ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA

CORSO DI LAUREA MAGISTRALE
IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in
Data Mining

New biologically inspired models towards understanding the Italian Power Exchange market

CANDIDATO
Sara Bevilacqua

RELATORE:
Chiar.mo Prof. Ing. Claudio Sartori

CORRELATORI
Maître de Conférences Célia da Costa Pereira
Professeur Frédéric Precioso

Anno Accademico 2016/17

Sessione III

Contents

Introduction

In recent years, new simulation and optimization techniques have been developed and improved. Now experts and developers are able to write and validate many types of computational models, looking for the one that better reproduces real scenarios and has the best forecasting performance. These techniques can be used in several areas, like in economics; for example, with the aim of providing good models that are able to reproduce as well as possible a specific market trend, or that can be used as decision support systems, in order to choose the better strategy and, therefore, maximize the profit.

This thesis focuses on a specific market: the electricity market in Italy, where the electricity industry has been liberalized and wholesale trading of electricity takes place in an organized market.

Several oligopolistic models of the liberalized power exchange have been proposed, based on different assumptions of rationality, learning and cognition. One of the most used tools is the agent-based model, which allows to simulate the competing interests between companies (i.e. agents): good performances may be obtained by means of a model that more adequately copes with heterogeneous learning agents, which operate in a highly complex and uncertain environment. In particular, agent-based computational models are built upon the assumption that the decision entity/agent learns to strategically behave in the market by iteratively exploring strategies and exploiting them on the basis of their relative performances[1].

Guerci and Sapio proposed in [1] an agent-based model, which simulates the Italian Power Exchange, taking into account a detailed and realistic market environment. Their model has been validated on historical data. However, their model does not allow an independent coevolution of the strategies of the companies: the agents depend on competitors previous strategies and therefore, they share their own private strategies with competitors. The aim of this thesis is to fill this gap. More precisely, we want to make agents completely autonomous, with the capability to predict and anticipate competitors' choices without exchanging information.

The thesis is structured as follows. Chapter 1 describes the structure of the Italian Power Exchange market and the actual project modeled on it, illustrating, at the end, the results. Chapter 2 proposes a refactoring of the actual project, applying Object Oriented programming and alternative algorithms. Chapter 3 introduces an extension that makes agents more intelligent than ones introduced previously. Chapter 4 presents some related works, in order to see differences and similarities with our proposal. Chapter 5 concludes the thesis and presents the future work.

Chapter 1

The IPEX and the actual model

In this section, I will present a realistic model of the Italian wholesale electricity market and its first implementation, proposed by Eric Guerci[1].

1.1 Backgroud

In Italy, day-ahead wholesale trading of electricity takes place in the Italian Power Exchange (IPEX), run by Gestore dei Mercati Energetici (GME), a State-owned company. The Ipex day-ahead market is a closed, uniform-price (non discriminatory), double auction. Each day, market participants can submit bids concerning each hour of the next day. Demand bids are submitted by large industrial consumers, by independent power providers (who serve final users) and by Acquirente Unico, a State-owned company that takes care of final customers.

This Italian Market model considers a two-settlement market configuration with a generic forward market and the day-ahead market (DAM). The DAM price value is commonly adopted as underlying for forward contracts; therefore, in what follow we will refer to DAM as the spot (i.e. immediate, instant) market session for simplicity. The forward market session is modeled by assuming a common and unique forward market price P^f for all market participants and by determining exact historical quantity commitments for each generating unit.

A generating company (GenCo) g (with g = 1, 2, ..., G, where G is the number of GenCos) owns N_g generators. Each electrical generation unit i (where i=1, 2, ..., N_g) has lower $\underline{Q}_{i,g}$ and upper $\bar{Q}_{i,g}$ production limits, which define the feasible production interval for its hourly real-power production level $\hat{Q}_{i,g,h} = \hat{Q}^f_{i,g,h} + \hat{Q}^s_{i,g,h}$ ([MW]) $\underline{Q}_{i,g} \leq \hat{Q}_{i,g,h} \leq \bar{Q}_{i,g}$ where $\hat{Q}^f_{i,g,h}$ and $\hat{Q}^s_{i,g,h}$ are respectively the quantity sold in the forward market and the quantity accepted in the DAM. It is assumed that the company g takes a long position in the forward market (it means that the company makes agreement with the market operator with large advance) for each owned generator i_g, corresponding to a fraction $f_{i,g,h}$ (where h indicates the hour of the day) of its hourly production capacity, that is $\hat{Q}^f_{i,g,h} = f_{i,g,h}\bar{Q}_{i,g}$. The value of such fraction varies throughout the day, indeed forward contracts are commonly sold according to a standard daily profiles. In this model, the value of $f_{i,g,h}$ has been estimated by looking at historical data and thus corresponds to a realistic daily profile for each generator.

The revenue [€/h] from forward contracts for company g is:

$$R^f_{g,h} = \sum_{i=1}^{N_g} \hat{Q}^f_{i,g,h} P^f \tag{1.1}$$

The spot revenue per hour for GenCo g is obtained as follows:

$$R^s_{g,h} = \sum_{z=1}^{Z} \hat{Q}_{z,g,h} P^s_{z,h} \tag{1.2}$$

where $P^s_{z,h}$ is the price in the spot market in zone z at hour h.

The total cost function [€/h] of the i^{th} generator of GenCo g is given by the following quadratic formula:

$$C_{i,g,h}(\hat{Q}_{i,g,h}) = (P_{fl} + P_{co2}h_{fl}) \cdot (a_{i,g}\hat{Q}^2_{i,g,h} + b_{i,g}\hat{Q}_{i,g,h} + c_{i,g}) \quad (1.3)$$

where P_{fl} and h_{fl} ([€/GJ]) are the price of the fuel fl which is used by the i^{th} generator and the conversion value to determine the amount of CO2 generated by the combustion of a unit of fuel fl ([GJ]), respectively. P_{co2} is the price of carbon permits in the European Emission Trading System - EUETS. In the model, it is assumed a mark-to-market hypothesis, that is, ETS prices are updated on a daily basis according to current values. The coefficients $a_{i,g}$ ([GJ/MWh^2]), $b_{i,g}$ ([GJ/MWh])and $c_{i,g}$([GJ/h]) are assumed constant over time, but vary across power plants with different technologies and efficiency levels. The constant term $(P_{fl} + P_{co2}h_{fl}) \cdot c_{i,g}$ corresponds to the no-load cost, i.e. quasi-fixed costs that generators bear if they keep running almost at zero output. However, these costs vanish once shutdown occurs.

Finally, the total profit per hour [€/h] for GenCo g is equal to:

$$\pi_{g,h} = R^s_{g,h} + R^f_{g,h} - \sum_{i=1}^{N_g} C_{i,g,h}(\hat{Q}_{i,g,h}) \quad (1.4)$$

In this model only thermal power plant are considered, as they represent around three fourth of the national gross production capacity. Furthermore, the remaining national production (hydro, geothermal, solar and wind) and imported production can be reasonably modeled as quantity bids at zero price. Imports correspond in general to power generated abroad by cheap technologies, such that hydro or nuclear power plants, coming mainly from France and Switzerland. In any case, exact historical values have been assumed for both hydro and imports.

The considered set of thermal power plants consists up to 224 generating units, comprising 5 different technologies, i.e. Coal-Fired (CF), Oil-Fired (OF), Combined Cycle (CC), Turbogas (TG) and Repower (RP). These power plants are independently owned by Gencos.

The number of generation companies and generating units offering in the DAM varies throughout the day. Based on historical data, it has determined for each period (day and hour) the active thermal power plants, i.e. the thermal power plants that offered in DAM. What plants were actually present is supposed to be common knowledge, since bid data are publicly available on the power exchange website with a one-week delay. For each power plant in the dataset, information on the maximum and minimum capacity limits is available, as well as on the parameters of the cost function, as defined in equation 1.3. These data are used to calibrate the computational model in order to run realistic simulations for specific days.

On its i^{th} generator, GenCo g submits to the DAM a bid consisting of a pair values corresponding to the limit price $P^s_{i,g,h}$ ([€/MW]) and the maximum quantity of power $Q^s_{i,g,h} \leq \bar{Q}_{i,g} - \hat{Q}^f_{i,g,h}$([MW]) that it is willing to be paid and to produce, respectively. After receiving all generators' bids, the market operator clears the DAM by performing a social welfare maximization, subject to the following constraints: the zonal energy balance (Kirchhoff's laws), the maximum and minimum capacity of each power plant and the inter-zonal transmission limits. It is worth nothing that the Italian demand curve in the DAM is price-inelastic, i.e. it is unaffected when the price changes. Therefore, the social welfare maximization can be transformed into a minimization of the total reported production costs, i.e. of the bid prices (see eq. 1.5). This mechanism determines both the unit commitments for each generator and the Locational Marginal Price (LMP) for each bus. However, the Italian market introduces two slight modifications. Firstly, sellers are paid the zonal prices (LMP), whereas buyers pay a unique national price (PUN, Prezzo Unico Nazionale) common for the whole market and computed as a weighted average of the zonal prices with respect to the zonal loads. Secondly, transmission power-flow constraints differ according to the flow direction.

The factor which has to be minimized by solving the linear program is the following:

$$\min \sum_{g=1}^{G} \sum_{i=1}^{N_g} P^s_{i,g,h} \hat{Q}^s_{i,g,h} \tag{1.5}$$

It is subjected to the following constraints:

- Active power generation limits:
  $\underline{Q}_{i,g} \leq \hat{Q}_{i,g,h} = \hat{Q}^s_{i,g,h} + \hat{Q}^f_{i,g,h} \leq \overline{Q}_{i,g}$ [MW]
- Active power balance equations for each zone z:
  $\sum_{g=1}^{G} \sum_{j \in z} \hat{Q}^s_{j,g,h} - Q_{z,load,h} = Q_{z,inject,h}$ [MW]
  being $\sum_{g=1}^{G} \sum_{j \in z} \hat{Q}^s_{j,g,h}$ the sum over all generators located in zone z, $Q_{z,load}$ the load demand at zone z and $Q_{z,inject}$ the net oriented power injection in the network at zone z.
- Real power flow limits of line l, in the grid:
  $Q_{l,st} \leq \bar{Q}_{l,st}$ [MW]
  $Q_{l,ts} \leq \bar{Q}_{l,ts}$ [MW]
  being $Q_{l,st}$ the power flowing from zone s to zone t of line l and $\bar{Q}_{l,st}$ the maximum transmission capacity of line l in the same direction. $Q_{l,st}$ are calculated with the standard DC power flow model.

The solution consists of the set of the active powers $\hat{Q}^s_{i,g,h}$ generated by each plant i and the set of zonal prices P^s_z (LMPs) for each zone $z \in [1,2, \dots, Z]$.

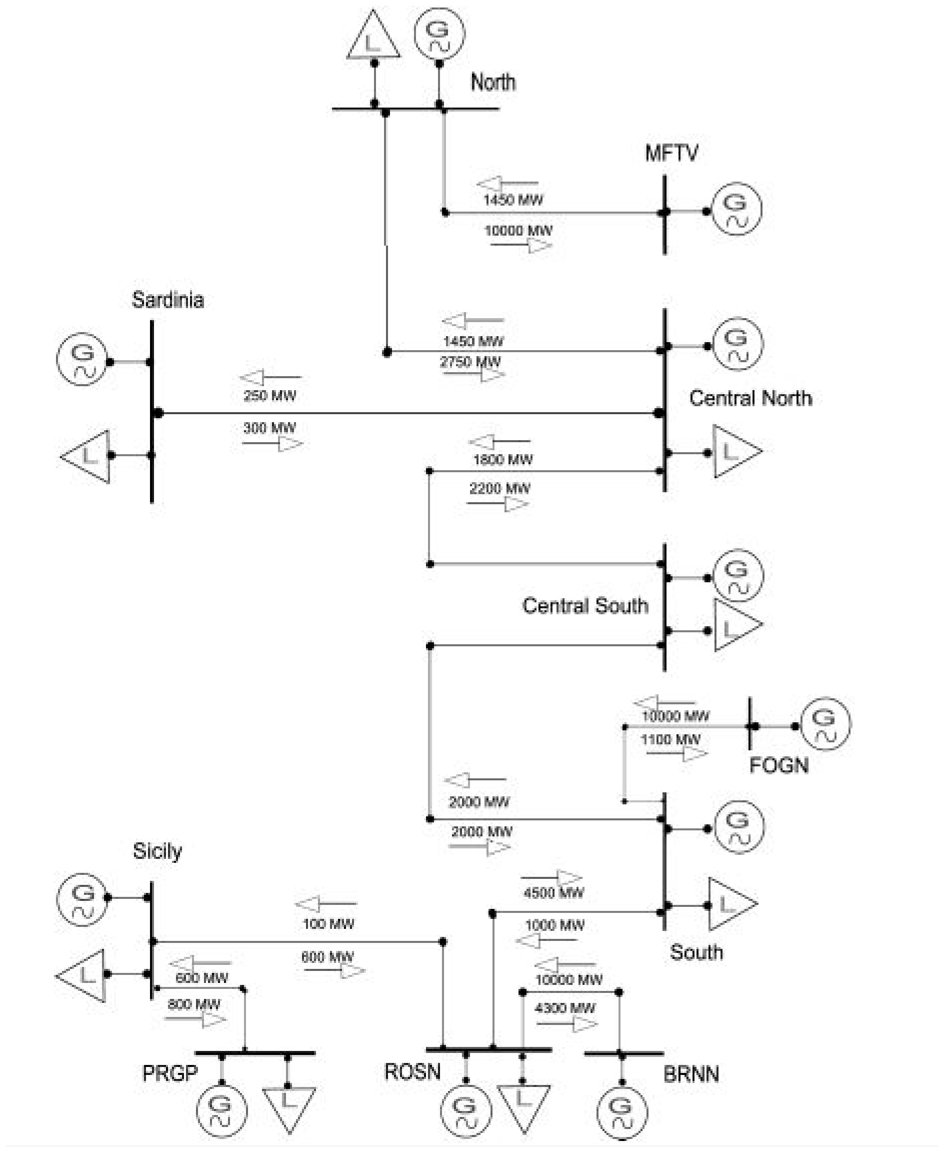


Figure 1.1: The Italian grid model

The adopted market clearing procedure requires the definition of a transmission network. The grid model considered (Figure 1.1) reproduces exactly the zonal market structure and the relative maximum transmission capacities between neighboring zones of the Italian grid model. It corresponds to the grid model defined by the Italian transmission system operator, i.e. TERNA S.p.A., which is adopted by the market operator. The grid comprises 11 zones (BRNN (BR), Central North (CN), Central South (CS), FOGN (FG), MFTV, North (NO), PRGP (PR), ROSN (RS), Sardinia (SA), Sicily (SI), South (SO)) and 10 transmission lines depicting a chained shape, which connects the North to the South of Italy. The different values of maximum transmission capacities for both directions of all transmission lines are also reported. The values are realistic, but just indicative. Exact values for specific days that have been simulated have then been used. Figure 1.1 further shows also the distribution of generators in the network and the representative load serving entities (LSE) at a zonal level. The neighboring country virtual zones (point of interconnection with neighboring countries) have been neglected in the definition of the grid model, but their contributions to national loads of production capacities have been adequately included in the simulations.

1.2 Model description

In the following, I describe how generation companies are modeled in the agent-based simulator and how the simulation is structured.

Each Genco g must submit to the DAM a bid, i.e. a set of prices for each of its own power plant (see Equation 1.5). Therefore, each Genco has an action space for each power plant, which is a set of possible prices that the Genco can choose. This set is made by following this formula:

$$AS_{i,g} = MC_{i,g} \cdot MKset \tag{1.6}$$

where $AS_{i,g}$ is the action space of power plant i of Genco g, $MC_{i,g}$ is the marginal cost of the same power plant and the $MKset = [1.00, 1.04, \ldots, 5.00]$ is the mark-up levels set. In this way, Gencos are sure to not propose a price lower than the costs.

The multi-agent system is depicted in Figure 1.2. G GenCos are reported on the top of the figure. These GenCos repeatedly interact among each other at the end of each run $r \in\{1,\ldots,R\}$, that is they all submit bids to the DAM according to their current beliefs on opponents' strategies. At the beginning of each

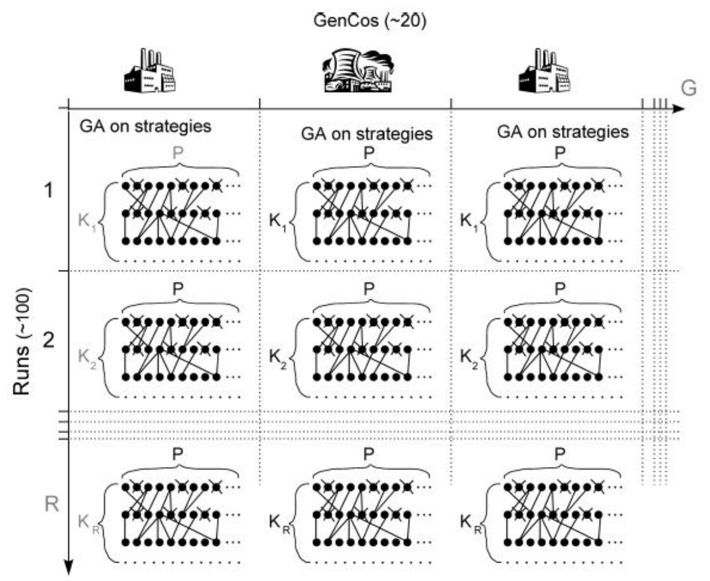


Figure 1.2: A schematic representation of the learning algorithm

run r, GenCos need to study the current market situation in order to identify a better reply to the opponents, to be played at run r+1.

In order to choose the most competitive strategy, Gencos need repeatedly solve the market for different private strategies. A genetic algorithm is adopted, in order to keep a large population of candidate strategies and to improve at the same time their fitness/performance in the market. Thus, a population of size P (see Figure 1.2) of strategies is defined, which will evolve throughout the K_r generations. In order to reduce the complexity, we collect the power plant of a Genco g in the same zone and with the same technology under a subAgent, which will apply a common strategy for its power plant.

Genetic algorithm is a computational technique inspired by biology: as an individual of a population in the real world adapts itself to the environment in order to survive and reproduce, in the same way a possible solution shall be adapted to solve the related problem. Genetic algorithms reproduce the main phases of the biological evolution process: crossover, in which new children are obtained by combining pairs of parents in the current population, mutation, in which some genes of children are randomly changed, and selection, in which only the best solutions are selected. In order to understand if a solution is good or not, a fitness function of the problem must be defined.

A strategy, depicted as a black dot in Figure 1.2, , is a set of in-

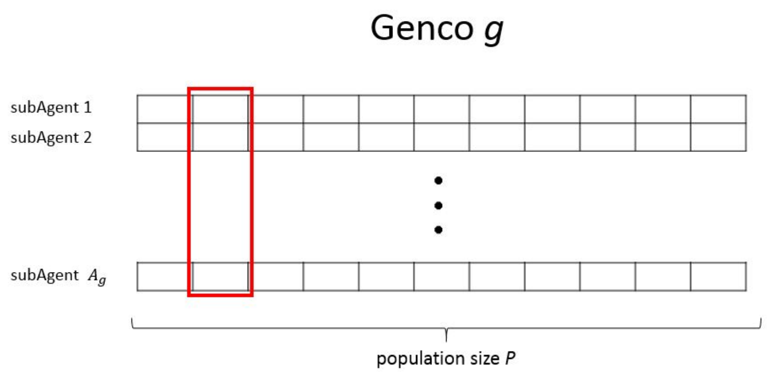


Figure 1.3: Representation of Genco's populations: a population of prices for each generating unit managed. In this case, Genco g manages A_g subAgents. The red rectangle identifies a strategy, which is the input of the fitness function, i.e. the genome.

dexes that identify a single price in the action space. Thanks to the approximation made by the introduction of subAgents, instead of having, for the Genco g, N_g indexes, we have A_g indexes, where A_g is the number of subAgents of Genco g ($A_g \leq N_g$). Therefore, each Genco owns a set of "sub-populations", one for each subAgent. This concept is shown in figure 1.3: each line corresponds to a "sub-population" of a specific subAgent of Genco g, i.e. each little square represents a price index in the action space; stacking all the subAgents and taking into account columns, we have the population of strategy of Genco g and one strategy (the black dot Figure 1.2) is represented by the red rectangle.

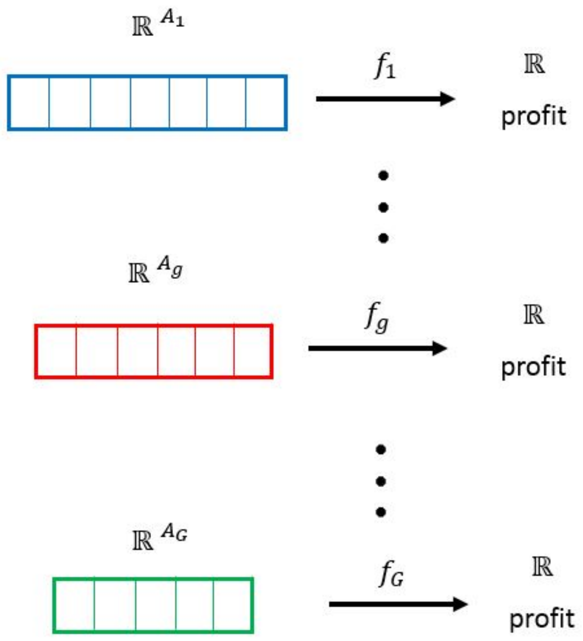


Figure 1.4: Gencos fitness functions of the genetihic algorithms. The red rectange in this figure has the same meaning of the one in Figure 1.3

The adopted genetic algorithm is a sort of meta-genetic algorithm, since it accepts a function. The domain of the function of a Genco g is $\mathbb{R}^{A_g}$ and the codomain is $\mathbb{R}$, because the function returns the profit, i.e. the fitness value, of the strategy (see Figure 1.4). Such a fitness reinforces the weight of the power plants with low prices.

At the end of each run r, each GenCo bids to the market by selecting one strategy belonging to its current population of candidates. The selection is done according to a probabilistic choice model in order to favor the most represented strategy in the population, i.e. the one that has best responded to the evolutionary pressure by ensuring the highest fitness. The functional form of the probabilistic choice model is the following:

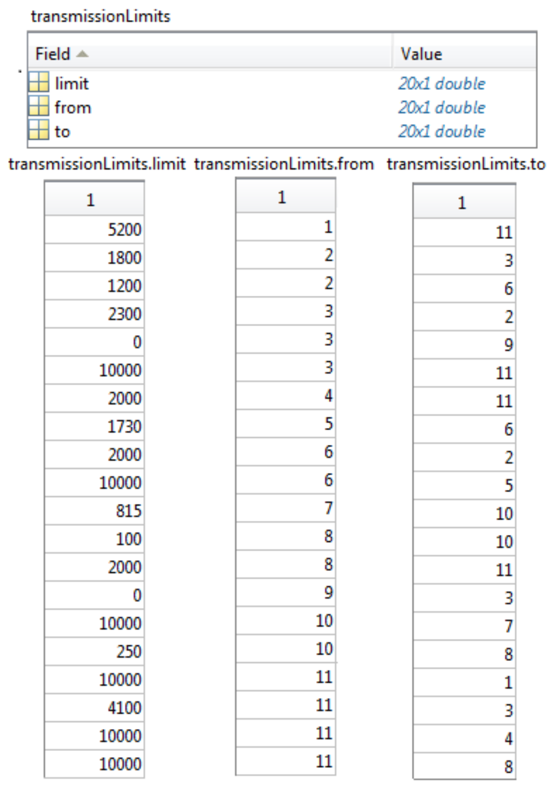
$$\Pr(s \mid r+1) = \frac{e^{f_g(s)/\lambda}}{\sum_{s' \in S_g} e^{f_g(s')/\lambda}} \tag{1.7}$$

where $\Pr(s \mid r+1)$ denotes the probability of selecting action s at period $r+1$, and S_g is the final population of strategies produced by the genetic algorithm.

1.3 Data

The simulation needs a multitude of data, which will be described in this section.

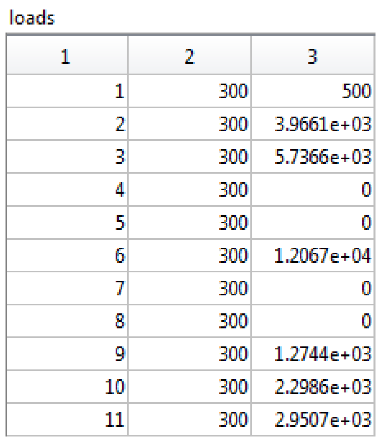
The Italian electrical grid, depicted in Figure 1.1, is based on the *transmissionLimits* structure which represents the transmission limits within the grid. This structure contains three arrays, which give the information about the limits in both direction for each link between zones (see figure 1.5) .
The demand of energy for each zone is provided by the *loads* matrix. The first column contains the zones, the second one contains the maximum limit prices and the third one contains the demand quantities of electricity.


transmissionLimits

Field	Value
limit	20x1 double
from	20x1 double
to	20x1 double

transmissionLimits.limit	transmissionLimits.from	transmissionLimits.to
1	1	1
5200	1	11
1800	2	3
1200	2	6
2300	3	2
0	3	9
10000	3	11
2000	4	11
1730	5	6
2000	6	2
10000	6	5
815	7	10
100	8	10
2000	8	11
0	9	3
10000	10	7
250	10	8
10000	11	1
4100	11	3
10000	11	4
10000	11	8

Figure 1.5: *transmissionLimits* structure

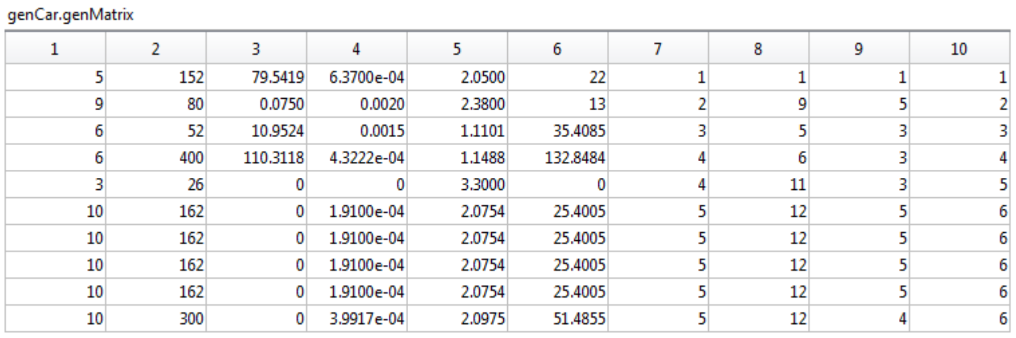

loads

1	2	3
1	300	500
2	300	3.9661e+03
3	300	5.7366e+03
4	300	0
5	300	0
6	300	1.2067e+04
7	300	0
8	300	0
9	300	1.2744e+03
10	300	2.2986e+03
11	300	2.9507e+03

Figure 1.6: *loads* matrix

All the characteristics of the generating power plants are collected in the genCar structure:

- *genCar.operators* contains the names of Gencos (for example ATEL, EDISON, ...).
- *genCar.technology* contains the names of technologies(for example carbone, ccgt combinato, ...).
- *genCar.fuelCosts* contains the prices of the fuels.
- *genCar.genMatrix* contains all the information related to Italian power plants. The columns indicate respectively: zone, maximum production quantity, minimum production quantity, coefficient a, coefficient b, coefficient c (see section 1.1), Genco's id, technology index, fuel index and subAgent's id.


genCar.genMatrix

1	2	3	4	5	6	7	8	9	10
5	152	79.5419	6.3700e-04	2.0500	22	1	1	1	1
9	80	0.0750	0.0020	2.3800	13	2	9	5	2
6	52	10.9524	0.0015	1.1101	35.4085	3	5	3	3
6	400	110.3118	4.3222e-04	1.1488	132.8484	4	6	3	4
3	26	0	0	3.3000	0	4	11	3	5
10	162	0	1.9100e-04	2.0754	25.4005	5	12	5	6
10	162	0	1.9100e-04	2.0754	25.4005	5	12	5	6
10	162	0	1.9100e-04	2.0754	25.4005	5	12	5	6
10	162	0	1.9100e-04	2.0754	25.4005	5	12	5	6
10	300	0	3.9917e-04	2.0975	51.4855	5	12	4	6

Figure 1.7: First lines of *genCag.genMatrix*

- *genGar.represPPnotREF* provides production quantity data of power plants which sell energy a zero euros.

1.4 Implementation

This part shows some of the most significant features implemented in the Matlab project, developed by Guerci and colleagues. The implementation reflects all the aspects discussed previously.

The project is object oriented and objects have a hierarchical structure, as told before:

- ***@sellerEM***: this object represents a Genco and it has one or more subAgents;
- ***@gaAgent***: it is a subAgents and it owns one or more power plants.

These data are retrieved from columns 7 and 10 of the ***genMatrix*** in the dataset (in which each line provides information about all the power plants), which respectively contain the id numbers of the seller objects and the id numbers of the subAgent.

Before starting the simulation, there is a configuration and initialization phase, in which the action space is created for each subAgent by the function actionSpaceCreator. This function has 4 outputs:

- *actionSpace*, a 100×4 matrix: for each rows, it contains the zone, the proposal price (see section 1.2), the max quantity and the min quantity of power production. Only the second column is varying.
- *coeff*, a matrix which contains the three coefficients $a_{i,g}$, $b_{i,g}$ and $c_{i,g}$ (see section 1.1). They are retrieved from column 4, 5 and 6 of the *genMatrix.*
- *capConstr*, a matrix which contains the two capacity constraints, retrieved from column 2 and 3 of the *genMatrix.*
- *maxProfit*, the maximum profit calculated with the quadratic formula described above (see equations 3 and 4).

After this first part of settings and initialization, the simulation starts. It is organized in session and for each session there are *numIterations* runs (*numIterations* is set in the first phase). Each run represents a bid session in the DAM and it is implemented by the function ***play***. A bid session is organized as follows.

Firstly, all the sellers' choices are collected in a structure by the ***chooseStrategy*** function: it selects the best strategy elaborated in the previous run for each GenCo, by applying equation 1.7.

Then there is a phase in which each GenCo learns the actual market situation. This behavior is implemented by the following two functions :

- ***MGPMKP***: this function simulates a market session by solving a linear programming problem. In detail, it invokes the function
  `[SDAcc,fval,exitflag,output,lambda]=`
  `linprog(prices,A,B,Aeq,Beq,LB,UB,option);`
  where:
  - *prices* is a vector of prices, which represents the objective function that has to be maximized.
  - A and B represent the linear inequality constraints $A \cdot x \leq B$. In this model, A is an identity matrix and B contains all the supply, demand and transmission limits.
  - Aeq and Beq represent the linear equality constraints $Aeq \cdot x = Beq$. In this case, they depict the power balance (Kirchhoff's laws: the total sum of the electrical powers must be zero), therefore Beq is a vector of zeros and Aeq contains the supply, demand, input (from another transmission line) and output (to another transmission line) for each zone.

  - LB and UB are respectively the lower bounds and the upper bounds, whose values are given by the physical limits of the system.

  The solution provides two important information: the production quantity, in the vector *SDAcc*, and the LMPs, in the structure *lambda.eqlin*. These data are also used to calculate the PUN.

- ***getPayoffs***: this function gets the revenues from the market mechanism. Firstly, it collects the actual sellers' choices, then it runs the market (by invoking the previous function) and at the end, it estimates the profits for each power plant and subAgent. The profits are estimated with a uniform auction and quadratic production function. The function has many outputs, which are used by sellers after, during the elaboration of a strategy: profits and supply information, the actual sellers' choices, the PUN and the zonal prices (LMPs).

At this point, when each GenCo has learned the actual situation of the market, they opportunely elaborate a new strategy by invoking the ***update*** function: each GenCo tests a series of strategies, by applying the Matlab genetic algorithm. ***estimateFitnessMGP*** is the fitness function implemented in this project: firstly, it recovers all the data about other sellers' choices, then it plays privately the market game by invoking the *MGPMKP* function with the specific strategy indicated by the genome and, at the end, it updates the fitness value (i.e. the profit) for this specific element of the population. Therefore, at the end of this phase, each strategy has its own fitness value.

These passages are repeated for each run of each session. At the end, a statistical analysis is computed, in order to validate and evaluate the goodness of the simulation.

1.5 Results

This model has been implemented and then validated at a macro level by studying the relative ability to reproduce the daily PUN time series[1]. The model tends to overestimate the prices in off-peak hours, as depicted in the figure below. A possible rationale is that the model does not consider that some thermal technologies and power-plants are not so flexible to switch off and on overnight. Shutdown and startup costs are not negligible thus pushing Gencos to bit below marginal costs their minimum power-plants operating capacity. The results in Figure 1.8 are completely different: in the first case the model exhibits a clear overestimation of price levels for all hours, in the second case the model predicts better except for the price levels of off-peaks hour.

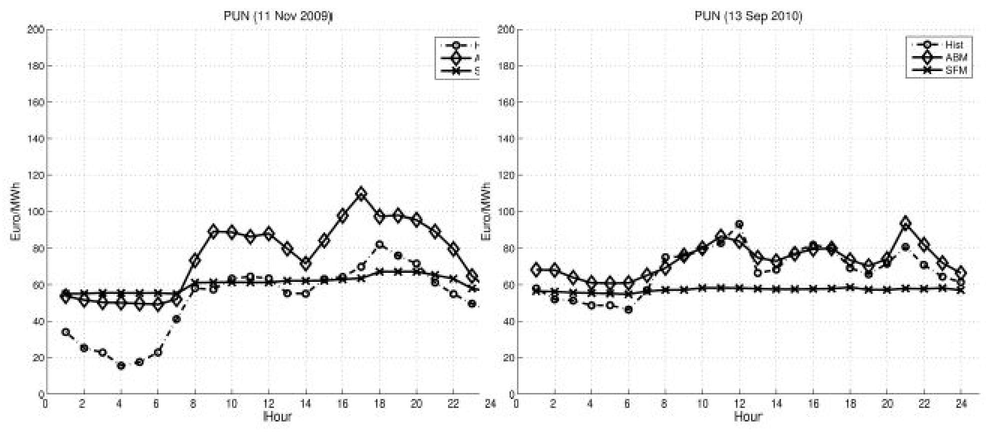


Figure 1.8: Real and simulated PUNs for the 2009-11-11 (left) 2010-09-13 (right). The Hist line (with circles) reports historical outcomes. The ABM line (with diamond) shows the agent-based simulation results, whereas the SFM represents results of another model (Supply Function Model by Klemperer and Meyer [1])

Chapter 2

A new project

In this section, I will propose a new model for the Italian Power Exchange, trying to improve performance of the actual one.

2.1 Motivation and problem analysis

Analyzing the model depicted in the previous section, some problems emerge. The model has a valid background and it reflects the main aspects of the considered context (the IPEX), but it simplifies and neglects some aspects. From a research point of view, the current model simplifies the managing of power plants, since subAgents adopt common strategies. From a technical point of view, it is a first attempt to use objects, but it does not consider a series of good practices of Object Oriented programming: many data are replicated in two or more objects and therefore it is not clear the behavior separation between these objects.

In detail, there are three main problems:

1. There is not a definite separation between the *@electrictyMarket* object and the market implementation (in this case *@GMEMKP* object): the first one represents the Italian electricity market, while the second one implements the

concept of GME (Gestore dei Mercati Energetici) based on markup levels and it could be substituted by another model. The problem is given by the fact of the two objects have many data replicated. This can cause confusion, because it is difficult to understand the exact behavior of the two objects.

2. The relation between *@sellerEM*, *@agent* and *@gaAgent* objects is not well defined. Therefore, it is difficult to propose and test alternative optimization algorithm.

3. The project is based on objects, even if it is implemented in Matlab, but there is not a very Object Oriented programming methodology.

In order to solve these problems, which make it difficult to maintain and extend the model, in the following I will propose a new OO agent-based model and its implementation.

2.2 Technical refactoring of the model

In recent versions, Matlab completely supports the object oriented programming [2], therefore the project could include interfaces, abstract classes and support class hierarchies. These concepts are very important to project an open and extensible system.

The main object of the new model (Figure 2.1) is the electricity market, which contains all the basic information about the IPEX, discussed in the previous chapter: the Italian grid model and its transmission limits, zonal loads and all the information (capacities and characteristics) for each power plant. In this model, the *ElectricityMarket* takes into account the list of Gencos and the effective implementation of the GME.

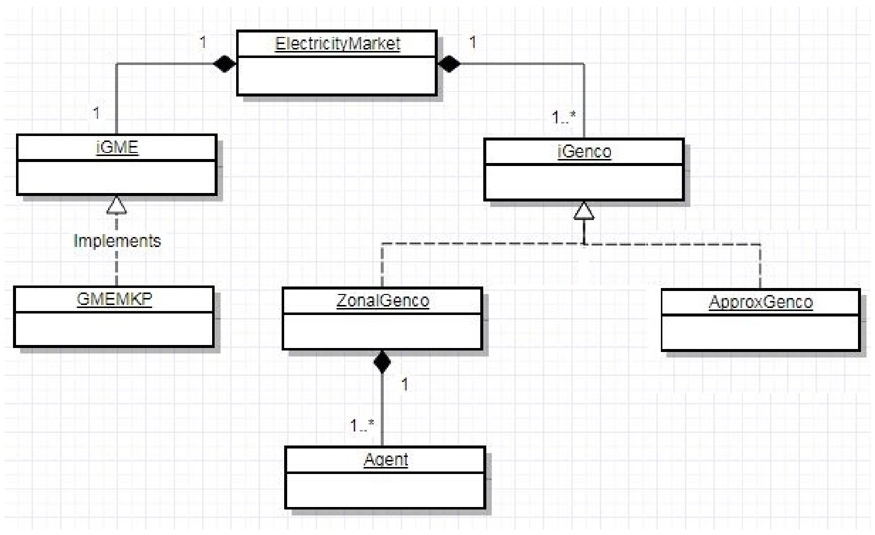


Figure 2.1: UML class diagram of the new model

The *iGME* is an interface, which allows to define a specific behavior for the construction of the market. Many implementations can be developed, but Figure 2.1 shows only the mark-up levels model *GMEMKP*, depicted in section 1.2.

iGenco is an interface for Gencos. The system supports two implementations, which represents the different scenarios supported in the previous version:

1. *ApproxGenco*: Gencos adopt a common strategy for power plants in the same zone (see section 1.2).

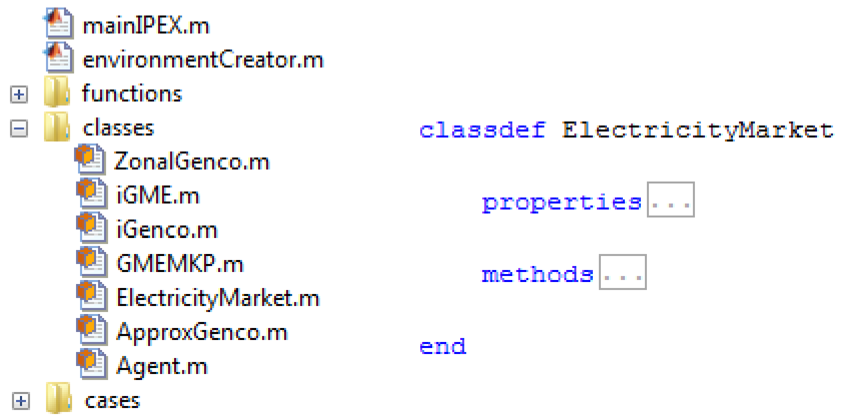
2. *ZonalGenco*: companies are devided by zones, therefore Agents behave like Gencos, choosing a different strategy for each power plants. This scenario has been implemented for the sake of completeness (since it was implemented by Guerci), but it will be not explored in this thesis.

By doing so, adding a new scenario is simple: it is sufficient define a new class that implements the *iGenco* interface (in order to interact correctly with the environment) and define internally the new behavior.

The simulation proceeds like in the previous model: at each run, all Gencos study the current market situation, in order to choose a strategy that replies as well as possible to the competitors. They evolve their population of strategies and then they choose the best strategy found, by following Equation 1.7.

2.3 Implementation

If the new model continues to reflect the main aspects depicted in sections 1.1 and 1.2, the implementation is different with respect to the previous one, adopting completely the OO programming: all the objects described in the previous section are implemented with a Matlab class and, during the run, they are instantiated by the main script (*mainIPEX.m*).

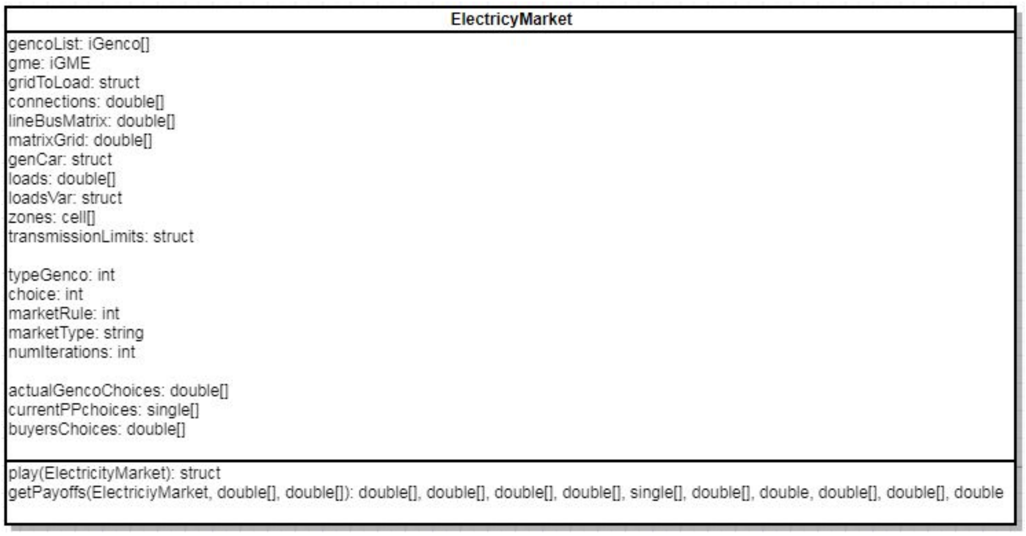


```
mainIPEX.m
environmentCreator.m
functions
classes
    ZonalGenco.m
    iGME.m
    iGenco.m
    GMEMKP.m
    ElectricityMarket.m
    ApproxGenco.m
    Agent.m
cases
```

```
classdef ElectricityMarket

    properties...

    methods...

end
```

Figure 2.2: On the left, the project's objects. On the right, the general definition of a Matlab class

Now the implementation of *ElectricityMarket* and *GMEMKP* objects is not redundant: data are not replicated and they are assigned to the right object. *ElectricityMarket* represents the Italian market itself, therefore it contains all the information about it.



```
ElectricyMarket

gencoList: iGenco[]
gme: iGME
gridToLoad: struct
connections: double[]
lineBusMatrix: double[]
matrixGrid: double[]
genCar: struct
loads: double[]
loadsVar: struct
zones: cell[]
transmissionLimits: struct

typeGenco: int
choice: int
marketRule: int
marketType: string
numIterations: int

actualGencoChoices: double[]
currentPPchoices: single[]
buyersChoices: double[]

play(ElectricityMarket): struct
getPayoffs(ElectriciyMarket, double[], double[]): double[], double[], double[], double[], single[], double[], double, double[], double[], double
```

Figure 2.3: ElectricityMarket object in UML

The first group of attributes contains all the information and characteristics of the IPEX, including the list of Gencos and the reference to the current GME instance. The second group represents settings and the third one includes attributes that collect choices during the simulation.

This object implements two functions:

- *play*: it simulates, for *numIterations* times, the electricity market session, described in the previous section.
- *getPayoffs*: it invokes the function *MGPMKP.m* (whose implementation is not changed) in order to obtain the PUN and the production quantities of each Genco (market clearing phase); then it uses these information to estimate the profits for each Genco. The implementation of this function

does not change a lot between the two projects, but before this function was located in the *@GMEMKP* class. The location is changed because the parameters of this function are provided by *ElectricityMarket* and they are not replicated in the GME class.

In the figure above, the *gme* property is an object that implements *iGME* interface. It defines only one property and one method. In order to define an interface in Matlab [3], it is necessary implements the Matlab superclass *handle* and set the method's section abstract:

```
classdef iGME < handle
    properties
        marketType
    end

    methods (Abstract)
        actionSpaceCreator(obj,gen,nrS,technology,co2Costs,
            fuelCosts,priceCap,competitiveFringe)
    end

end
```

The *GMEMKP* object inherits the property of *iGME* and it defines its own properties.

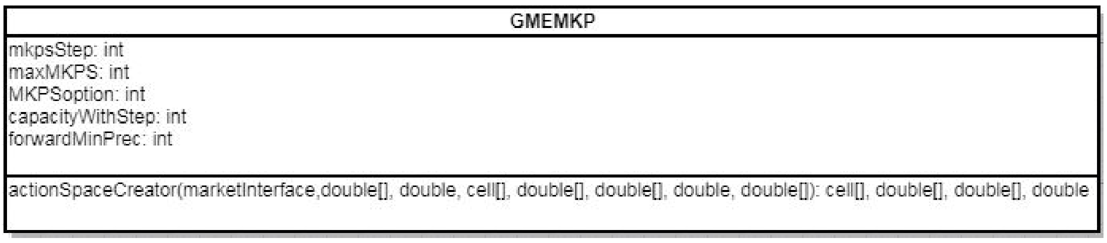

GMEMKP
mkpsStep: int
maxMKPS: int
MKPSoption: int
capacityWithStep: int
forwardMinPrec: int
actionSpaceCreator(marketInterface,double[], double, cell[], double[], double[], double, double[]): cell[], double[], double[], double

Figure 2.4: GMEMKP object in UML

The *actionSpaceCreator* function evaluates the action space for every power plant, as depicted in section 1.2.

In order to facilitate the extensibility, the project adopts a sort of pattern factory to instantiate the *gme* object:

```
function actualGME = gmeFactory(marketType, ASCsettings)
%Function that simulate the pattern factory.
    switch marketType
        case 'mkp'
            actualGME = GMEMKP(marketType, ASCsettings);
        otherwise
            display("Error: marketType not defined");
    end
end
```

The *gencoList* property of *ElectricityMarket* contains all the Gencos, which participate to the simulation. These objects must implement *iGenco* interface:

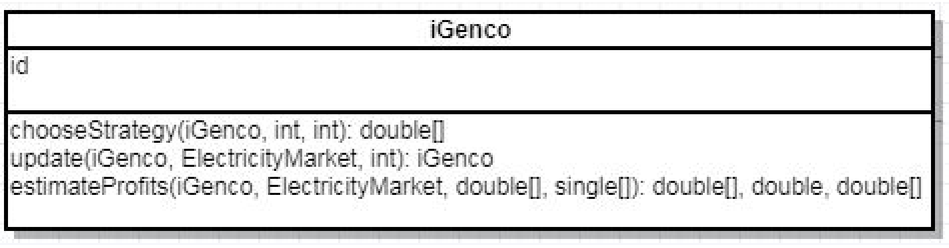

iGenco
id
chooseStrategy(iGenco, int, int): double[]
update(iGenco, ElectricityMarket, int): iGenco
estimateProfits(iGenco, ElectricityMarket, double[], single[]): double[], double, double[]

Figure 2.5: Genco object in UML

These three methods are fundamental in order to interact with the environment, since they are invoked in *ElectriciyMarket*'s functions:

- *chooseStrategy* returns the best strategy (i.e. an element of the action space) found.
- *update* applies one of the updating algorithm implemented, in order to update the population of strategies.
- *estimateProfits* evaluates the profit, after the bid to the DAM.

In order to make easily extensible the project, there is another factory to instantiate Genco objects.

In Guerci's project, Genco is implemented by *@sellerEM*, *@agent* and *@gaAgent*. These three pseudo-objects (there is not a *classdef* declaration [2]) divide the Genco's properties: the first one provides coefficient and capacity constraints data, the second one contains information about the action space and a reference to the third one, which contains the population and the update algorithm (the genetic algorithm). Therefore, in this case, defining a new object is necessary in order to implement an alternative optimization algorithm. This structure creates misunderstanding and data are redundant.

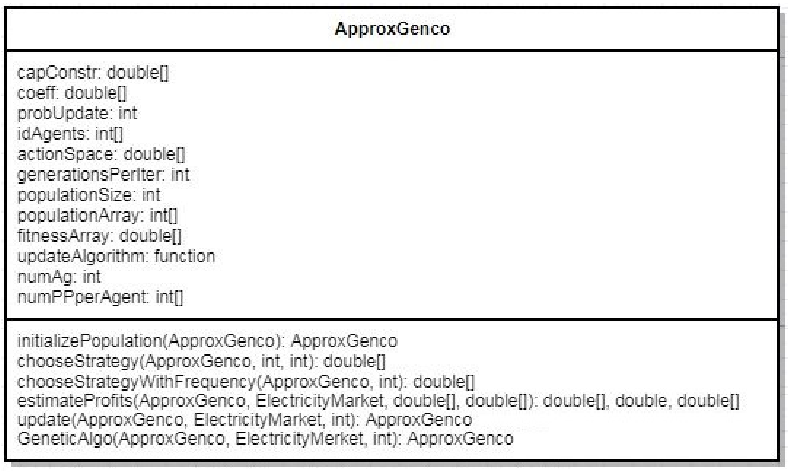

ApproxGenco
capConstr: double[]
coeff: double[]
probUpdate: int
idAgents: int[]
actionSpace: double[]
generationsPerIter: int
populationSize: int
populationArray: int[]
fitnessArray: double[]
updateAlgorithm: function
numAg: int
numPPperAgent: int[]
initializePopulation(ApproxGenco): ApproxGenco
chooseStrategy(ApproxGenco, int, int): double[]
chooseStrategyWithFrequency(ApproxGenco, int): double[]
estimateProfits(ApproxGenco, ElectricityMarket, double[], double[]): double[], double, double[]
update(ApproxGenco, ElectricityMarket, int): ApproxGenco
GeneticAlgo(ApproxGenco, ElectricityMerket, int): ApproxGenco

Figure 2.6: ApproxGenco Object UML

Now the project has two different implementations of *iGenco* and each of them has its own behavior, depicted in the previous section. Therefore, each object has its own properties and data are not split among different objects.

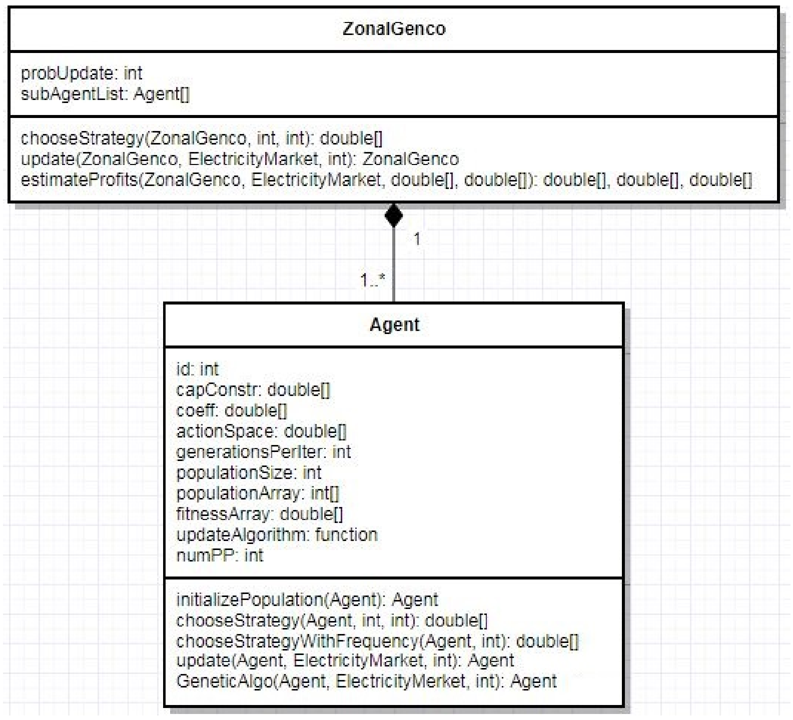

ZonalGenco
probUpdate: int
subAgentList: Agent[]
chooseStrategy(ZonalGenco, int, int): double[]
update(ZonalGenco, ElectricityMarket, int): ZonalGenco
estimateProfits(ZonalGenco, ElectricityMarket, double[], double[]): double[], double[], double[]
1
1..*
Agent
id: int
capConstr: double[]
coeff: double[]
actionSpace: double[]
generationsPerIter: int
populationSize: int
populationArray: int[]
fitnessArray: double[]
updateAlgorithm: function
numPP: int
initializePopulation(Agent): Agent
chooseStrategy(Agent, int, int): double[]
chooseStrategyWithFrequency(Agent, int): double[]
update(Agent, ElectricityMarket, int): Agent
GeneticAlgo(Agent, ElectricityMerket, int): Agent

Figure 2.7: ZonalGenco and Agent Objects UML

Each object has a population of strategy (*populationArray* property). It is initialized with random values by *initializePopulation* function and then, at each run, it evolves following a specific updating algorithm. The related fitness values are then saved in the *fitnessArray*. *chooseStrategy* function calls directly *chooseStrategyWithFrequency*, but the last parameter represents a possibility of choosing alternative approaches: this feature is designed to maintain extensible the framework. The *updateAlgorithm* property of these objects is a function handle, which references the updating algorithm (i.e. the *GeneticAlgo* function): now, in order to implement a new optimization algorithm, it is sufficient to define a new function and set it to this property in the setting phase of the main.

The fitness function, *estimateFitnessApprox1*, is the same adopted in the old version of the framework, which simulates the market-clearing phase for the specific strategy of the genome, by collecting all the previous competitors' choices. We denote this type of fitness function, since it reinforces the weight of the power plant with low prices, with the number 1.

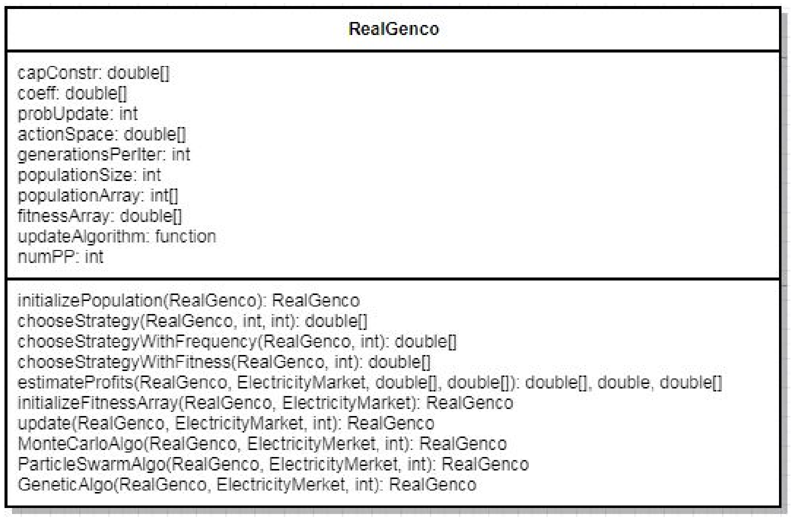
2.4 A first extension

The current framework has been extended, in order to complete some aspects which had been neglected and add new features. The main issues are:

- Remove the approximation of the Genco's strategies: add a new *iGenco* implementation class which chooses a different strategy for each power plant.
- Test the efficacy of the genetic algorithm, by applying alternative optimization algorithms.
- Add another method to choose the strategy, based on the fitness instead of the frequency probability.
- Propose and test a less constrained and more realistic fitness function.

The new implementation is called *RealGenco* and implements all the properties and methods described in the previous section. Figure 2.8 includes also the new features described above. The *chooseStrategyWithFitness* function takes the element of the population with the highest fitness; a parameter, defined in the setting phase in the main, is passed to *chooseStartegy* function in order to choose which approach apply.

Then, two optimization algorithms have been implemented and applied, as test bench for the genetic algorithm. The extension is now simple: we need only to define a new function and set


RealGenco

capConstr: double[]
coeff: double[]
probUpdate: int
actionSpace: double[]
generationsPerIter: int
populationSize: int
populationArray: int[]
fitnessArray: double[]
updateAlgorithm: function
numPP: int

initializePopulation(RealGenco): RealGenco
chooseStrategy(RealGenco, int, int): double[]
chooseStrategyWithFrequency(RealGenco, int): double[]
chooseStrategyWithFitness(RealGenco, int): double[]
estimateProfits(RealGenco, ElectricityMarket, double[], double[]): double[], double, double[]
initializeFitnessArray(RealGenco, ElectricityMarket): RealGenco
update(RealGenco, ElectricityMarket, int): RealGenco
MonteCarloAlgo(RealGenco, ElectricityMerket, int): RealGenco
ParticleSwarmAlgo(RealGenco, ElectricityMerket, int): RealGenco
GeneticAlgo(RealGenco, ElectricityMerket, int): RealGenco

Figure 2.8: RealGenco Object UML

it in the parameter of the main, then the function handle *updateAlgorithm* calls automatically the right method.

In *Monte Carlo Optimization* [4], an approximation to the optimum of an objective function is obtained by drawing random points from a probability distribution, evaluating them, and keeping the one for which the value of the objective function is the greatest (if a maximum is sought for) or the least (if a minimum is sought for). As the number of points increases, the approximation converges to the optimum. This optimization algorithm is implemented in the *MonteCarloAlgo* function and the objective function values are estimated by the function *initializeFitnessArray*.

The *Particle Swarm Optimization* algorithm [5, 6], implemented in *ParticleSwarmAlgo* function, is a meta-heuristic method inspired by the behavior or rules that guide the group of animals, for example bird flocks. According to these rules, the members of the swarm need to balance two opposite behaviors in order to reach the goal: individualistic behavior, in which each element searchs for an optimal solution, and social behavior, which allows the swarm to be compact. Therefore, individuals take advantage from other searches moving toward promising region. In this algorithm, the evolution of the population is re-created by the changing of the velocity of the particles. The idea is to tweak the values of a group of variables in order to make them become closer to the member of the group whose value is closest to the considered target. In Matlab, Particle Swarm (PS) [7] is similar to genetic algorithm (GA). It is also a population-based method with the particularity that the elements of the population are iteratively modified until a termination criterion is satisfied. The objective function accepted by the algorithm has the same characteristic of the one accepted by genetic algorithm, therefore it is possible to apply the fitness function depicted in section 2.3.

The last part of this first extension focuses on the fitness function. Firstly, another fitness function of type 1 has been implemented for the RealGenco object: *estimateFitnessReal1*. Then, we proposed a more realistic solution which consists in studying the impact of using a “relaxed” counterpart of the type 1 fitness function. This second fitness is computed as the amount of profit (given by Equation 1.4) a given individual allows the power plans to get. Therefore, two fitness functions of type 2 has been implemented: *estimateFitnessApprox2* and *estimateFitnessReal2*. It is possible to choose which type of fitness function adopts in the simulation with a setting in the main.

2.5 Results

The project has been implemented in MATLAB R2017a with Optimization and Global Optimization toolboxes. Experiments were performed on a computer running Windows 7 and based on a Intel©CoreTMi7-3610QM @2.30GHz with 8 GB main memory.

This new project reproduces exactly the same market mechanism proposed in the previous project, changing only the implementation of Gencos. Indeed, if at the first run all Gencos choose the first strategy of the action space, we obtain the same result, i.e. the same PUN, in the two project.

The new framework has been implemented and then validated at a macro level like the previous one. I have tested the ability of each algorithm to reproduce the daily PUN time series. Combining the two fitness functions and the two typologies of choice, we obtain 4 different scenarios for the genetic algorithm:

- GAfreq1: the genetic algorithm uses the fitness function of type 1 and the choice of the best strategy is based on frequency probability.
- GAfreq2: the genetic algorithm uses the fitness function of type 2 and the choice of the best strategy is based on frequency probability.
- GAfitness1: the genetic algorithm uses the fitness function of type 1 and the choice of the best strategy is based on the fitness.
- GAfitness2: the genetic algorithm uses the fitness function of type 2 and the choice of the best strategy is based on the fitness.

The same combinations are applied for the particle swarm algorithm, obtaining PSfreq1, PSfreq2, PSfitness1, PSfitness2. The MonteCarlo algorithm is applied only with the fitness based for selection.

The time required for the exectuion of one simulation's iteration with a population of 10 elements is about 3 seconds for the genetic algorithm and Monte Carlo algorithms and 4 seconds for the particle swarm algorithm. We executed some experiments with a bigger size of the population (for example, a population of 50 elements), but results did not improve.

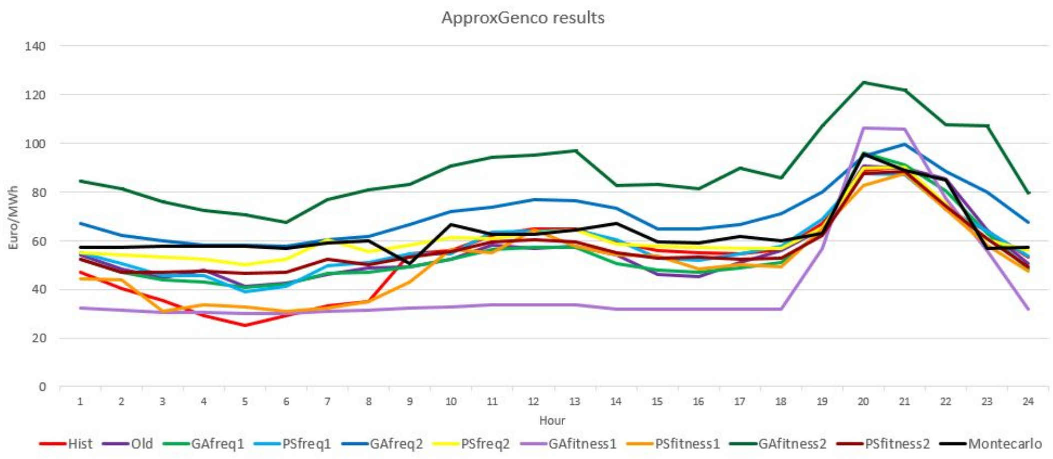


Figure 2.9: Real and simulated PUNs for the 2010-10-03, provided by ApproxGenco

Figure 2.9 shows historical values (red line), old project's results (purple line) and results of new algorithms. GAfreq2 (blue line), GAfitness2(dark-green line) and Montecarlo (black line) are the worst algorithms: all of them overestimate the PUN for all hours and the Montecarlo is not "stable", since the evaluation of the PUN could change a lot between different runs. GAfitness1 (lilac line) is wrong: it is too, low except in peak hours in which it is too high. PSfreq2 (yellow line) is high in the first part of the day (when PUN is low), but it seems good in the second part.

Focusing on the last four algorithms, it could be seen that results of old project (purple line) and GAfreq1 (green line) are very similar: this is good, because the old project implements exactly this scenario. PSfreq1 (light-blue line) is similar to the

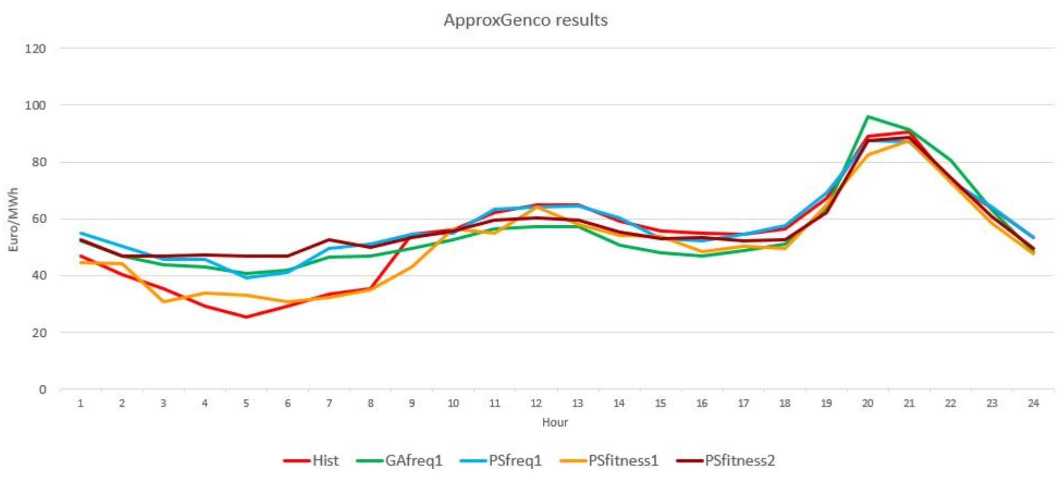


Figure 2.10: Focus on real PUNs and the four best algorithms seen in 2.9

previous ones in the off-peak hours, but it reproduces the PUN better in central hours. PSfitness2 (brown line) is good, except in off-peak hours, in which is a bit higher than other algorithms. PSfitness1(orange line) is the best algorithm: it is the only one which is able to reproduce the PUN in off-peak hours.

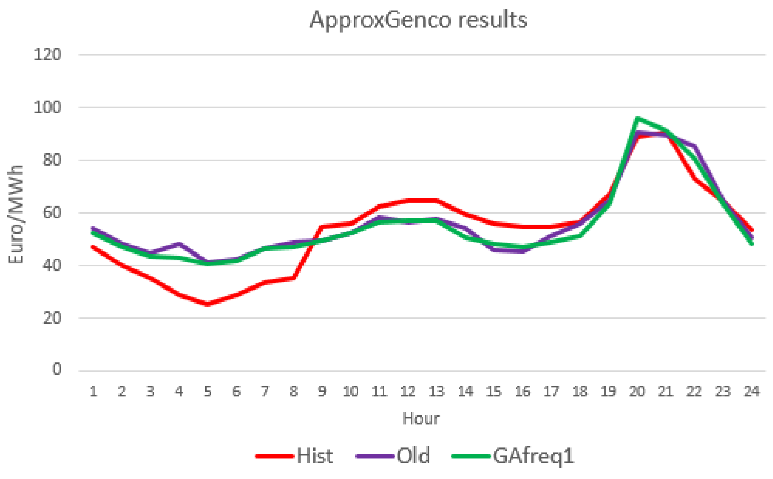


Figure 2.11: Focus on old project's results and GAfreq1

These considerations at macro-level has been supported by the evaluation of the root-mean-square deviation (RMSD): it is a frequently used measure of the difference between values predicted by a model and the values actually observed. The RMSD represents the sample standard deviation of the differences between predicted values and observed values. The formula is the following:

$$RMSD = \sqrt{\frac{\sum_{h=1}^{24}(\hat{y}_h - y_h)^2}{24}} \tag{2.1}$$

where h represents the hour of the day (therefore it varies between 1 and 24), $\hat{y}_h$ and y_h are respectively the predicted value and the observed value of the PUN at hour h. Table 2.1 shows the RMSD of the algorithms depicted above.

Table 2.1: RMSD of ApproxGenco's methods

GAfreq1	PSfreq1	GAfreq2	PSfreq2	GAfitness1	PSfitness1	GAfitness2	PSfitness2	Montecarlo
8,30068	7,73928	18,493	12,116	18,6042	5,13951	35,8702	9,29075	14,9331

Then I have tested RealGenco's algorithms, trying to obtain good results thanks to the expansion of the strategy space.

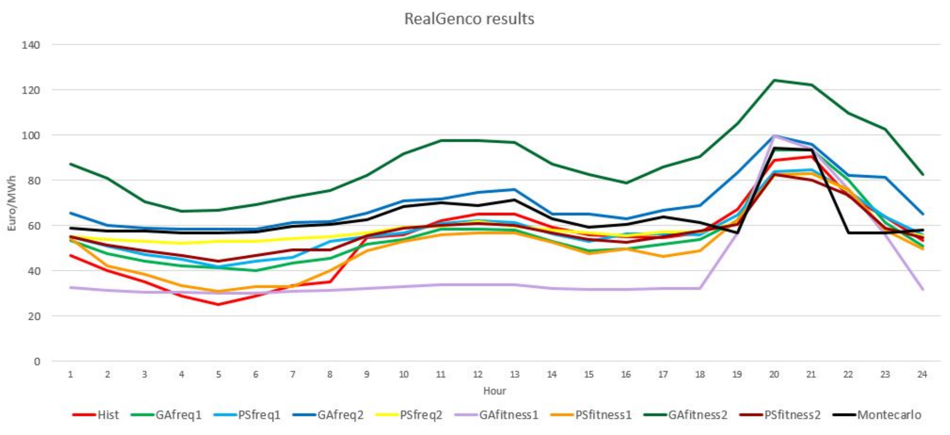


Figure 2.12: Real and simulated PUNs for the 2010-10-03, provided by RealGenco

Figure 2.12 compares the same algorithms depicted in Figure

2.9, but for RealGenco. GAfreq2 (blue line), GAfitness2 (dark-green line) and Montecarlo (black line) are still bad, because they clearly overestimate PUNs for a large part of the day. PS-freq2 (yellow line) is a bit higher in off-peak hours and a bit lower in peak hours. GAfitness1 (lilac line) is still too low. Like in the previous case, GAfreq1 (light-green line), PSfreq1 (light-blue line), PSfitness1 (orange line) and PSfitness2 (brown line) are the best algorithms. In this case, they are basically equivalent (except PSfitness1 in off-peak hours), but in peak hours they little overestimate or underestimate the historical values.

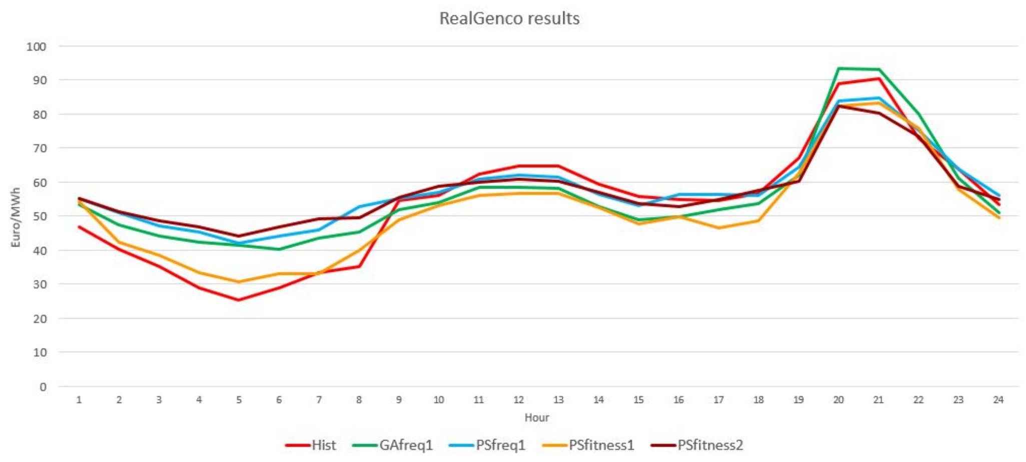


Figure 2.13: Focus on real PUNs and the four best algorithms seen in 2.12

These considerations are again supported by the evaluation of the RMSD:

Table 2.2: RMSD of RealGenco's methods

GAfreq1	PSfreq1	GAfreq2	PSfreq2	GAfitness1	PSfitness1	GAfitness2	PSfitness2	Montecarlo
7,41478	8,4084	17,8398	12,0772	18,1189	5,84692	34,9365	9,42481	15,5475

The analysis of these different algorithms has revealed an important feature: the Particle Swarm optimization algorithm seems more robust to changes. Taking into account the four possibilities, Particle Swarm has 3 out of 4 algorithms very good

(PSfreq1, PSfitness1 and PSfitness2) and the last one (PSfreq2) works well but it is higher in off-peak hours. On the contrary, Genetic algorithms works well only once (GAfreq1) and other algorithms are too much low (GAfitness1) or too much high (GAfreq2 and GAfitness2).

The expansion of the strategy space, adopted with RealGenco,

ApproxGenco				RealGenco			
GAfreq1	PSfreq1	PSfitness1	PSfitness2	GAfreq1	PSfreq1	PSfitness1	PSfitness2
8,30068	7,73928	5,13951	9,29057	7,41478	8,4084	5,84692	9,42481

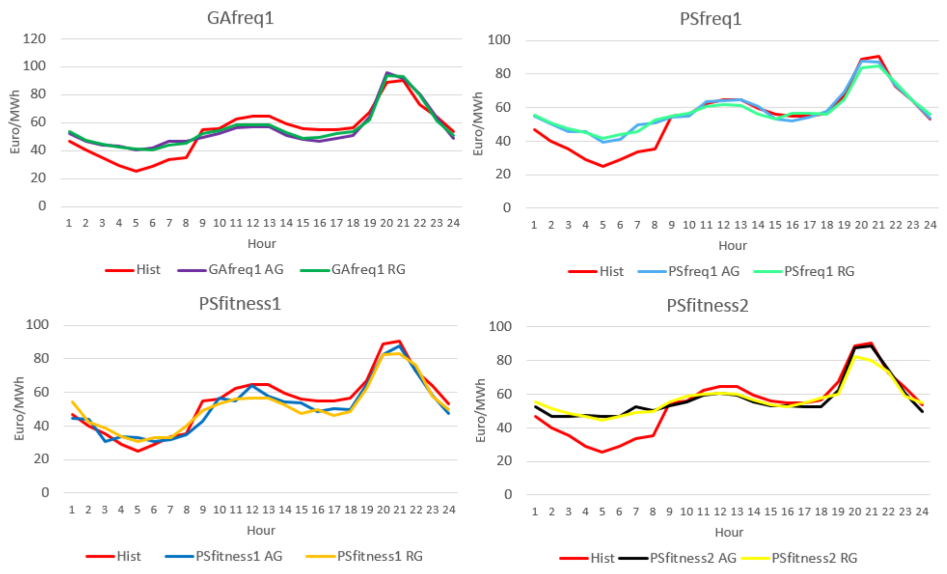


Figure 2.14: On the top, the RMSD of the best algorithms for ApproxGenco and RealGenco. On the bottom,the graphical representations of the best algorithms for ApproxGenco (AG) and RealGenco (RG), divided by functions.

does not improve the results: PUN's series are similar in both cases, with differences of some euros, and the RMSDs are similar, as it could be seen in Figure 2.14.

Figure 2.15 represents the absolute errors between the methods depicted in figures 2.14, divided by ApproxGenco and RealGenco. The absolute error represents the distance between the two curves ad each point:

$$AE_i = |pr_i - hist_i| \tag{2.2}$$

where pr_i is the prediction at hour i and $hist_i$ is the historical value of the PUN at the same hour. In this figure, the clear over-estimation of the PUN in off-peak hours is easy distinguished in the first hours of the day, since the absolute error is very high for all the methods, except for PSfitness1.

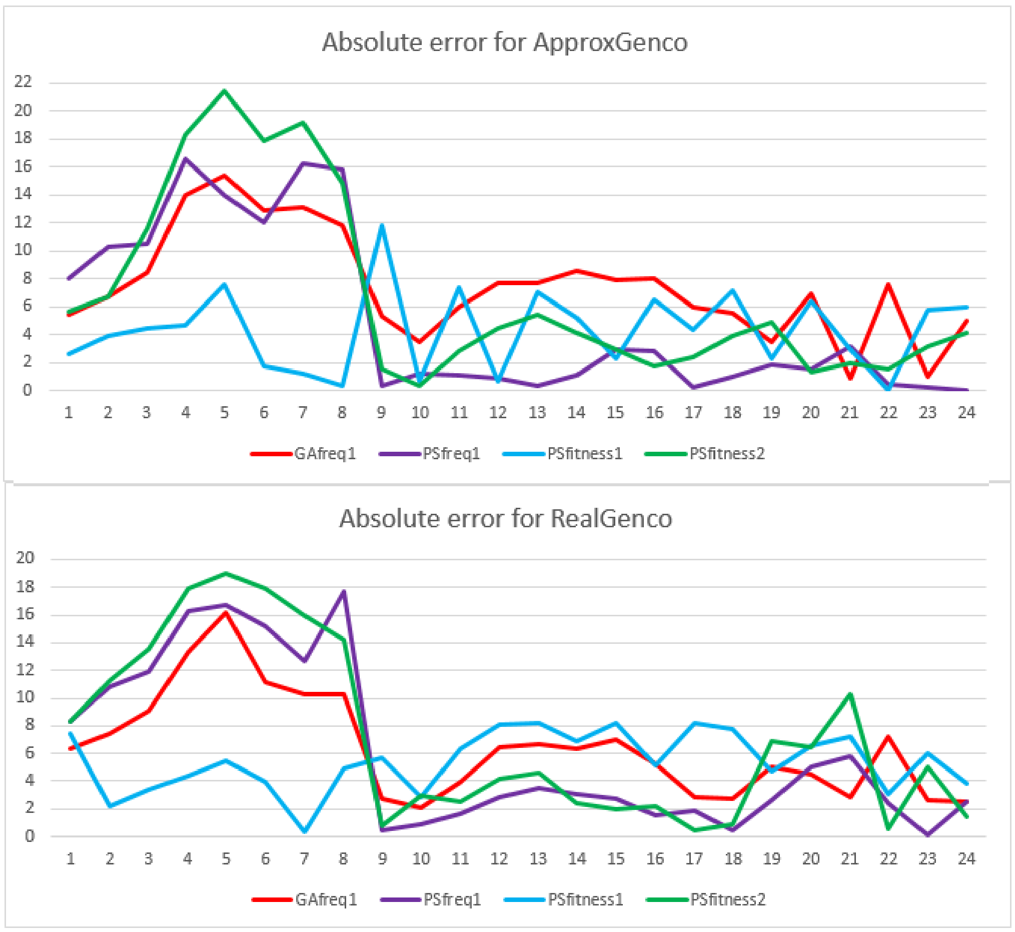


Figure 2.15: The absolute error of the best algorithms (GAfreq1, PSfreq1, PSfitness1 and PSfitness2) for ApproxGenco, on the top, and RealGenco, on the bottom.

Chapter 3

More intelligent Gencos

In this chapter, the framework will be expanded with a new version of Gencos. The goal is to develop more intelligent Gencos.

3.1 The proposal

In the current framework, discussed in the previous chapters, there is an updating algorithm (genetic algorithm) for each Genco. A population of strategies evolves in order to find the optimal strategy for the associated Genco, i.e. the strategy which allows the Genco to make the better profit.

The populations evolve separately. The profit of a Genco depends on the strategies of other Gencos: in the current version of the framework, each Genco considers, at time t, the strategies adopted by competitors at time $t-1$, not the ones it expects the competitors will adopt at time t. In other words, the expectation of a Genco is that all the competitors will repeat the strategy they used at the previous auction.

Therefore, the idea is to make the Genco more intelligent: the Genco should consider, during the period in which it contructs its strategy, all the possible strategies the other Gencos might adopt in the future. This way, the strategy under construction can be optimized against the most unfavorable competitors' strategies.

This approach may be seen as a form of adversarial reasoning, but it is a bit different from the game reasoning. In a game, the goal is to win against the opponent, but here there are not winners and losers in the same sense: the goal for a Genco is to obtain high profits as much as possible for its own characteristics and possibilities, respecting all the constraints imposed by the market. Therefore, under this point of view, it is possible that a Genco's best strategy allows to a competitor to obtain a higher profit.

3.2 Methodology

Instead of having G (where G is the number of Genco) updating algorithms evolving with one different population for each Genco, the proposal is to evolve G updating algorithms with two population for each Genco: the first one concerning the Genco's strategy itself and the second one concerning the possible strategies of all other Gencos. Therefore, the individuals of the second population represents the strategies of the remaining $G-1$ Gencos.

Taking into account two populations, we have two benefits: the adversarial reasoning, as told before, and the independency between Gencos. In the current framework, Gencos must share their own strategy in order to allow the evolution of the population. This practice is not realistic: in the real market, companies do not share their strategies with competitors. Therefore, by adding a second population, Gencos can avoid sharing these precious information and they can reason by themselves, like in the real world.

In the following, I will consider the hypotesis of the Approx-Genco, in which the Genco manages power plant of the same subAgent with a single strategy. The same reasoning is valid for the other case.

Regarding Figure 3.1, let x_i be one individual of population 1

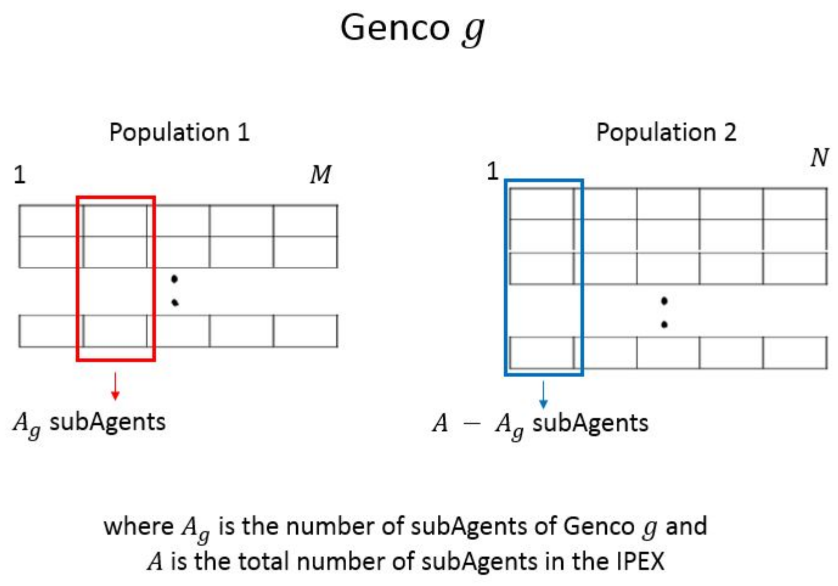

Genco g
Population 1
1
M
A_g subAgents
Population 2
1
N
$A - A_g$ subAgents
where A_g is the number of subAgents of Genco g and
A is the total number of subAgents in the IPEX

Figure 3.1: Schematic representation of the two populations of a Genco

(red rectangle) and let y_j be one individual of population 2 (blue rectangle). Computing the fitness for individuals in population 1, the fitness of x_i corresponds to the average of profit that the Genco would obtain if it adopts the strategy x_i by considering all the possible strategies of the competitors:

$$f(x_i) = \frac{1}{N} \cdot \sum_{j=1}^{N} profit(x_i, y_j) \tag{3.1}$$

where N in the size of population 2 and $profit$ is the function that estimates the profit of Genco g, given all generating units bids. The objective is to maximize the fitness $f(x_i)$ for each x_i in the population 1. Other combinations can be applied for different variations of the computing of the fitness. For example, the fitness of x_i can be estimated taking into account the minimum fitness value, obtained by adopting x_i and considering all the possible strategies of the competitors

$$f(x_i) = min_{j=1..N}(profit(x_i, y_j)) \tag{3.2}$$

These cases will increase considerably the execution time: instead of solving the linear programming problem M times, now it is solved NM times. In order to reduce the execution time, an idea is to consider only the best strategy for competitors (therefore, the worst for the Genco g itself):

$$f(x_i) = profit(x_i, y_{best}) \tag{3.3}$$

where y_{best} is the element of the population 2 with the lowest fitness or the element with the highest frequency. In this case, the linear programming problem is solved M times.

The goal is to find the most robust strategy for the Genco, i.e. the strategy which better replies to competitors even if they do all theirs best to minimize the Genco's profit. Therefore, computing the fitness for individuals in population 2, several possibilities can be considered here as well:

- Compute the fitness as an average:

$$f(y_j) = \frac{1}{M} \cdot \sum_{i=1}^{M} profit(x_i, y_j) \tag{3.4}$$

  where M is the size of population 1.

- Take the strategy with the maximum profit among all the possibilities:

$$f(y_j) = max_{i=1..M}(profit(x_i, y_j)) \tag{3.5}$$

- Estimate the fitness considering only the best strategy of population 1, in order to reduce the computational effort of the two previous case (from NM to N):

  $$f(y_j) = profit(x_{best}, y_j) \tag{3.6}$$

  where x_{best} is the element of population 1 with the highest fitness value of with the highest frequency.

Therefore, many scenarios can be explored by mixing different approaches of computing the fitness from the two populations. Since that, the computational effort varies: in the best case (by applying equations 3.3 and 3.6) the linear programming problem is solved $N + M$ times and in the worst case it is solved $2NM$ times per iteration. Taking into account the generations of the genetic algorithm n_{gen}, the computational efforts become respectively $n_{gen} \cdot (N + M)$ and $n_{gen} \cdot (2NM)$ per iteration for each Genco.

In general, these approaches are strictly related to the min-max concept: in Artificial Intelligence a game is characterized by two players, whose decisions are complementary, since one tries to maximize the score and the other tries to minimize it.

3.3 Expanding the framework

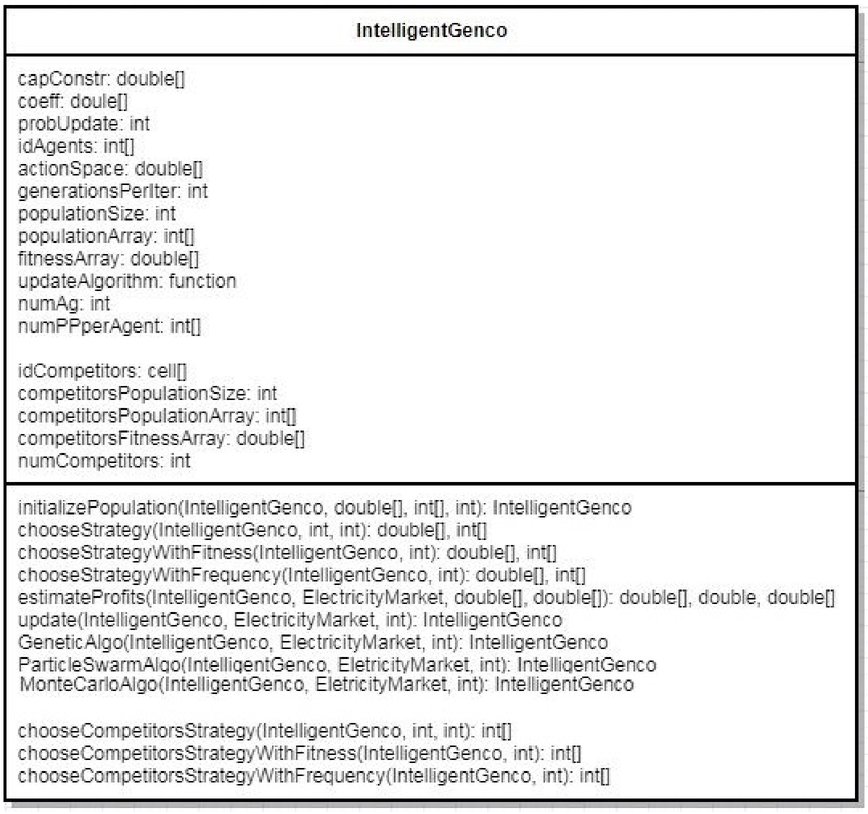
The framework has been expanded by adding a new implementation of *iGenco*: the *InelligentGenco* object. Since that, a new case has been simply added to the *gencoFactory*, with the definition of a new function *createIntelligentGenco*.

The new implementation requires also a new property in the *ElectricityMarket* object: the *generalActionSpace*, which is used by *IntelligentGenco* objects in order to know competitors strategy space.

```
em.generalActionSpace = cell(1, size(em.gencoList, 2));
for i = 1:size(em.gencoList, 2)
    em.generalActionSpace{1,i}=em.gencoList(i).actionSpace;
end
```

Figure 3.2 shows the main properties and methods of the object. The first sections contains respectively properties and functions which come from the previous ApproxGenco implementation (see section 2.3). The only change is the output values of chooseStrategy functions, which return both the indexes of the strategies in the action space and the strategies themselves.

The second sections depict properties and methods related to

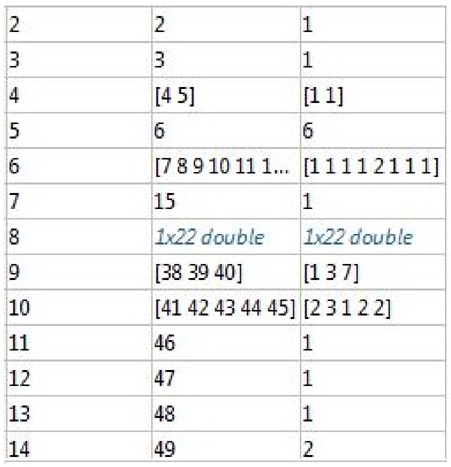


```
IntelligentGenco

capConstr: double[]
coeff: doule[]
probUpdate: int
idAgents: int[]
actionSpace: double[]
generationsPerIter: int
populationSize: int
populationArray: int[]
fitnessArray: double[]
updateAlgorithm: function
numAg: int
numPPperAgent: int[]

idCompetitors: cell[]
competitorsPopulationSize: int
competitorsPopulationArray: int[]
competitorsFitnessArray: double[]
numCompetitors: int

initializePopulation(IntelligentGenco, double[], int[], int): IntelligentGenco
chooseStrategy(IntelligentGenco, int, int): double[], int[]
chooseStrategyWithFitness(IntelligentGenco, int): double[], int[]
chooseStrategyWithFrequency(IntelligentGenco, int): double[], int[]
estimateProfits(IntelligentGenco, ElectricityMarket, double[], double[]): double[], double, double[]
update(IntelligentGenco, ElectricityMarket, int): IntelligentGenco
GeneticAlgo(IntelligentGenco, ElectricityMarket, int): IntelligentGenco
ParticleSwarmAlgo(IntelligentGenco, EletricityMarket, int): IntelligentGenco
MonteCarloAlgo(IntelligentGenco, EletricityMarket, int): IntelligentGenco

chooseCompetitorsStrategy(IntelligentGenco, int, int): int[]
chooseCompetitorsStrategyWithFitness(IntelligentGenco, int): int[]
chooseCompetitorsStrategyWithFrequency(IntelligentGenco, int): int[]
```

Figure 3.2: IntelligentGenco object in UML

the second population:

- *idCompetitors* is a cell matrix which provide a series of information. In the first column there are the ids of competitors, the second one contains an array of the subAgents ids of the corresponding genco and the last one contains the *numPPperAgent* information.



2	2	1
3	3	1
4	[4 5]	[1 1]
5	6	6
6	[7 8 9 10 11 1...	[1 1 1 1 2 1 1 1]
7	15	1
8	1x22 double	1x22 double
9	[38 39 40]	[1 3 7]
10	[41 42 43 44 45]	[2 3 1 2 2]
11	46	1
12	47	1
13	48	1
14	49	2

Figure 3.3: *idCompetitors* matrix of the first IntelligentGenco

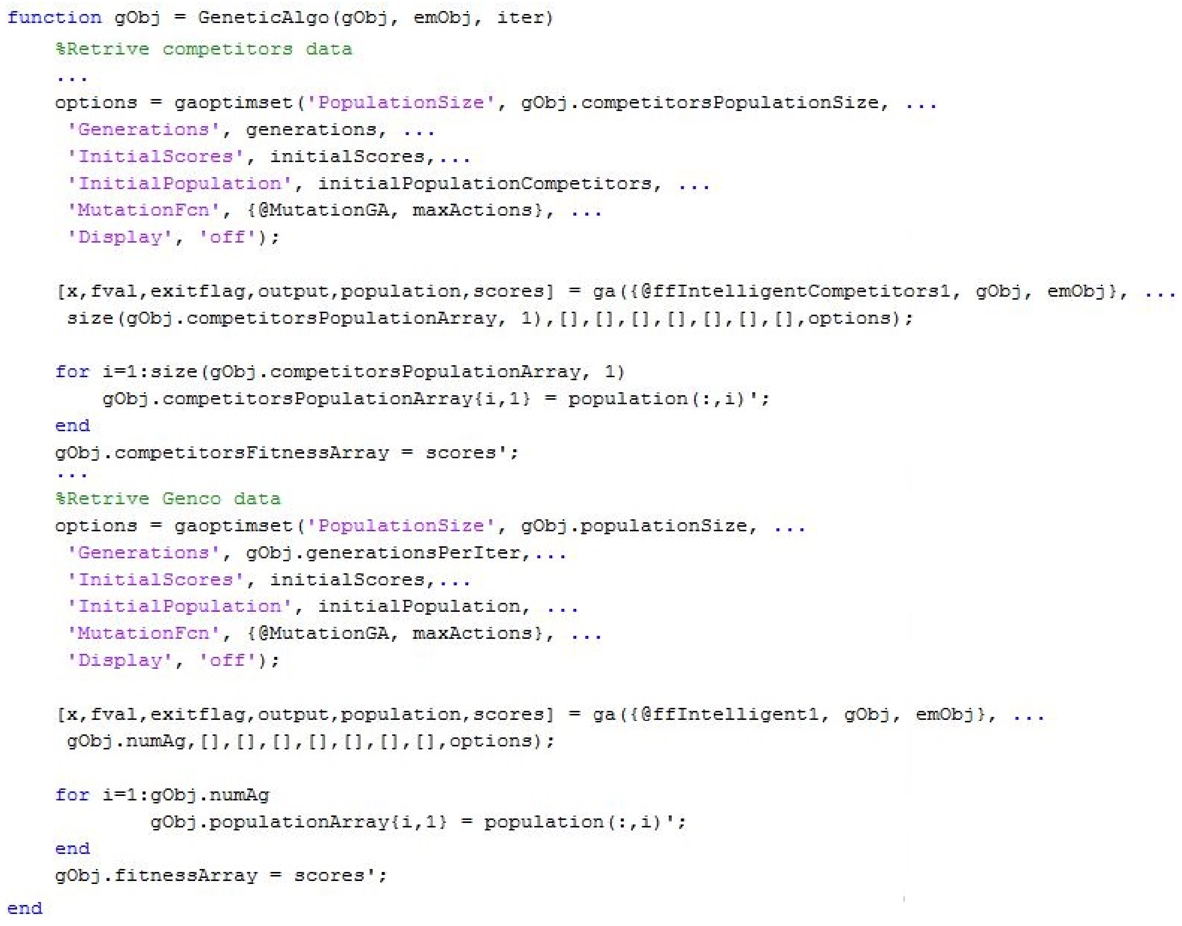
These data are used in order to correctly access to the *generalActionSpace* matrix.

- *competitorsPopulationSize* denotes the number of strategies of the second population, which could be different from the first one.

- *competitorsPopulationArray* and *competitorsFitnessArray* represent respectively the second population and the corresponding fitness values array.

- *chooseCompetitorsStrategy* function calls one of the following functions on the basis of the settings.

- *chooseCompetitorsStrategyWithFitness* function is similar to the one dedicated to the Genco itself, but it returns the competitors strategy with the lowest fitness value instead of the highest one (see Equation 3.3).
- *chooseCompetitorsStrategyWithFrequency* function returns the competitors strategy with the highest frequency, like *chooseStrategyWithFrequency*.



```
function gObj = GeneticAlgo(gObj, emObj, iter)
    %Retrive competitors data
    ...
    options = gaoptimset('PopulationSize', gObj.competitorsPopulationSize, ...
     'Generations', generations, ...
     'InitialScores', initialScores,...
     'InitialPopulation', initialPopulationCompetitors, ...
     'MutationFcn', {@MutationGA, maxActions}, ...
     'Display', 'off');

    [x,fval,exitflag,output,population,scores] = ga({@ffIntelligentCompetitors1, gObj, emObj}, ...
     size(gObj.competitorsPopulationArray, 1),[],[],[],[],[],[],[],options);

    for i=1:size(gObj.competitorsPopulationArray, 1)
        gObj.competitorsPopulationArray{i,1} = population(:,i)';
    end
    gObj.competitorsFitnessArray = scores';
    ...
    %Retrive Genco data
    options = gaoptimset('PopulationSize', gObj.populationSize, ...
     'Generations', gObj.generationsPerIter,...
     'InitialScores', initialScores,...
     'InitialPopulation', initialPopulation, ...
     'MutationFcn', {@MutationGA, maxActions}, ...
     'Display', 'off');

    [x,fval,exitflag,output,population,scores] = ga({@ffIntelligent1, gObj, emObj}, ...
     gObj.numAg,[],[],[],[],[],[],[],options);

    for i=1:gObj.numAg
            gObj.populationArray{i,1} = population(:,i)';
    end
    gObj.fitnessArray = scores';
end
```

Figure 3.4: *GeneticAlgo* function of *ItnelligentGenco* objects

The structure of the updating algorithm is obviously changed, since it has to manage two population. In the new object, the function GeneticAlgo (but also ParticleSwarmAlgo function) invokes the Matlab genetic algorithm two times with different fitness functions.

Figure 3.4 shows the two important sections of this function. In the first part, the Matlab *ga* (genetic algorithm) function is invoked with the competitors information and the competitors fitness function (*ffIntelligentCompetitors1*). In the second part, *ga* is invoked with the information of the current genco and the genco fitness function (*ffIntelligent1*). In this way both populations are alternatively updated.

Figure 3.4 shows only the first version of fitness functions, but there are 3 versions of both of them (genco and competitors), as discussed in the previous section:

- *ffIntelligentCompetitors1* implements fitness function of Equation 3.6. It constructs the array of power plant choices (used to estimate the profit, i.e. the fitness), taking the genome for competitors' power plants and the best genco's strategy:

  ```
  if isnan(seller.fitnessArray)
      gStrat=chooseStrategy(seller,1,emObj.choice);
  else
      %in this case, the second parameter can be any
          number except 1
      gStrat=chooseStrategy(seller,8,emObj.choice);
  end
  PPchoices(idxGenco, :) = gStrat;
  ```

  where the second parameter is the iteration number and the third is the choice criterion (frequency or fitness).

- *ffIntelligent1* implements the Equation 3.3, by calling the following function:

  ```
  idxCompetitorsStrat = chooseCompetitorsStrategy(seller,
   6, emObj.choice);
  ```

where the output value is an array which contains the indexes to access to the *generalActionSpace* in order to collect competitors choices.

- *ffIntelligentCompetitors2* implements Equation 3.5. In this function, a loop is needed in order to explore, for the same genome, all the elements of the first population:

```
maxFitness = −1000000000;
for j = 1:seller.populationSize
    ...
end
fitnessValue = maxFitness;
```

- *ffIntelligent2* implements Equation 3.2 and it is specular with respect to the previous one. Therefore, by iterating on `seller.competitorsPopulationSize`, it takes only the `minFitness`.

- *ffIntelligentCompetitors3* implements Equation 3.4. It works as the second version (i.e. there is a loop) but with the average:

```
sumFitness = 0;
for j = 1:seller.populationSize
    ...
   sumFitness = sumFitness + tmpFitnessValue;
end
fitnessValue = sumFitness/seller.populationSize;
```

- *ffIntelligent3* implements Equation 3.1 and it is the dual of the previous one, i.e. it iterates on `seller.competitorsPopulationSize`.

All of these fitness functions can adopt a behavior of type 1 or type 2 (see section 2.4), by setting the parameter in the main.

3.4 Results

This new extension has been executed and tested combining respectively the two fitness functions: in version 1-1 *ffIntelligent1* and *ffIntelligentCompeitors1* are applied, in version 2-2 we use *ffIntelligent2* and *ffIntelligentCompetitors2*. Version 3-3 (with *ffIntelligent3* and *ffIntelligentCompeitors3*) has not been executed, since it requires too much time, but it should be tested in a future work.

The experiments have been executed with the same computer's configurations depicted in section 2.5. The execution time varies a lot between different versions, since the execution effort varies. With two population of 10 elements, the version 1-1 takes about 16 seconds for the GA per iteration, 20 seconds for the PS and 6 seconds per MonteVarlo. The version 2-2 requires about 180 seconds per iteration for the GA, 240 second for the PS and 55 seconds for MonteCarlo.

Figure 3.5 shows the results of genetic algorithm, particle swarm optimization and MonteCarlo optimization of IntelligentGenco in version 1-1. Results are basically similar to the previous version (see section 2.5). GAfitness1 (lilac line) is still low, GAfreq2 (dark-blue line) and GAfitness2 (dark-green line) still overestimate the PUN (historical red line). PSfreq1 (light-blue line), PSfreq2 (yellow line) and PSfitness2 (brown line) are quite good (see also figures 3.6 and 3.7), except for the overestimation in the off-peak hours. GAfreq1 (light-green line) is equivalent to the previous ones, but it overestimates the PUN also in peak hours. PSfitness1 (orange line) is again the best algorithm, since it is able to reproduce the PUN in off-peak hours. The main difference is in MonteCarlo (dark dashed line with crosses), which has basically the same trend of GAfitness1 in this version.

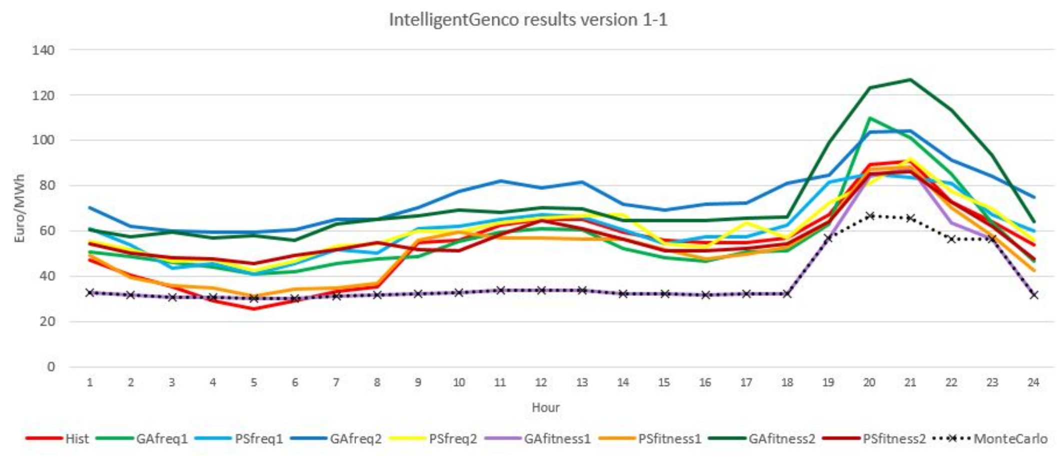


Figure 3.5: Real and simulated PUNs for the 2010-10-03, provided by IntelligentGenco in version 1-1

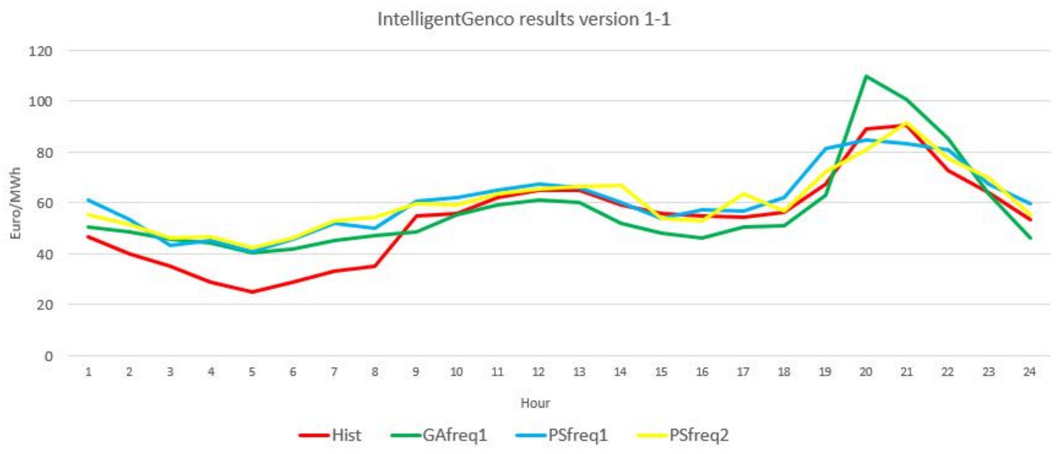


Figure 3.6: Focus on real PUNs and the best algorithms, based on frequency choice, seen in 3.5

The RMSDs (showed in table 3.1) of GAfreq1, PSfreq1, PSfreq2 and PSfitness2 are between 9 and 10; in some cases they increase of 1-2 units from the previous version of the project and in other cases they decrease. GAfitness1 is not changed, GAfreq2 is a little higher and GAfitness2 decrease a lot (from

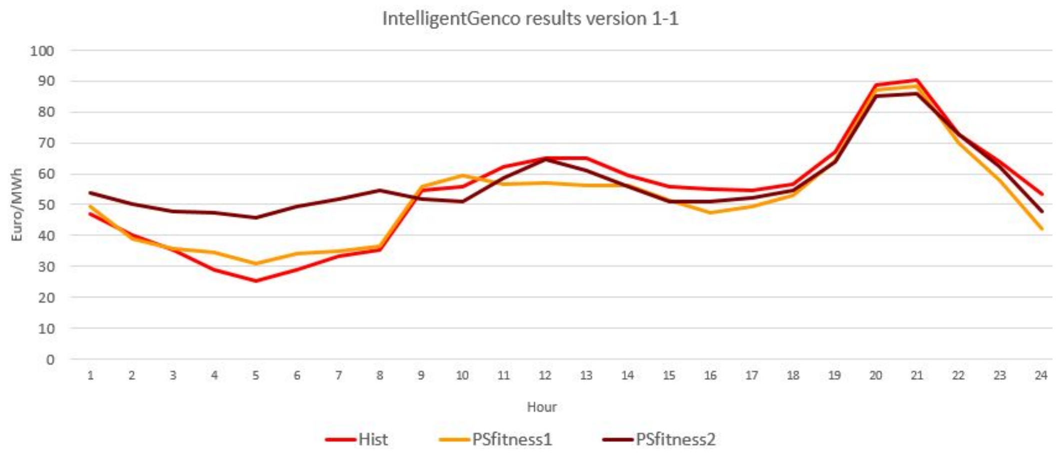


Figure 3.7: Focus on real PUNs and the best algorithms, based on fitness choice, seen in 3.5

34,8702 to 22,6235). The RMSD of PSfitness1 is the lowest, decreasing under 5.

Table 3.1: RMSD of IntelligentGenco methods in version 1-1

GAfreq1	PSfreq1	GAfreq2	PSfreq2	GAfitness1	PSfitness1	GAfitness2	PSfitness2	MonteCarlo
9,5816	9,8726	21,8860	9,8265	18,10526	4,9483	22,6235	9,965892	19,51315

Figure 3.8 shows the results for the IntelligentGenco's version 2-2. In this version results are very different with respect to the previous cases: now no line clearly overestimates the historical values (red line), except in off-peak hours. This result could be a consequence of the adversarial behavior of Gencos. In Figure 3.8, algorithms' lines are very close each other, especially in central hours. Thus, the following detailed figures will focalize on similar lines.

GAfitness1 (lilac line) is still low. Both PSfitness1 (orange line) and PSfitness2 (brown line) are low in off-peak hours, almost reaching the historical line (see figure 3.9).

The remaining genetic algorithms GAfreq1 (light-green line),

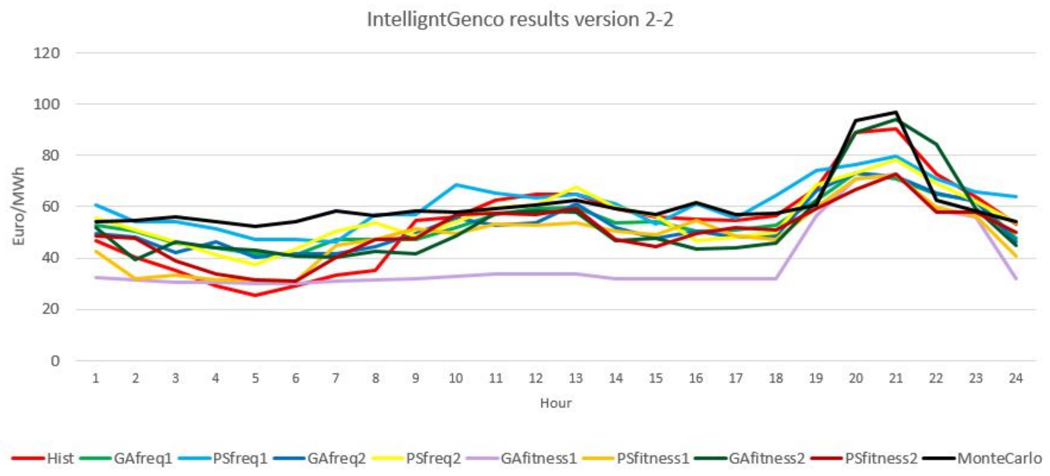


Figure 3.8: Real and simulated PUNs for the 2010-10-03, provided by IntelligentGenco in version 2-2

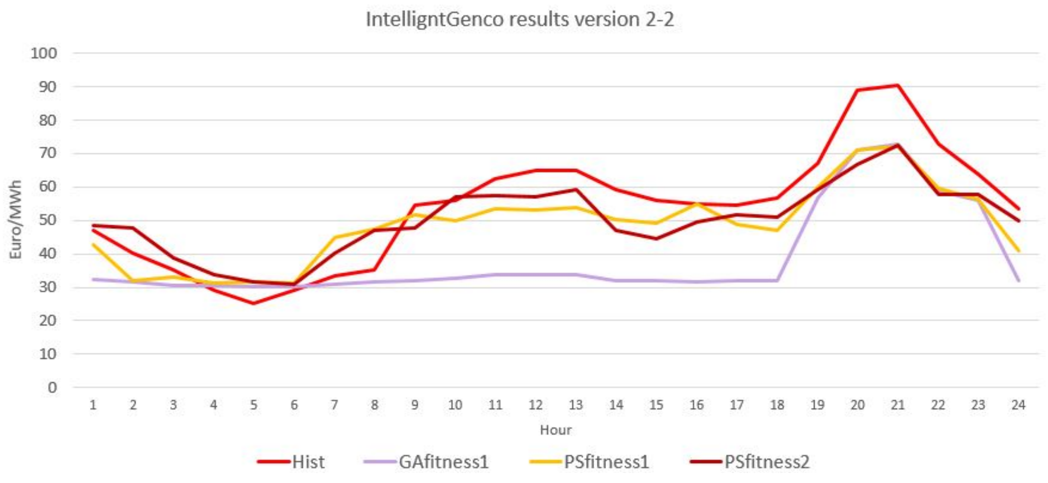


Figure 3.9: Focus on real PUNs and the lowest algorithms seen in 3.8

GAfreq2 (dark-blue line) and GAfitness2 (dark-green line) are very close each other, but only the last one is able to reach the peaks at 8pm and 9pm (as it could be seen in Figure 3.10).

The last three algorithms are very close to the historical line in the central hours, but they overestimate the off-peak hours (see figure 3.11). Both PSfreq1 (light-blue line) and PSfreq2 (yellow line) underestimate the peak hours, on the contrary MonteCarlo

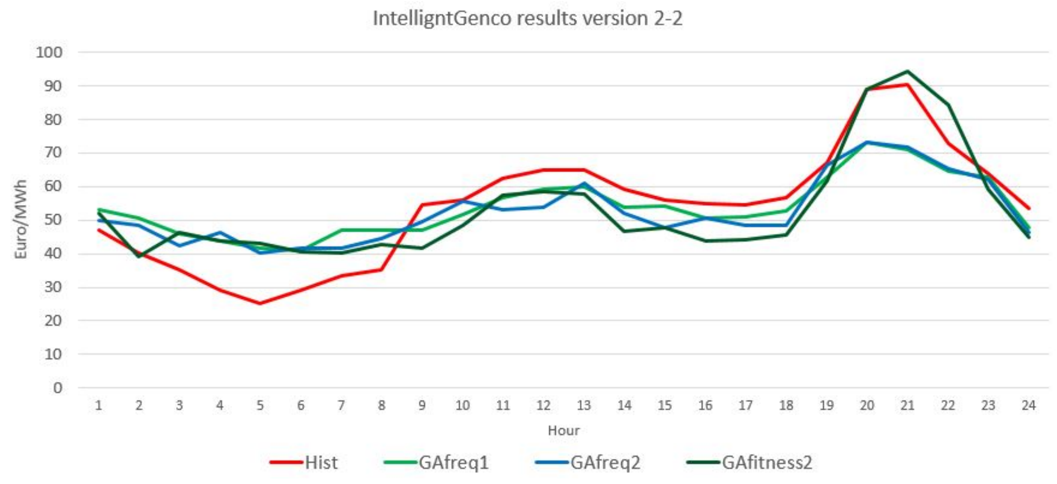


Figure 3.10: Focus on real PUNs and genetic algorithms seen in 3.8

(black line) is good at these hours.

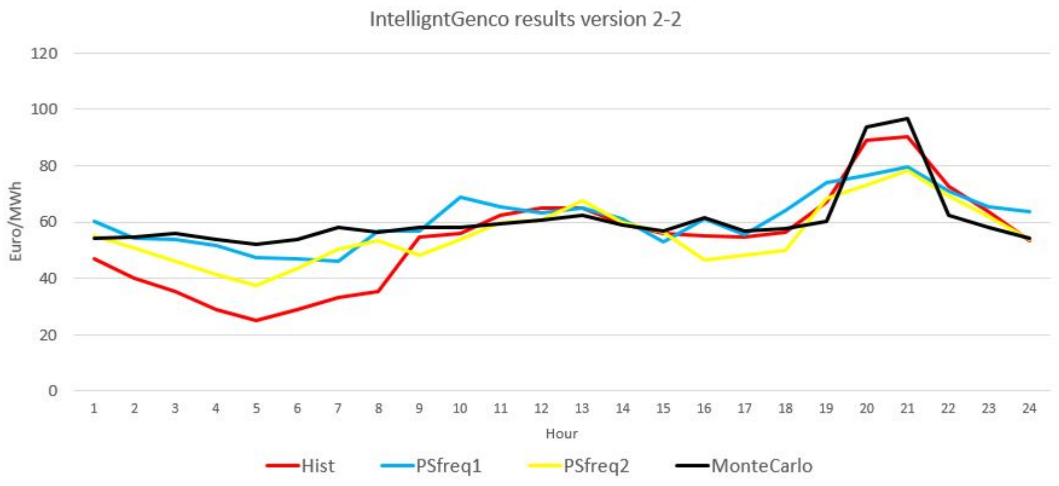


Figure 3.11: Focus on real PUNs and some algorithms seen in 3.8

The RMSDs prove the similarity between the algorithms of this version, as it could be seen in the following table: in most cases, the RMSD is between 9 and 10, only except three algorithms.

Table 3.2: RMSD of IntelligentGenco methods in version 2-2

GAfreq1	PSfreq1	GAfreq2	PSfreq2	GAfitness1	PSfitness1	GAfitness2	PSfitness2	MonteCarlo
9,6079	11,5955	9,5128	9,3197	18,9050	9,5064	9,4104	9,1137	13,0449

Taking into account the RMSDs of the version 1-1, we can see the following main differences:

- GAfreq2 and GAfitness2 have a good improvement by decreasing the RMSD from more than 20 to 9;
- MonteCarlo is now 13,0449, instead of 19,51315;
- PSfitness1 has the biggest increasing, by varying from less than 5 to 9,5064.

The other algorithms have just slight variations.

Chapter 4

Related Works

This chapter presents other agent-based models proposed in literature, in order to compare them with our framework, analyzing advantages and limitations.

4.1 The MABS framework

The first related work we analyze is the Multi-Agent Based Simulation (MABS): it is a general framework for the implementation of agent-based models, which has been applied for an Electricity Market proposed by Trigo and Coelho [8].

The paper describes the conceptual elements of the MABS: the *environment entity*, which owns a distinct existence in the real environment (e.g. a resource or the physical embody of an agent), and the *environmental property*, which is a little aspect of the real environment (e.g. the price of a bid or the electricity demand). Environment entities act and communicate each other basing their knowledge and decisions on the environmental properties.

They followed the MABS modeling proposal to describe the electric market simulation. The entities considered are the *GenCo*s, their *GenUnit*s and the *Pool*, which represents the institutional power entity. GenCos act by submitting a price and a production quantity for each GenUnit to the Pool. The approach of submittig the pair price-quantity is commonly called "*block bids*" approach. The Pool applies the same predefined auction rules in order to determine the market price and hence the block bids that clear the market. The agent decision process is a Q-learning algorithm with an ϵ-greedy exploration strategy, which picks a random action with probability ϵ and behaves greedily otherwise (i.e. it picks the action with the highes estimated action value).

4.2 The ITEM-game simulator

Trigo, de Sousa and Marquez proposed in their paper [9] a framework called ITEM-game ("Investment and Trading in Electricity Markets") where human and virtual agents can explore the investment and trading strategies for the electricity market. This simulator has been implemented in order to extend the previous TEMMAS ("The Electricity Market Multi-Agent Simulator") simulator.

Each ITEM-game player represents a generator company that pursues a profit maximization strategy. The framework has a globally view of the environment, taking into account both short-term tradings and long-term investments. The first ones are supported through a spot market, which is operated via a Pool institutional entity. GenCos submit prices and production quantities to the Pool according to the "block-bids" approach. The Pool determines the market price by clearing the market. The second ones focus on the 3-stage life cycle of the generating units: construction, operation and decommission. The goal of this research work is to study and analyze dependencies and coalitions between GenCos.

While the previous TEMMAS followed a machine (reinforcement) learning methods to autonomously search for an optimal competitive trading, the ITEM-game is an interactive tool designed for humans to explore investments and trading strategies.

4.3 The MASCEM

The MASCEM is a Multi-Agent Simulator of Competitive Electricity Market proposed by Pinto and colleagues [10]. The simulator includes some modules:

- *Automatic Data Extraction* module: this tool maintains a database updated with historical information from real electricity markets.
- *Scenarios Generator* module: this framework offers the possibility of generating scenarios for different types of electricity market, including the day-ahead market, the balancing market, the forward market, etc. Data mining techniques are applied to define the players that act in each market.
- *Strategic Behavior*: in order to allow players to automatically adapt their strategic behavior according to their current situation the platform ALBidS (Adaptive Learning Strategic Bidding System) has been integrated with MASCEM. To choose the most adequate technique to each context, ALBidS uses reinforcement learning algorithms and the Bayes theorem. ALBidS techniques includes: artificial neural networks, data mining approaches, statistical approaches, machine learning algorithms, Game theory and competitors player's action prediction.
- *Upper Ontology for Systems' Interoperability*: it allows the use of scenarios and results obtained by other simulators

and even in the real systems. This way it is easier to compare and analyze results.

4.4 The SEPIA

The SEPIA (Simulator for Electric Power Industry Agents) is the first agent-based simulator for electricity market studied in *"Agent-based simulation of electricity markets: a survey of tools"* paper [11].

All major market participants in SEPIA are modeled as agents, which include Generation Companies, Consumers, Consumer Companies and Transmission Operator, There is not an agent for the Transmission Operator (i.e. the Pool), but they implemented the OASIS (Open Access Same-Time Information System) database.

Regarding the adaptation mechanism, both a Q-Learning module with Boltzmann selection and genetic classifier learning module are designed to guide the Generation Company agents in making decisions. These components are two complete and independent modules in SEPIA.

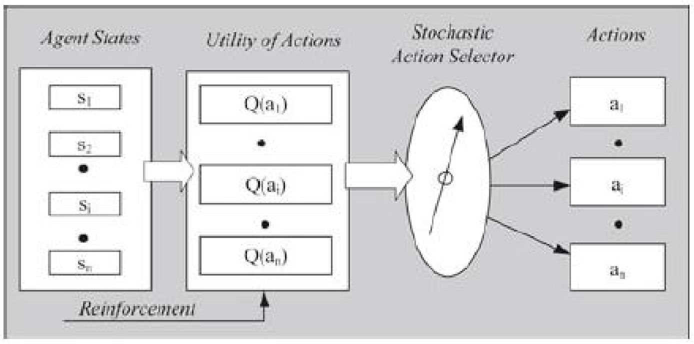


Figure 4.1: The structure of the Q-Learning module in SEPIA

The Q-Learning algorithm evaluates the reward of an action *a* as a function $Q(a)$. Then, a stochastic selector based on Boltzmann selection mechanism is used to choose a promising action.

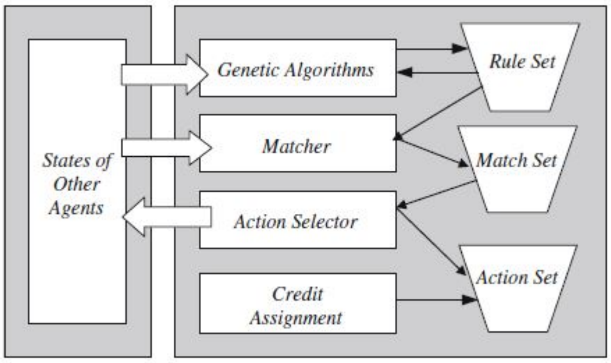


Figure 4.2: The genetic classifier learning module in SEPIA

The learning module consists of three data sets (rule set, match set and action set) and four separate and independent submodules (genetic algorithms, matcher, action selector, credit assignment). The main process is summarized as follows: (1) the classifier module contains a knowledge base represented by a set of rules; (2) each rule has a condition part that specifies an agent's current state and an action part that specifies the consequent action the agent would take; (3) the rules with certain conditions satisfied are placed into a match set by the Matcher; (4) the Action Selector uses a stochastic selector based on the Boltzmann selection mechanism to choose a rule in the Match Set and implements the selected action; (5) after the effects resulting from taking that action are cumulated and measured, a credit is assigned to the implemented rule in the action set; (6) finally, GA is used to optimize and update the rule set and the fitness of each rule is evaluated by its assigned credit.

4.5 The EMCAS

The Electricity Market Complex Adaptive System is the second framework analyzed in [11].

Market participants (agents) in EMCAS include the Independent System Operator (ISO), Gencos, Consumer and Demand companies. The market adopts bilateral contracts and bids to the pool market operated by the ISO. Each Genco is allowed to make decision on six levels, which are Hourly/Real-Time dispatch, Day-ahead planning, Week-ahead planning, Month-ahead planning, Year-ahead planning and Multi-Year-ahead planning.

The basic decision process in each agent is composed of several procedures. First, each agent must specify and comply with some decision rules depending on its roles. Whenever an agent makes a decision, it will consider the results of similar decision made previously (Look Back), those related to projection results (Look Ahead), and its current conditions (Look Sideways).

Adaptation in EMCAS could assists its agent to make decision. There are two forms of learning including observation-based learning and exploration-based learning. In observation-based learning, the decision for the next step mainly depends on the previous performance, while in exploration-based learning the agent explores various possible strategies and then slightly adjust the adopted strategy that expects to have a good performance in the near future. The adaptation process is supported by pre-specified decision rule and no adaptation (self-learning mechanism) exists for such decision rules.

4.6 The AMES

The Agent-based Modeling of Electricity Market [11] is an open-source agent-based framework, which has four main components: traders, transmission grids, markets and the ISO. The trader agent contains two types of entities: buyers (load serving entities) and sellers (generators). The market component has a two-settlement system, which consists of a day-ahead market and a real-time market.

A reinforcement learning module, called *JreLM*, is integrated into the simulation framework for adaptive decision making of traders.

4.7 Other agent-based simulators in electricity market

The paper *"Agent-based simulation of electricity markets: a survey of tools"* [11] presents other simulators. Each of them has its own market model, but generally they adopt day-ahead market or real-time market and the block-bid model. In the following, I present the adaptive learning methods of each simulator:

- STEMS-RT (Short-term Electricity Market Simulator – Real Time) does not provide an explicit adaptation or learning process in the decision making of each agent. It applies mathematical models (like Linear Programming, Mixed Integer Programming, Quadratic Programming, Linear Complementarity Programming and Mathematical Programming with Equilibrium Constraints) in the market-clearing phase.
- NEMSIM (National Electricity Market Simulation System) is developed particularly for the Australia National Electricity Market. The adaptation mechanism is based on the

look-ahead decision process. Simulations are used to test or compare various possible strategies. The agents then choose those strategies that lead to the best results.

- In [12], Veit and collagues use agent-based model to study the dynamics of a two-settlement electricity market consisting of one forward market and one spot market. A learning making decision is not implemented, but mathematical models (nonlinear programming or mixed linear complementarity problems) are applied in order to solve the market.

4.8 Similarities and differences with IPEX

There are many similarities between the IPEX framework and the other frameworks presented in this section. Even if different frameworks are based on different electricity markets (Italy, United Kingdom, Australia, USA, Portugal ...) with their own characteristics, some aspects are similar:

- In all the models, agents represent generating companies, i.e. our Genco objects. The ISO and the pool are represented in IPEX by the ElectricityMarket object and an object that implements iGME interface.
- The information set is the same for all the models: prices are based on the marginal cost of each generating unit owned by a Genco and transmission limits are based on the grid capacities.
- The “block-bid” approach, used in some simulators, is similar to our proposal: in the IPEX model, Gencos bid to the market a pair of values, the price and the limit production quantity. The effective production level required is estimated in the market-clearing phase (see section 1.2).

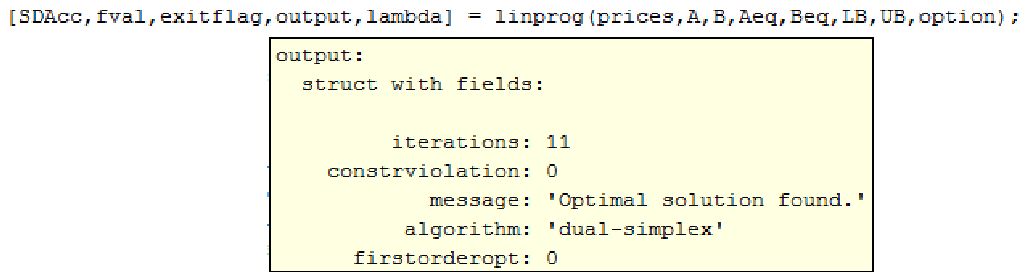
- The final price, in our case the PUN, is estimated after a market-clearing phase operated by the ISO.

The main differences with respect to the IPEX model are two. The IPEX does not include consumer/demand companies as agents, but the demand electricity levels are just considered as data. The current version of the IPEX simulates only the DAM, taking into account the energy quantities sold in the forward market: the value of the PUN is hourly evaluated and based on transmission limits and electricity demand. In the following, I will focalize the attention on the adaptive behavior and the different learning approaches adopted in IPEX and the other frameworks, since that is the main topic of this thesis.

The majority of these related works (MABS framework, TEMMAS, MASCEM, SEPIA and AMES) applies reinforcement learning, in particular the Q-learning algorithm. The reinforcement learning is based on the concepts of reward and punishment: if an action (in this case a bid) produces a high profit, the reward will be high; otherwise, if the bid generates a low profit, the punishment will be high. Therefore, a Genco will submit a bid with high profit.

The population-based algorithms (genetic algorithm or partice swarm algorithm) adopted in the IPEX model follows basically the same idea: the first population, related to the Genco strategies, evolves in order to maximize the fitness, i.e. the profit. Therefore, element of the population with high fitness will survive and elements with low profits will not be reproduced.

Like STEMS-RT and Veit's framework, which use mathematical models to find an optimal solution during the market clearing phase, the IPEX solves a linear programming problem for each strategy and each generation of the population-based algorithm, by invoking the *MGPMKP* function (see section 1.3) inside the fitness function.



```
[SDAcc,fval,exitflag,output,lambda] = linprog(prices,A,B,Aeq,Beq,LB,UB,option);
```

```
output:
  struct with fields:

         iterations: 11
    constrviolation: 0
            message: 'Optimal solution found.'
          algorithm: 'dual-simplex'
      firstorderopt: 0
```

Figure 4.3: The *linprog* function, called in the MGPMKP function, applies the dual-simplex algorithm in order to find the optimal solution

Extending the framework, we add a second population by applying adversarial reasoning and game theory like concepts. Indeed, since the second population evolves in order to minimize the profit of a Genco, the IPEX recreates a sort of min-max approach. In this way, agents do not need to interact each other in order to exchange information: in the real world, companies do not reveal strategies to competitors, but they try to predict next actions and react in consequences.

The combination of these methodologies does not seem to be explicitly applied in other works. Unfortunately, we can not compare results of this proposed methodology with results of the other frameworks depicted before: different frameworks are based on different electricity markets and different grids. Moreover, we do not have the code of the other frameworks.

Chapter 5

Conclusion and future work

This thesis describes the preliminary work in the extension of the IPEX framework proposed by Guerci[1].

The obtained results in the two incremental extensions are promising and interesting. Both in section 2.5 and in section 3.4, particle swarm optimization generally outperforms Monte Carlo optimization and the genetic algorithm; the negative side concerns the execution time, since particle swarm always requires more time than the other algorithms because of the cooperation behavior.

The expansion of the strategy space, adopted with the RealGencos, does not improve results we obtained with the ApproxGencos agents, as we wanted. Thus, we have continued to work on the second part of project taking into account only the initial approximated configuration.

Results incrementally improve in the last part of the project, thanks the introduction of the competitive behavior and adversarial reasoning of IntelligentGenco agents: in version 1-1 the RMDSs start to decrease and in version 2-2 no algorithm overestimates the historical data (except on off-peak hours). The main problem of version 2-2 is the excessive amount of time required.

This current version of the framework can be further extended in several ways:

- Implementing a new Genco agent, based on the Q-learning algorithm (reinforcement learning), since it is the most used algorithm in the related works presented in the previous chapter.
- Trying different implementation of the iGME, based for example on the load curve.
- Reducing the execution time of the current IntelligentGenco. Our idea is to add a recurrent neural network (Long Short Term Memory), which should be trained with historical values, in order to reduce the action space. By doing so, the LSTM will cut a big part of the action space and the global optimization algorithm will focalize only in a small region of that space.

Bibliography

[1] E. Guerci, S. Sapio, *Comparison and empirical validation of optimizing and agent-based models of the Italian electricity market*, 2011

[2] Matlab documentation, object oreinted programming: `https://it.mathworks.com/help/matlab/object-oriented-programming.html`

[3] Matlab documentation, define an inetface `https://it.mathworks.com/help/matlab/matlab_oop/defining-interfaces.html`

[4] B. Betrò, M. Cugiani and F.Schoen. 1990. *Monte Carlo methods in Numerical Integration and Optimization.* Giardini, Pisa.

[5] James Kennedy. 2017. Particle Swarm Optimization. In *Encyclopedia of Machine Learning and Data Mining.* Springer, 967-972.

[6] James Kennedy and Russel Eberhart. 1995. Particle Swarm Optimization. In *Proceedings of IEEE International Conference on Neural Networks, Part IV.* 1942-1948.

[7] Matlab documentation, particle swarm optimization `https://it.mathworks.com/help/gads/what-is-particle-swarm-optimization.html`

[8] P. Trigo and H. Coelho, *Simulating a Milti-Agent Electricity Market*, 2010

[9] P. Trigo, J. de Sousa and P. Marques, *Milti-agent Simulation od Electricity Markets Economically-Motivated decision-making*, 2013

[10] T. Pinto, G. Santos, Z. Vale, I. Parça, F.Lopes and H. Algarvio, *Realistic Multi-Agent Simulation of Competitive Electricity Markets*, 2014

[11] Z. Zhou, W. K. Chan and J. H. Chow, *Agent-based simulation of electricity markets: a survey of tools*, 2007

[12] Veit DJ, Weidlich A, Tao J, Oren S (2006) *Simulating the dynamics in two-settlement electricity markets via an agent-based approach.* Int J Manag Sci Eng Manag 1(2):83-97