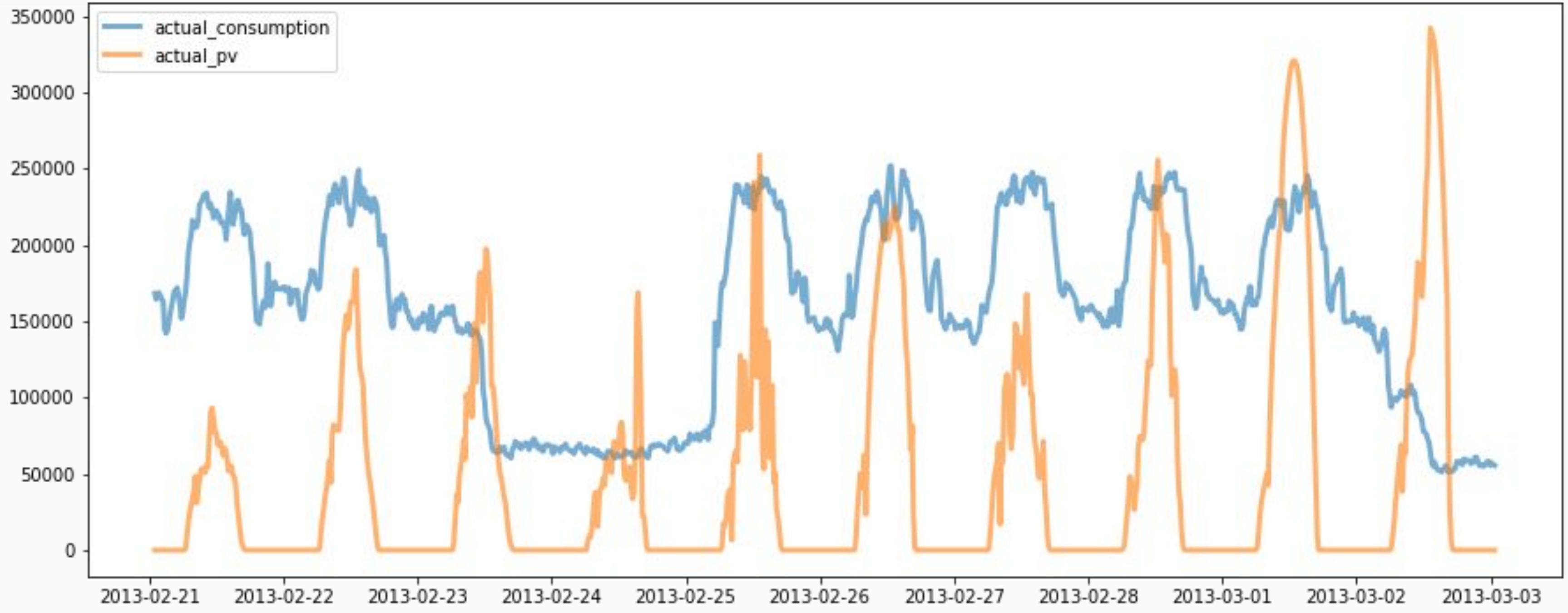
Visualization of the time periods for the different sites



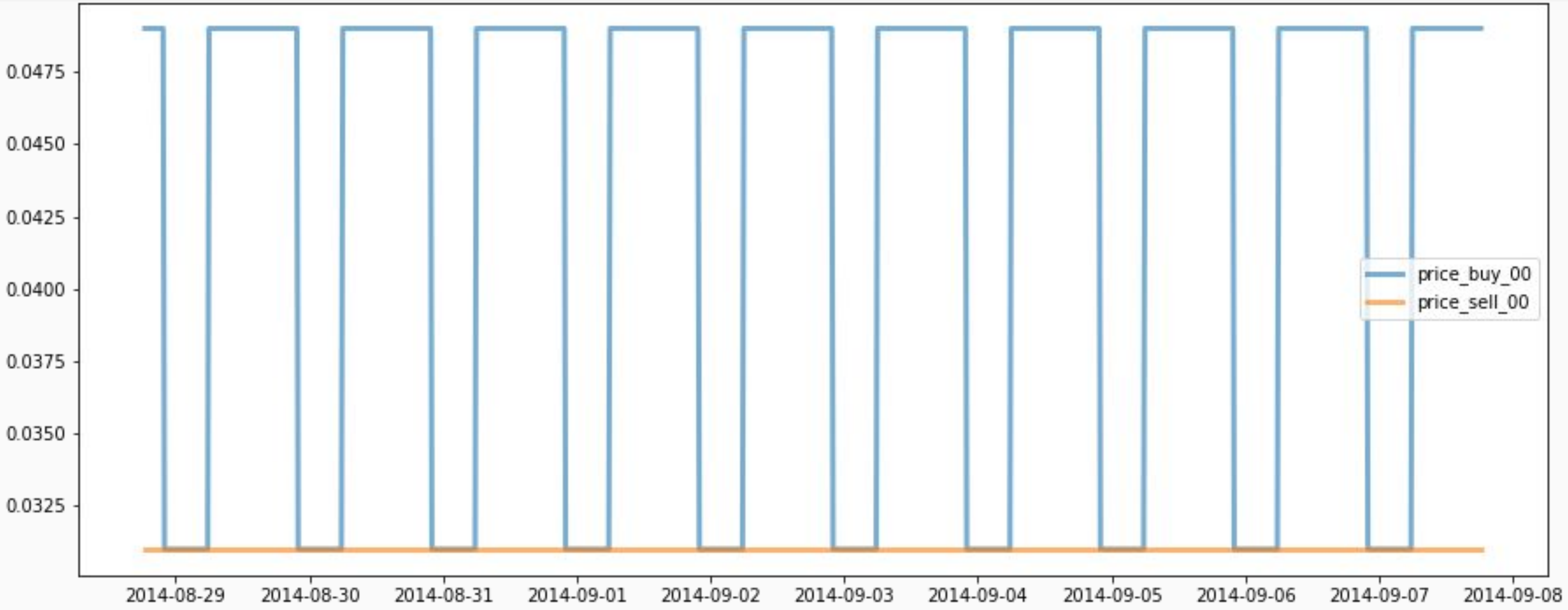
Consumption and photovoltaic production



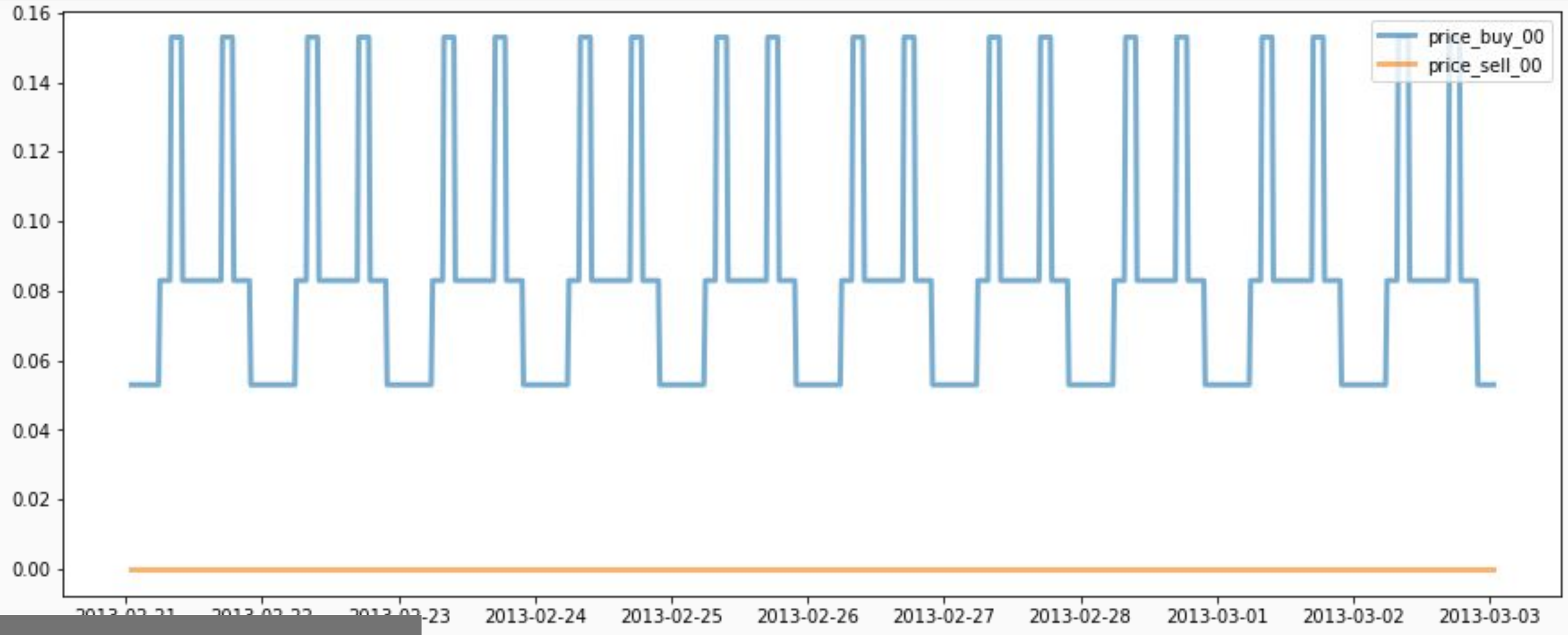
Consumption and photovoltaic production



Price schemes



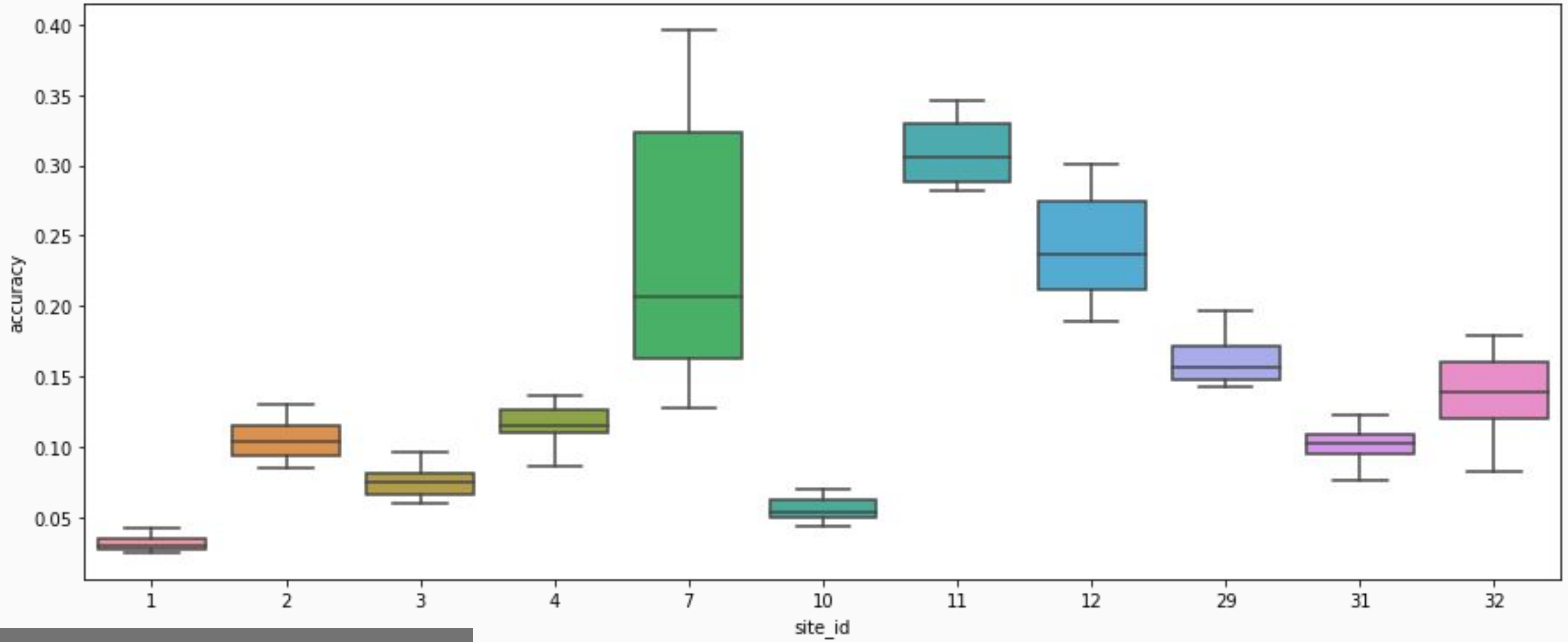
Price schemes



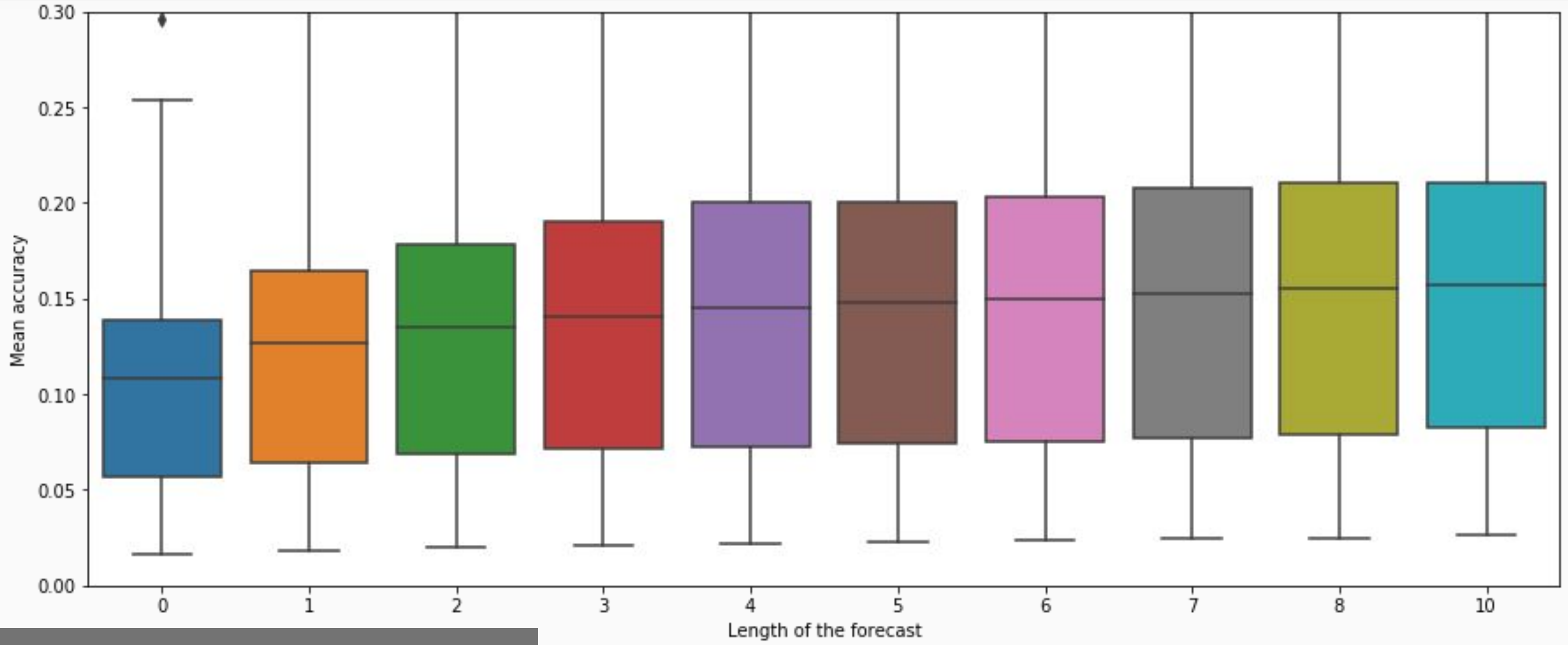
Findings on the data

- Clear periodicity of the data. Days and weeks. Weekends can be observed
- Different ratios between consumption and photovoltaic production between the different sites
- 2 prices and 3 prices schemes

Accuracy of the forecast on different sites



Influence of the forecast length



Conclusions about the forecast

- Different sites have different accuracy
- The longer the prediction the smaller the accuracy

First steps and initial approach

Initial approach

1. Find the optimum policy (this is assumed to be a slow process)
2. Train a fast model to learn the policy

That would met both the requirements of minimizing the cost and the execution time constraint

Genetic algorithms

- The genome is as long as the number of timesteps of the period
- Each gene will be 0 if we want to discharge the battery or 1 if we want to charge it
- The goal is to minimize the cost of the period

Genetic algorithms

- Very slow: 2 hours to optimize a period
- The optimization was not as good as future optimizations
- The search space is very big 2^960

Probably this is optimization is not a good choice

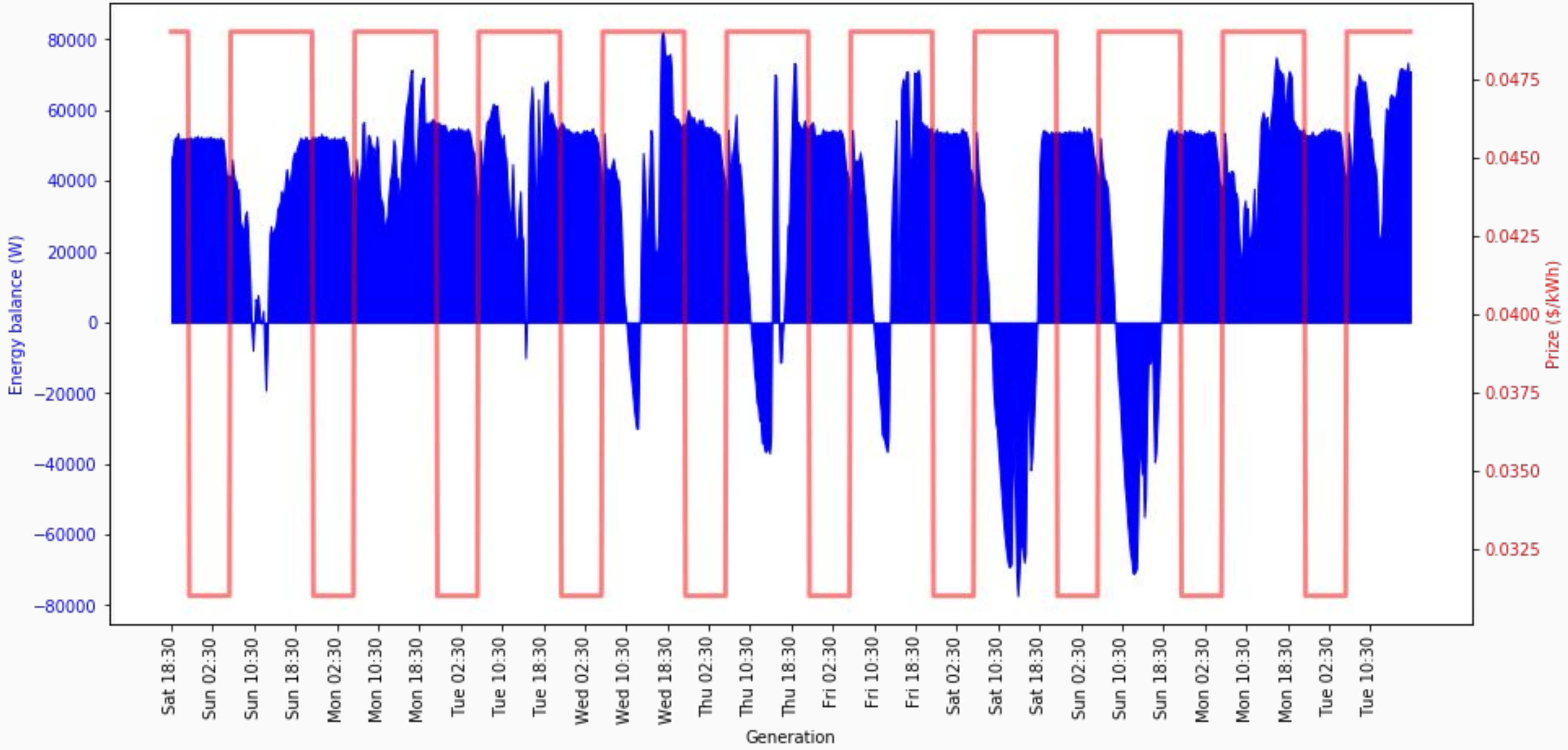
Probably we have to simplify the problem

Other perspective to the data

New representation of the data

- The consumption and photovoltaic production are not relevant. What matters is the energy balance
- Sell price is always constant, so we don’t need to plot it

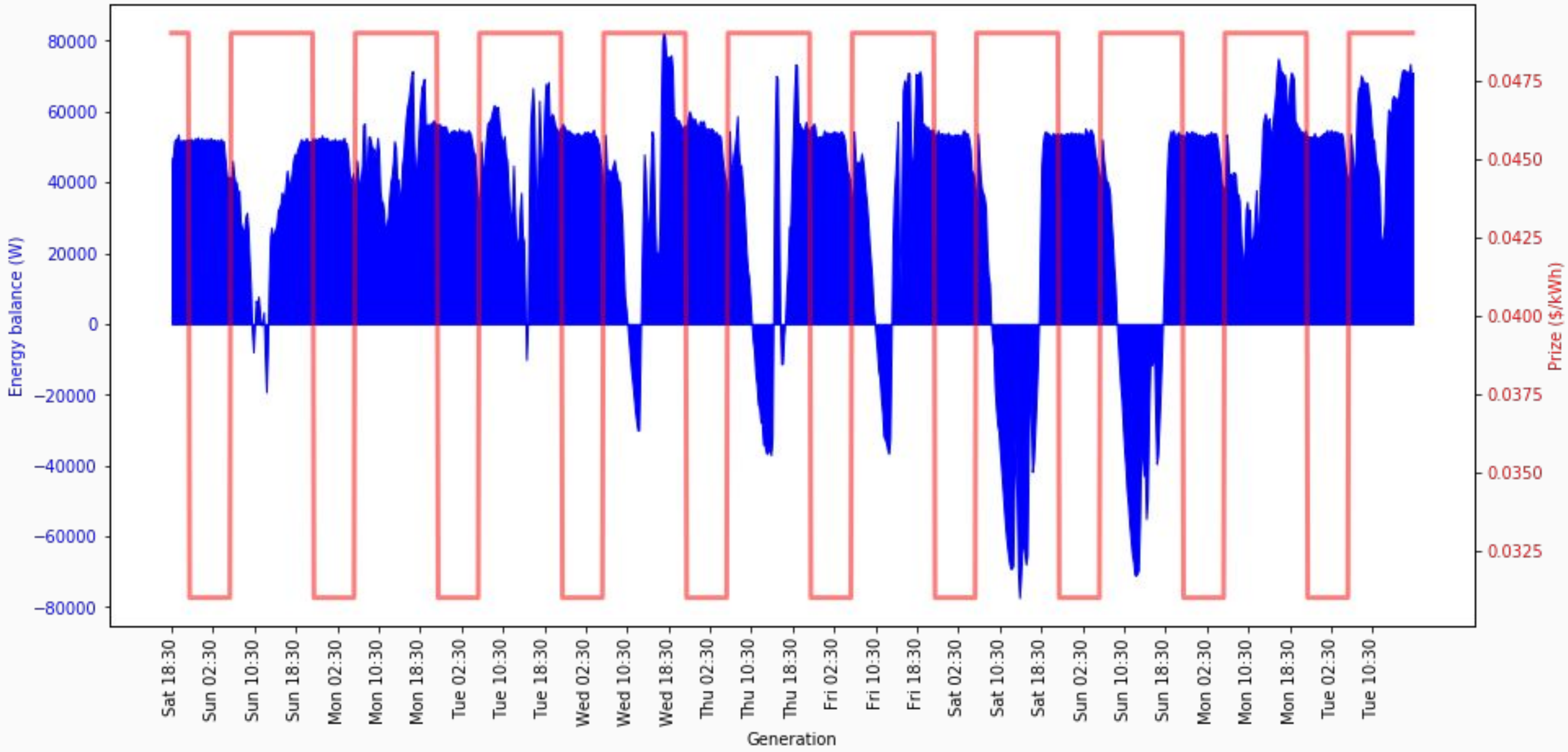
Energy balance and buy price



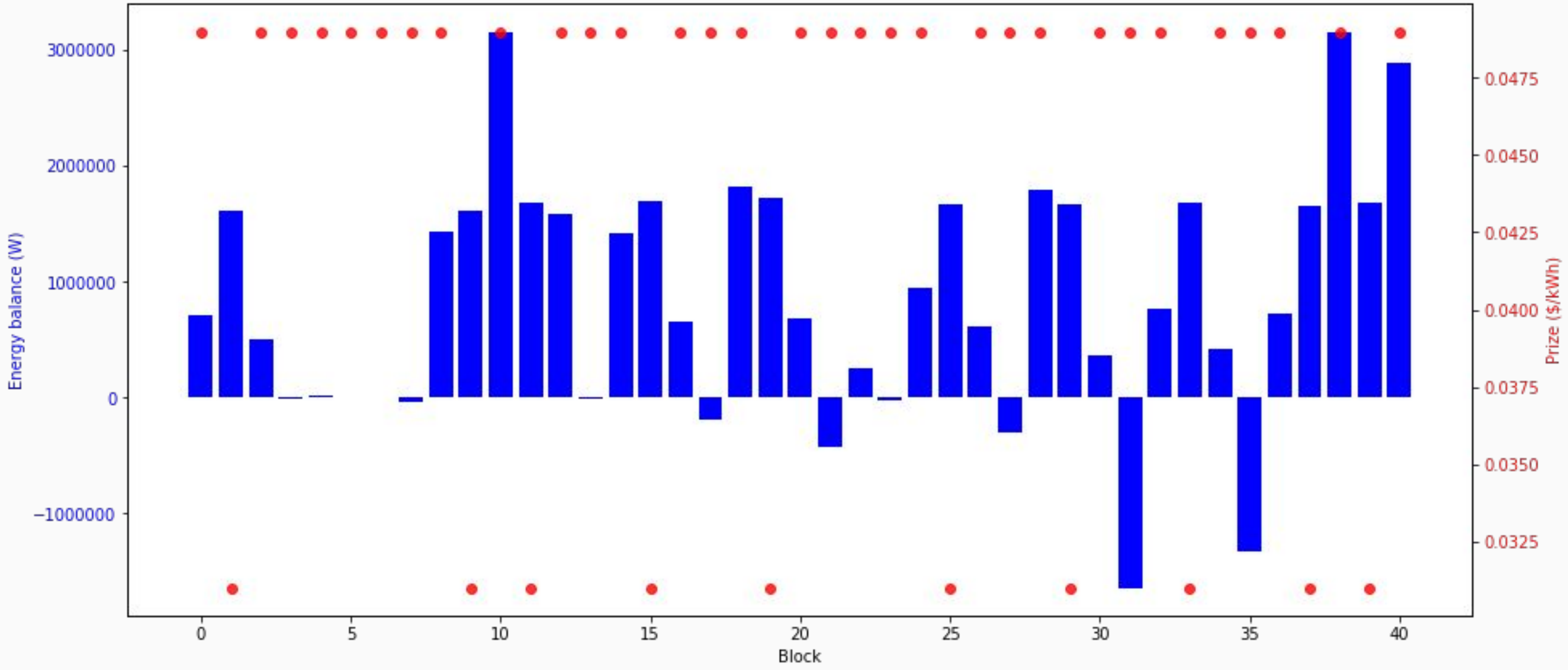
It’s possible to simplify the problem

- We can group the timesteps into bigger blocks
- The criteria for joining timesteps is that they have the same buy price.
    - If we have the same buy price on several consecutive timesteps it is not relevant if I buy energy on the first or the last timestep. The cost will be the same
- If the energy balance changes that is also a reason for breaking a block.
    - If the energy balance is negative then we have the opportunity to charge the battery using that energy. It is equivalent of buying that amount of energy at the sell price at that moment, which is equivalent to a change in the buy price

Energy balance and buy price before simplification



Energy balance and buy price after simplification



Effects of the simplification

We have reduced the length of the period **from 960 steps to just 40.**

The search space is now much more smaller, so the optimization algorithms will run much faster.

Dynamic programming

Dynamic programming (Wikipedia)

Is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions.

The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time at the expense of a (hopefully) modest expenditure in storage space.

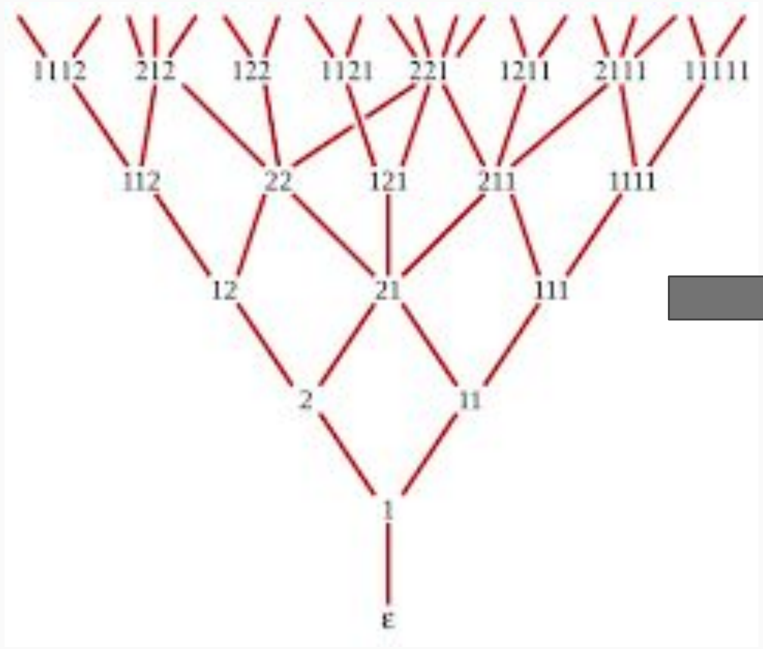
Dynamic programming

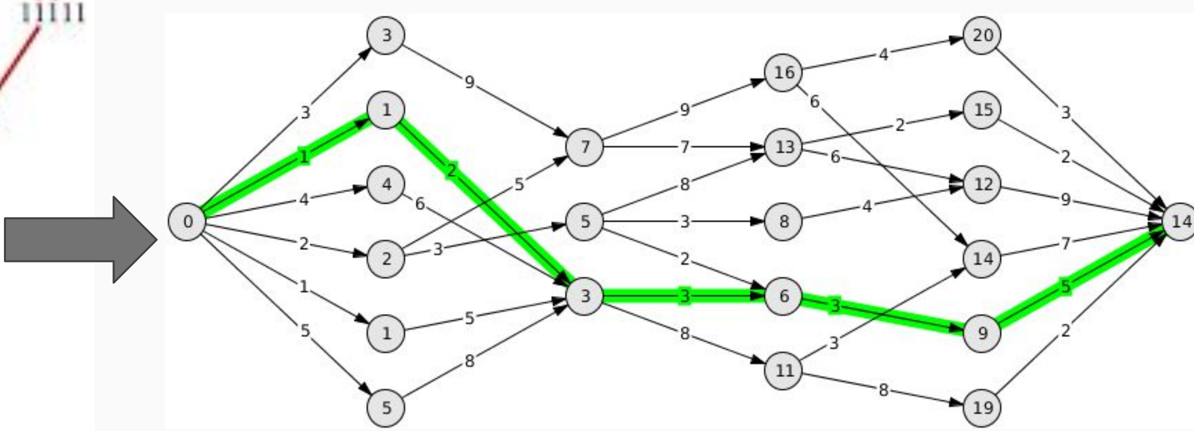
Useful in processes where the history is not relevant for taking decisions, the only relevant is the current state.

This is our case: the only relevant information is the current state of the battery. When the battery was charged or discharged is not relevant for taking decisions.

This simplifies a lot the search space

Simplification of the search space





Action pruning

Why action pruning

In order for dynamic programming to be fast the number of available actions on each state is very relevant.

The more the freedom on each state the more the bigger the search space and the slower the algorithm.

We have to careful select which actions are available on each state.

Available actions

- To do the opposite as the system. This allows to supply the exact amount of demanded energy or to charge the battery with the exact energy provided by the system
- *If we are on a block with low buy price* we can charge the whole battery
- *If the simulation is close to the end* we allow to do nothing or even to discharge the battery. This allows to arrive to the end without charge on the battery

Available actions

- *If the system is demanding energy but higher prices are expected soon* it is allowed to supply energy to the system but leaving enough energy for the incoming block with higher prices

Available actions

Notice that 3 of the 4 available actions are conditional.

In most of the cases there would be just one or two available actions

Summary of all the simplifications

- Period simplification. Reduces the length of the period
- Dynamic programming. Reduces the search space
- Action pruning. Reduces the search space

This allows to optimize the whole train set in less than 15 seconds. (8800 days of optimization). At the start of the challenge we needed 2 hours to optimize 10 days. The speedup is 4e5.

This speed eliminates the need to train a model to learn the policy. We can simply use this optimization to solve the challenge.

From high level policy to low level

We are using a simplification of the period for optimizing the policy. This way we get a high level policy.

However we have to take decisions on each timestep. So I had to implement methods for translating the high level policy into a low level policy.

Forecast inaccuracies

Forecast inaccuracies

Until now all the optimizations where done using perfect information. However during the simulation we only have access to forecasts of future values and to the previous real values.

The inaccuracy in the forecast reduces slightly the cost optimization. For the public test set the metric drops from -0.2173 to -0.1920 (the approximate meaning of this metric is that we save a 21% of the cost comparing to a building without the battery)

Improving the forecast

I tried different neural networks to improve the forecast. However the best solution was to use simple linear regression.

As input I used used:

- The forecast for the timestep
- The real values of the two previous timesteps

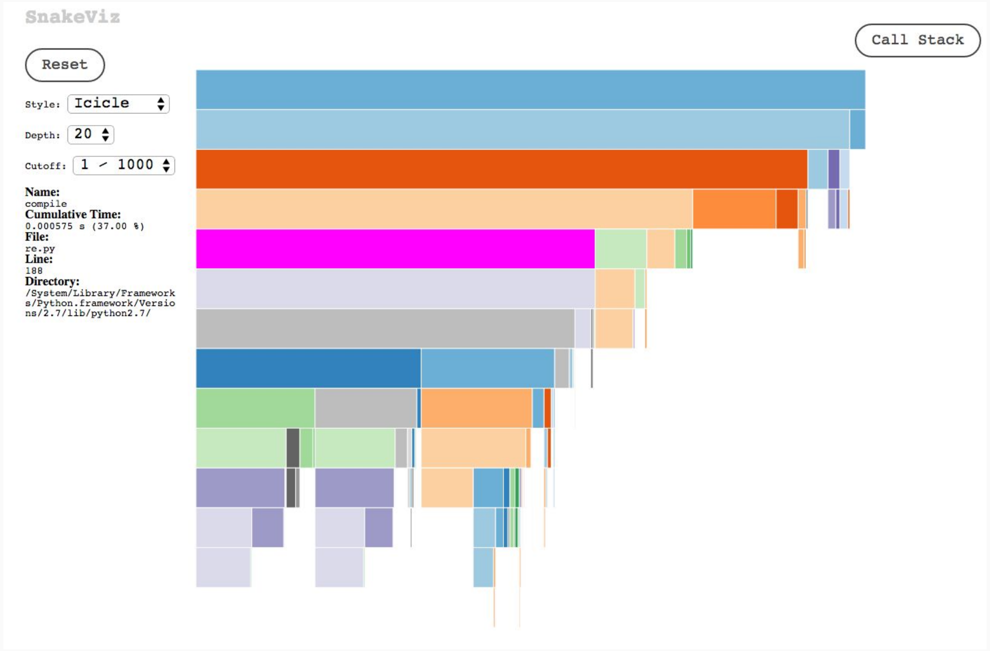
This allows to improve the score on public test set from -0.1920 to -0.1922

Other useful stuff

Cprofile and snakeviz

They helped a lot to make a faster implementation in code of the ideas previously described.

Clear visualization of the bottlenecks that guides the optimization work of the code



Style: Icicle
Depth: 20
Cutoff: 1 / 1000

Name:
compile
Cumulative Time:
0.000575 s (37.00 %)
File:
re.py
Line:
188
Directory:
/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/

Unit testing

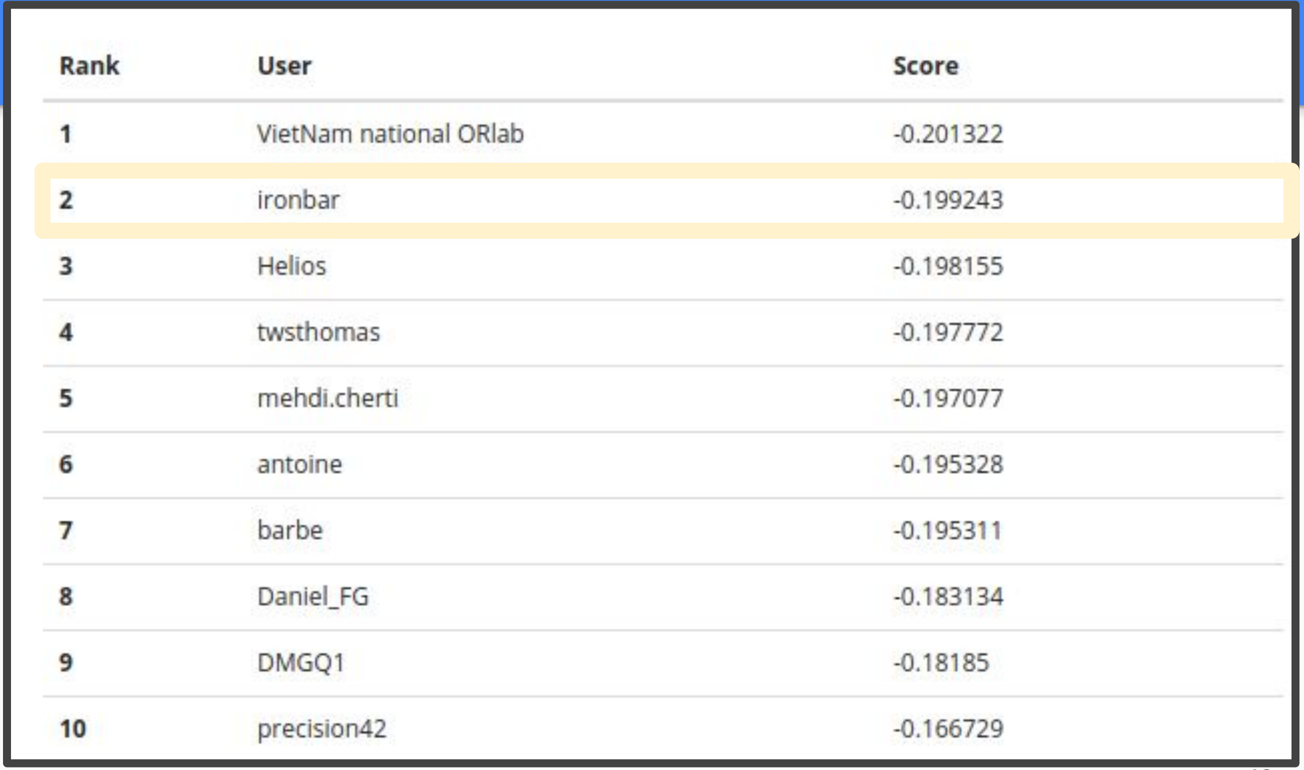
It was very helpful to make fast iterations being sure that nothing was broken when making changes to the optimizer

Results

Results

Public test set: -0.1922

Private test set: -0.1992



Rank	User	Score
1	VietNam national ORlab	-0.201322
2	ironbar	-0.199243
3	Helios	-0.198155
4	twsthomas	-0.197772
5	mehdi.cherti	-0.197077
6	antoine	-0.195328
7	barbe	-0.195311
8	Daniel_FG	-0.183134
9	DMGQ1	-0.18185
10	precision42	-0.166729

Links

https://www.drivendata.org/competitions/53/optimize-photovoltaic-battery/