# WCCI 2018 Competition
# Evolutionary Computation in Uncertain Environments: A Smart Grid Application

# Test bed:
# Optimal scheduling of distributed energy resources considering uncertainty of renewables, EVs, load forecast and market prices

**Fernando Lezama[1], João Soares[1], Zita Vale[1], and Jose Rueda[2]**

[1]School of engineering (ISEP), Polytechnic of Porto, Porto, Portugal

flzcl@isep.ipp.pt, joaps@isep.ipp.pt, zav@isep.ipp.pt

Delft University of Technology, Netherlands

j.l.ruedatorres@tudelft.nl

December 2018

**Table of contents**

# 1. Introduction

GECAD – Polytechnic of Porto –, in collaboration with Delft University, proposes the optimization of a centralized day-ahead energy resource management problem in smart grids under environments with uncertainty. Typically, the effect of uncertainty due to diverse factors (e.g., renewable generation or variable load consumption) is minimized or even ignored under the assumption that perfect (or highly accurate) forecast is available. However, in real-world applications, the effects of uncertainty cannot be neglected, and such assumption can lead to deviations that can compromise the entire system.

In this competition, we develop a framework of the energy resource management problem [1],[2] that considers the uncertainty associated with renewable generation, load forecast errors, electric vehicle scheduling and market prices. The consideration of these sources of uncertainty in the formulation adds more complexity to an already challenging problem.

This competition aims at the use of computational intelligence (CI) (e.g., evolutionary algorithms, swarm intelligence, search strategies) to solve the complex energy resource management problem under uncertain environments.

Due to the importance of the energy resource management in the energy community, several mathematical formulations have been successfully proposed to solve the problem [3]. However, due to the very dynamic evolution of power systems in the last years and the transformation of electrical grids mainly due to the development of smart grid technologies, the traditional formulations, which were designed for a completely different scenario, sometimes cannot deal with the problem efficiently.

It is in those situations, where traditional approaches fail, that CI has demonstrated being a very powerful tool. The test of CI and metaheuristics through competitions is not new in the scientific community. In fact, other competitions in the past in the energy domain have proved that CI is a competitive tool when near-optimal solutions are needed in short execution times [4].

Different from previous competitions proposed on the problem of energy resource management, in which the uncertainty of the environment is neglected, or it is assumed that a perfect forecast is available, we develop a framework closer to a real-world situation by introducing uncertainty regarding renewable generation, load forecast, EV scheduling and market prices. Our approach considers the generation of a high number of scenarios associated with those uncertainty sources. The consideration of different scenarios has an impact on the effectiveness of traditional approaches increasing the execution times dramatically. CI can tackle this issue by implementing the proper modifications to the existing algorithms or proposing ad-hoc techniques to deal with these types of problems.

The WWCI 2018 competition on *"Evolutionary Computation in Uncertain Environments: A Smart Grid Application"* has the purpose of bringing together and testing the more advanced CI techniques applied to an energy domain problem, namely the energy resource management problem under uncertain environments. The competition provides a coherent framework where participants and practitioners of CI can test their algorithms to solve a real-world optimization problem in the energy domain with uncertainty consideration, which makes the problem more challenging and worth to explore.

# 2. General description of the smart grid application

The problem considers an energy aggregator with aims of procuring energy needs from distributed resources and the electricity market. The aggregator looks for the minimization of operational costs while making revenues from selling energy in available electricity markets. Moreover, it may use its own assets, e.g., energy storage systems (ESS), to supply the load demand. In addition, a V2G feature that allows the use of energy in the battery of electric vehicles (EV), is also possible. The energy aggregator establishes bilateral energy contracts with those who seek electricity supply, e.g., residential and industry customers. In this case, it is assumed that the aggregator does not make profits from the supply of energy to fixed loads and EVs charging. The main idea is that the optimization software can perform the energy resource scheduling of the dedicated resources in the day-ahead context for the 24 hours of the following day.
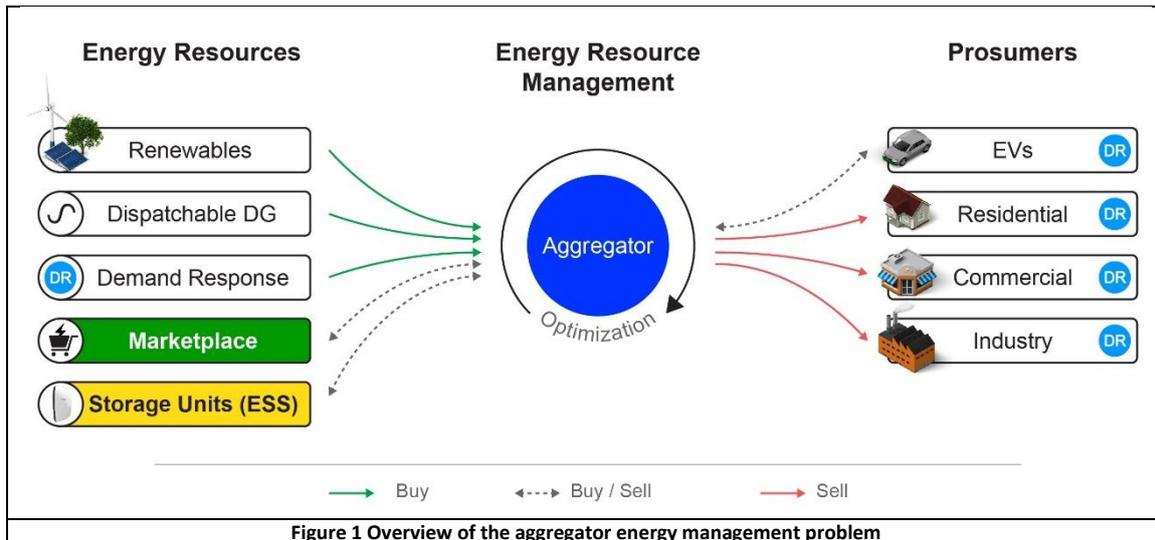


**Figure 1 Overview of the aggregator energy management problem**

Since the aggregator performs the scheduling of resources for the day-ahead (i.e., the next 24 hours), it relays in the forecast of weather conditions (to predict renewable generation), load demand, EV trips, and market prices. However, the assumption of "perfect" or "highly accurate" forecast might bring catastrophic consequences into the operation of the grid when the realizations do not follow the expected predictions.

Due to this situation, it is desired that the aggregator determines solutions that are robust to the uncertainty inherent in some parameters and the environment. Four aspects of uncertainty that affects the performance of a solution are considered in this competition, namely: a) Weather conditions, b) Load forecast, c) Planned EVs' trips, and d) Market prices.

Therefore, the aggregator should find solutions that provide not only an optimal (or near-optimal) value of operational costs but also those solutions must have the characteristic of being as less sensible as possible to the variations of the uncertain parameters. In [5], uncertainty in evolutionary computation is classified into four categories, namely noise, robustness, fitness approximation and time-varying fitness functions. This competition lays in the category of robustness, in which the design variables (or environmental parameters in this particular case) are subject to perturbations or changes after the optimal solution has been determined (i.e., the realizations of uncertain parameters).

To incorporate the uncertainty of parameters, we use Monte Carlo simulation (MCS) to generate a large number of possible scenarios using probability distribution functions of the forecast errors (obtained from historical data). A high number of scenarios increases the accuracy of the model but comes with a computations cost associated with a large number of variations in the parameters. Due to this, a reduction technique [1] is used to maintain a reasonably small number of scenarios while keeping the main statistical characteristics of the initial scenarios' set.

In the next section, we present the mathematical formulation of the problem, which provides a clear idea of the optimization problem that is solved in this competition.

# 3. Metaheuristic simulator framework

In this competition, the method of choice used by the participants to solve the problem must be a metaheuristic-based algorithm. The framework adopted in the competition is described in this document and follows the structure presented in **Figure 2**.
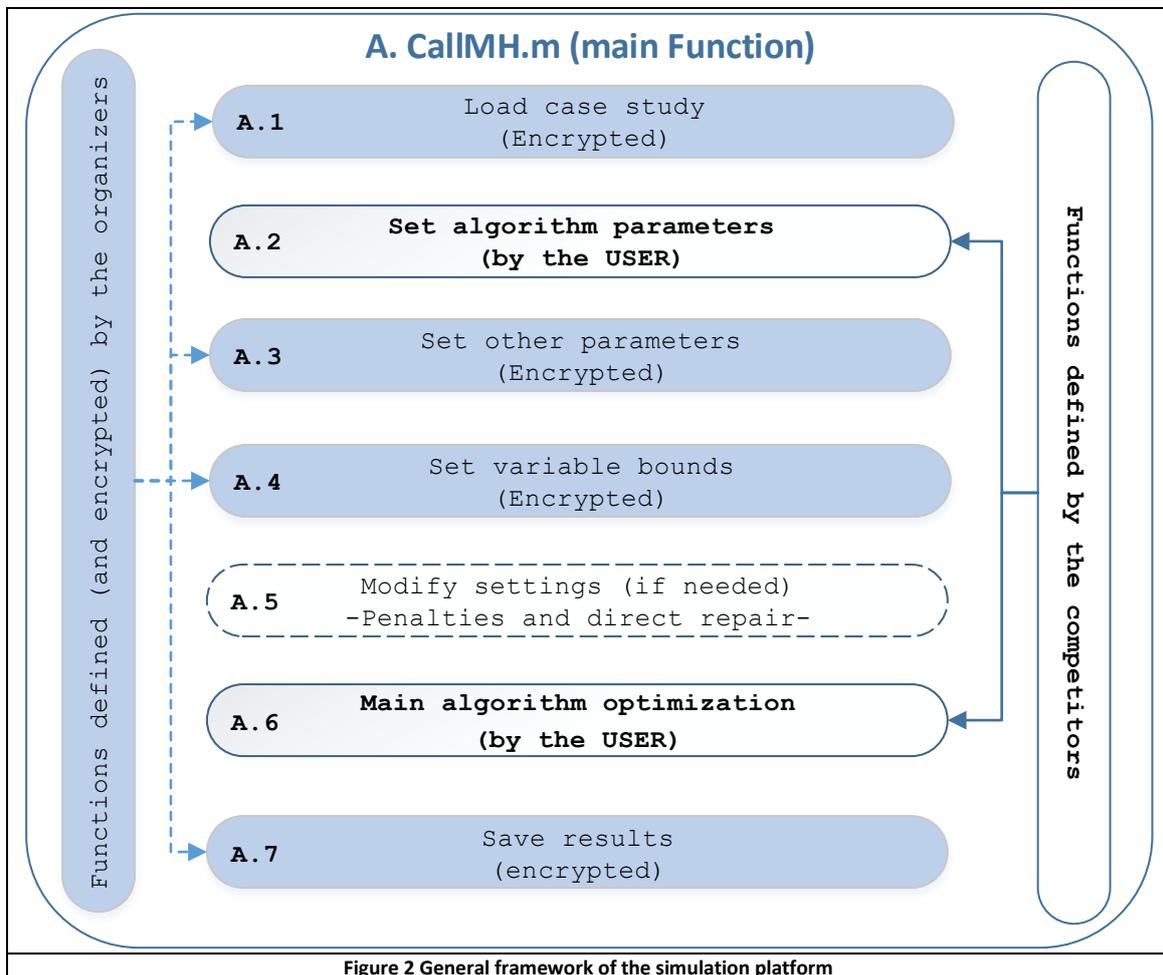


**Figure 2 General framework of the simulation platform**

The simulation platform has been implemented in MATLAB© 2016 64-bit, and consists of different scripts with specific targets in the simulation. As shown in **Figure 2**, some scripts correspond to encrypted files provided by the organizers (blue color in the figure). The user only needs to implement two scripts (see **Sect. 4.A.2** and **Sect. 4.A.6**), namely:

   i.   one script for setting the parameters required by their algorithm (A.2).
   ii.  a second script for the implementation of their proposed solution method (A.6).

Examples of how to implement these two script functions, and how the organizer's scripts work on the platform, are provided in **Sect. 4.**

Before of the guidelines for participants, we provide additional information on the encoding of the solutions, assumptions and some notes on the implementation of the problem below.

## 3.A) Encoding of the individual

The solution structure (e.g., an individual in DE, a particle in PSO, or genotype in GA) is a fundamental part of the metaheuristics to represent a given solution. The solution representation adopted in this competition follows the vector representation showed in **Figure 3**.

**Figure 3 Solution representation.**
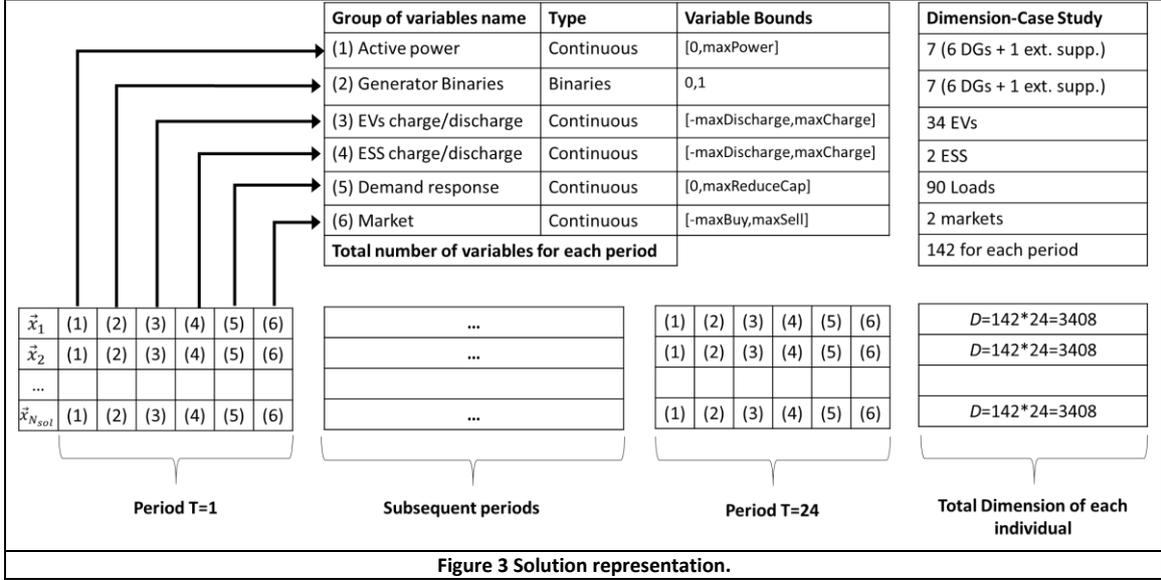
Each solution is encoded therefore as a vector with '6' groups of variables that are repeated sequentially across the 24 periods (hours) of optimization. In the vector representation, all variables, apart from group (2), are continuous variables with bounds matching the power or capacity limits of the associated variables. Group (2), *generator binaries*, corresponds to binary variables that are used to indicate if a generator is connected ('1' value) or disconnected ('0' value). Binaries variables might also present a continuous value since the fitness function internally correct their value using a simple round operation.

A special attention is pointed to group (1). That group belongs to variables of distributed generation (DGs). It is important to noticed that DGs include not only dispatchable generators but also PV generation. However, PV generation cannot be controlled, so even when it is part of the vector solution, the variable corresponding to PV generation (las variable of group (1)) will take a specific, and thus unalterable, value depending on the considered scenario.

## 3.B) Fitness function and uncertainty

The fitness function $f'$ considers the objective $Z$ of the aggregator (see Appendix section, Eq. (7)), plus the summation of the penalties found during evaluation of the solutions:

$$f'(\vec{X}) = Z + \rho \sum_{i=1}^{N_c} \max[0, g_i] \qquad (1)$$

where $\vec{X}$ is a solution that follows the structure showed in **Figure 3**. In this case, $g_i$ is the value of the $i$th constraint (equality or inequality) and $\rho$ is a configurable penalty factor (usually, a high value is considered). See **sect. 4.B** for instructions regarding fitness function and how penalties work.

In this competition, we consider uncertainty in some parameters that modify the value of the fitness function according to different scenarios generated by Monte Carlo simulation. The fitness function value is modifying by perturbation as follows:

$$F_s(\vec{X}) = f'(\vec{X} + \delta_s) \qquad (2)$$

where $\delta_s$ is the disturbance of variables and parameters in scenario $s$, and $F_s(\vec{X})$ is the fitness value associated to the $s$ Monte Carlo sampling. Therefore, an expected mean value for a given solution over the set of considered scenarios can be calculated as:

$$\mu FS(\vec{X}) = \frac{1}{N_s} \cdot \sum_{s=1}^{N_s} f'(\vec{X} + \delta_s) \qquad (3)$$

6

Similarly, the standard deviation of a solution over the set of scenarios can be calculated as:

$$\sigma FS(\vec{X}) = \sqrt{\frac{1}{N_s} \cdot \sum_{s=1}^{N_s} \left(f'(\vec{X} + \delta_s) - \mu FS(\vec{X})\right)^2} \qquad (4)$$

Eqs. (3) and (4) depends on the number of scenarios considered in the evaluation. As we will show below, the fitness function in the optimization process receive as a parameter the number of scenarios that the competitor wants to evaluate. However, keep in mind that for the final evaluation (See **Sect. 5**), the solutions will be evaluated through the total number of scenarios (100 for the competition).

**Figure 4** shows a schematic representation of the fitness function. We developed the fitness function as a black box as shown in **Figure 4(a)** (it is an encrypted function) that receives as input arguments an array with the solutions, the information of the case study, some additional parameters, and the number of scenarios that the user wants to evaluate (a maximum of 100 scenarios is considered). The function returns an array with the fitness values of the entire population over a randomly selected subset of scenarios (see **sect. 5.B** for details on the implementation of this function).

**Figure 4(b)** shows the internal operation of the fitness function, which randomly selects $N_{evalS}$ scenarios ($N_{evalS}$ is a parameter specified by the user) from the $N_s$ available ones. Notice from **Figure 4(b)** that the actual number of function's evaluations depends on the size of the population to evaluate, and the number of scenarios that the user wants to consider each time that the fitness function is called. The number of functions evaluations is therefore:

$$NFE = N_{sol} * N_{evalS} \qquad (5)$$

A maximum number of 50,000 function evaluations is allowed in the competition. Consider that each iteration of your algorithm could perform a variable number of function evaluations according to the number of times that the fitness function is called and Eq. (5).
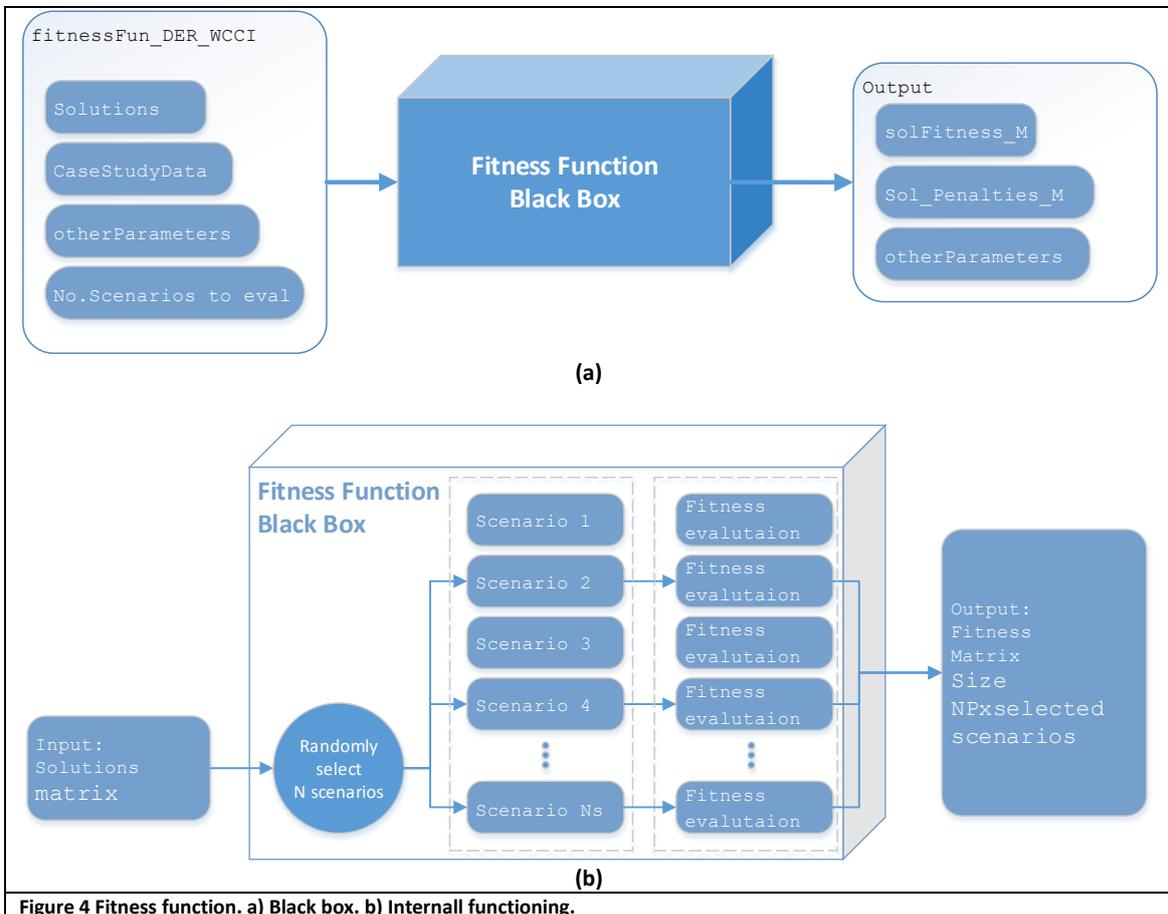


**Figure 4 Fitness function. a) Black box. b) Internall functioning.**

## 3.C) Some assumptions of the energy scheduling problem:

1. The aggregator minimizes operational costs while maximize its profits (costs minus income)
2. Electric vehicles can be controlled continuously (between 0 and max charge rate)
3. The same assumption applies to the V2G principle (between 0 and max discharge rate)
4. The stationary batteries or Energy Storage Systems (ESS) can be controlled continuously similar to the EVs/V2G
5. The cost function of DG units is assumed to be linear
6. It is assumed that the energy aggregator can submit bids and asks to the electricity market.
7. The markets in which the aggregator participates have different limits for bid and asks
8. Two markets are considered corresponding to wholesale and local markets
9. 5000 reduced to 100 scenarios are generated to simulate uncertainty of EVs travels, PV generation, load variations, and market prices

## 3.D) Some notes on the implementation of the problem:

1. Internally in the fitness function, it is assumed that the charge/discharge variables for the EVs are the same, but positive values for charge and negative values for discharge to save computational memory
2. The same principle described above for EV applies for the ESS variables
3. Internally, the market value is positive for an offer (sale) and negative for a buy bid
4. Binary variables are always rounded internally in the objective function
5. Direct repair of solution is used in the fitness function (see section 0)
6. The fitness function internally selects a random subset of the available 100 scenarios each time the function is called.

A maximum number of 50,000 evaluations is allowed in the competition. Take into account that it is not the same as algorithm iterations, and that each time the fitness function is evaluated, the actual number of function's evaluations varies according to Eq. (5).

## 3.E) Scenario overview

This section briefly describes the case study prepared for the competition, which is based on a 25-bus microgrid that represents a residential area with 6 DGs (5 dispatchable units and 1 PV generator), 1 external supplier, 2 ESSs, 34 EVs, and 90 loads with demand response capability. Moreover, it is considered that two markets (wholesale and local) are available for buy/sale of energy. Table 1 outlines the resources available in the MG.

*Table 1. Available Energy Resources*

| Energy resources | | Prices (m.u./kWh) | Capacity (kW) | Units |
|---|---|---|---|---|
| Dispatchable DGs | | 0.07-0.11 | 10-100 | 5 |
| External suppliers | | 0.074-0.16 | 0-150 | 1 |
| ESS | Charge | - | 0-16.6 | |
| | Discharge | 0.03 | 0-16-6 | 2 |
| EV | Charge | - | 0-111 | |
| | Discharge | 0.06 | 0-111 | 34 |
| DR curtailable loads | | 0.0375 | 4.06-8.95 | 90 |
| **Forecast (kW)** | | | | |
| Photovoltaic | | - | 0-106.81 | 1 (17 agg) |
| Load | | - | 35.82-83.39 | 90 |
| **Limits (kW)** | | | | |
| Market 1 (WS) | | 0.021-0.039 | 0-85 | 1 |
| Market 2 (LM) | | 0.021-0.039 | 0-40 | 1 |

**Uncertainty (generation of scenarios)**

For the competition, we created 5000 scenarios for PV generation, load consumption and market price variations. For the PV uncertainty generation, an error of 15% was used. Regarding the load forecasted and market prices, errors of 10% and 20% were used respectively. In a second step, the number of scenarios was reduced to 100 using specialized reduction techniques [7]. Regarding EVs trips, we have randomly generated 100 different forecast scheduling for each scenario using the tools in [8]. **Figure 5** shows graphically the generated scenarios.
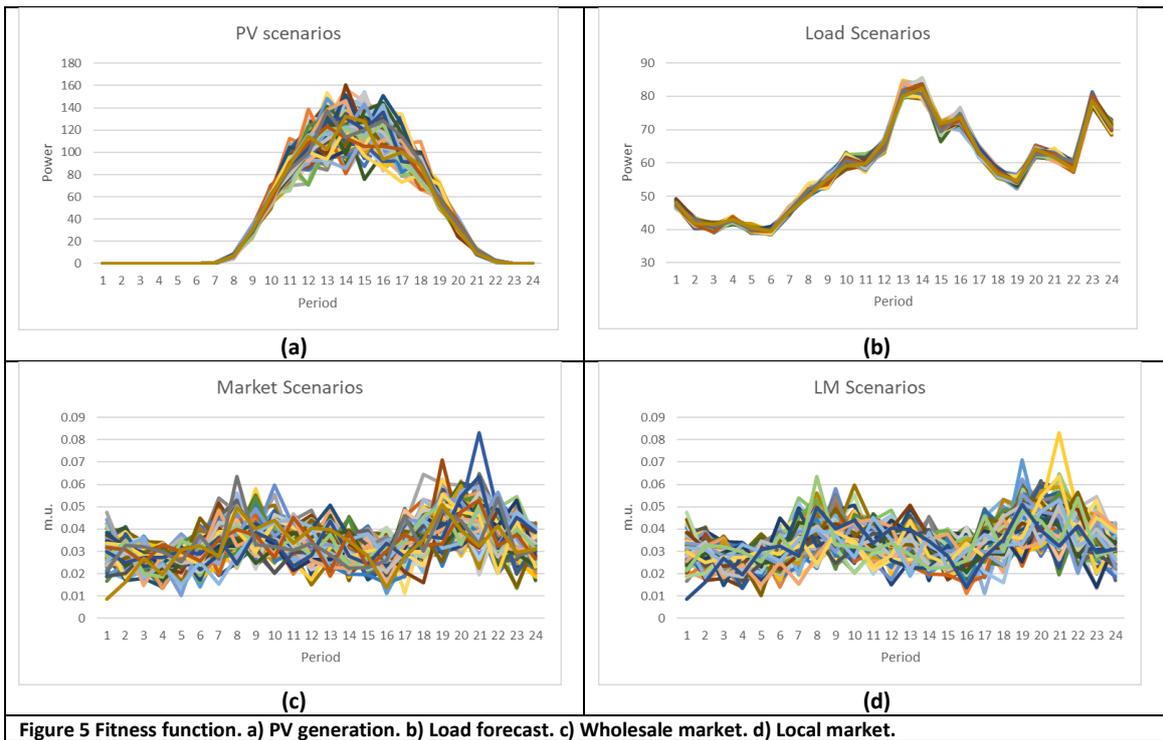
**Figure 5 Fitness function. a) PV generation. b) Load forecast. c) Wholesale market. d) Local market.**

# 4. Guidelines for participants

These instructions include as example the metaheuristic differential evolution (DE) [2] implemented and adapted to the energy resource management (It has been modified by GECAD).

It is important that the participants use the following recommendations and structure to avoid issues in using the supplied datasets and codes.

## 4.A) *mainWCCI_SG_2018.m -* Master function/script

**# *mainWCCI_SG_2018.m*** is the main file for the competition. The competitors can modify this main script as needed. Nevertheless, it is worth noting that this main script is ready to use. Participants should only include their functions (e.g., A.2-*MHparameters.m* and A.3-*MHalgorithm.m*) to perform the optimization of the problem.

***mainWCCI_SG_2018.m***

```matlab
clear;clc;close all; tTotalTime=tic; % lets track total computational time
noRuns = 2; % Number of trials here

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load Data base
caseStudyData=callDatabase(2);% No.Scenarios: (1) 10 (2) 100            [A.1]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load MH parameters (e.g., get MH parameters from DEparameters.m file)
algorithm='DE_rand'; %'The participants should include their algorithm here'
DEparameters %Function defined by the participant                       [A.2]
No_solutions=deParameters.I_NP; % Number of solutions used in the MH

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Set other parameters
otherParameters =setOtherParameters(caseStudyData,No_solutions);        [A.3]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Set lower/upper bounds of variables
[lowerBounds,upperBounds] = setVariablesBounds(caseStudyData,otherParameters); [A.4]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Some parameters that can be modified by the user
otherParameters.DirectMEthod=2; %1:without direct repair 2:With direct repairs (No
violations guarantee)                                                  [A.5]
otherParameters.ensPenalty=100; %Penalty factor:insufficient generation / energy not
supplied

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Call the MH for optimization
ResDB=struc([]);
    for iRuns=1:noRuns %Number of trails
        tOpt=tic;
        rand('state',sum(noRuns*100*clock))% ensure stochastic indpt trials

        [ResDB(iRuns).Fit_and_p, ...
         ResDB(iRuns).sol, ...
         ResDB(iRuns).fitVector, ...                                    [A.6]
         ResDB(iRuns).Best_otherInfo] =...
         deopt_simple(deParameters,caseStudyData,otherParameters,lowerB,upperB);

        ResDB(iRuns).tOpt=toc(tOpt); % time of each trial

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% Save the results and stats                                    [A.7]
        Save_results
    end
tTotalTime=toc(tTotalTime); %Total time
%% End of MH Optimization
```

As it can be seen, the main script follows the structure from Figure 2 (**Sect. 3**). Details in the implementation of each part of the code are given next.

## A.1 - *#callDatabase.m* - *Loading the case study datasets*

**# callDatabase.m (encrypted)** – This is an encrypted function to load the caseStudyData struct with all the relevant dataset information. Participants do not need to worry about the content of the case study and loading the files. "It is already done in the encrypted function". The participants, however, might want to change the argument to (2) to load a light version of the case study with only 10 scenarios. Argument (1) load the case study with 100 scenarios, which should be used in the competition for assessment of the results. The actual loading file is encrypted. The data set struct is caseStudyData.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load Data base
caseStudyData=callDatabase(1);
% No.Scenarios: (1) 100 scenarios
               (2) 10 scenarios
```

## A.2 - *#DEparameters.m* - *Set parameters of the metaheuristic*

**# DEparameters.m** file – This function file must be specific to the metaheuristic implemented by the participant. This is just an example using DE to show how participants should implement this function with all the parameters related to their algorithm.

```
deParameters.I_NP= 12; % Size of the population in DE
deParameters.F_weight= 0.3; %Mutation factor
deParameters.F_CR= 0.5; %Recombination constant
deParameters.I_itermax= 100; % number of max iterations/gen
deParameters.I_strategy   = 1; %DE strategy

deParameters.I_bnd_constr = 1; %Using bound constraints
% 1 repair to the lower or upper violated bound
% 2 rand value in the allowed range
% 3 bounce back
```

## A.3 - *#setOtherParameters.m* - *Set other necessary parameters and struct*

**# setOtherParameters.m (encrypted)** – This file is encrypted and should not be changed or modified by the user. It just sets parameters and data needed for the fitness function to work. It is a mandatory function that creates a struct "otherParameters" and should be run as illustrated in main function section:

```
%% Set other parameters
otherParameters =setOtherParameters(caseStudyData,No_solutions);
```

Participants must pass the "otherParameters" struct as argument to the functions:

```
[lowerBounds,upperBounds] = setVariablesBounds(caseStudyData,otherParameters);
[solFitness_M,solPenalties_M, temp] =
fitnessFun_DER_WCCI(solutions,caseStudyData,otherParameters,No_eval_Scenarios)
```

## A.4 - *#setVariablesBounds.m* - *Set bounds of variables*

**# setVariablesBounds.m (encrypted)** – This file is encrypted and should not be changed or modified by the user. It just sets the bounds of the problem variables:

```
%% Set lower/upper bounds of variables
[lowerBounds,upperBounds] = setVariablesBounds(caseStudyData,otherParameters);
```

The outputs of this function "[lowerBounds,upperBounds]" – should be used by your algorithm to generate the initial solutions and to validate if the bounds are being respected in each iteration.

The order of the variables in the implemented codes cannot be modify for the proper functioning of the fitness function. The structure of the solution is indicated in **Sect. 3.A** of this document

The following parameters are used to identify the ids of each type of variables. These "ids" are used to locate the type of variables in the solutions matrix (ids correspond to the columns while individuals to the rows).

```
otherParameters.ids.idsGen
otherParameters.ids.idsXGen
otherParameters.ids.idsV2G
otherParameters.ids.idsLoadDR
otherParameters.ids.idsStorage
otherParameters.ids.idsMarket
```

**Example of use:**
```
periods = caseStudyData.parameterData.numPeriods;
nParticles = size(solutions,1); %Number of population (solutions)
nVariables = size(solutions,2); %Number of variables (dimension)
idsV2G= otherParameters.ids.idsV2G;
getPeriod = 2; % Period 2 used to illustrate this example
tempIds=idsV2G+(nVariables/periods)*(getPeriod-1);
solutions(:,tempIds) % EVs variables for period 2, all individuals
solutions(2,tempIds) % EVs variables for period 2, second individual
```

## A.5 - Modifying some settings

If the participant wants to change the default assigned penalties regarding constraint violations, please add these lines and change the value accordingly. A tweak is accepted in the competition as some optimal penalties (i.e., penalties that adapt during the optimizations) may be suggested by the participants. These penalties should be 1 or higher or the fitness function will not accept it. If you don't want to change the default penalties just remove those lines from the master. Default penalty is 100 per each violation found.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Some parameters that can be modified by the user
otherParameters.DirectMEthod=2;
%1:without direct repair
%2:With direct repairs (No violations guarantee)
otherParameters.ensPenalty=100;
% Penalty factor:insufficient generation / energy not supplied
```

## A.6 - #deopt_simple.m - Algorithm proposed by the competitor

The participants should generate a scrip called **#MHalgorithm.m** or similar. This algorithm should replace **#deopt_simple.m** which is provided as example:

```
[ResDB(iRuns).Fit_and_p, ...
 ResDB(iRuns).sol, ...
 ResDB(iRuns).fitVector, ...
 ResDB(iRuns).Best_otherInfo] =...
  deopt_simple(deParameters,caseStudyData,otherParameters,lowerB,upperB);
```

Your metaheuristic should receive as input parameters:

1. **deParameters:** struct with the parameters configuration for your algorithm to work (it is generated by the user)
2. **caseStudyData:** struct with the information of the case study
3. **otherParameters:** struct with additional information required by the fitness function
4. **lowerB/upperB:** lower and upper bounds of variables

Your metaheuristic code should return to the main script the following variables:

1. **ResDB(iRuns).fit_and_p:** array of size 1x2 with the best fitness and penalties values
2. **ResDB(iRuns).sol:** vector of size: 1 x noVariables with the best candidate solution found by your algorithm
3. **ResDB(iRuns).fitVector:** array of size: 2xnoIterations with the value of the fitness and penalties over the iterations.
4. **ResDB(iRuns).Best_otherInfo:** Struct with additional information of the best individual (see sect. ()).

The participants are encouraged to save the results of each trial/run in a struct "**ResDB**", as shown in the example. That will ease the evaluation process by the organizers.

## A.7 - #Save_results.m - Benchmark results (text-files)

**#Save_results.m (encrypted)** – The output is written to text-files using this script. The following tables should be produced:

**Table 1. Table_Time:** Computing time spent for all optimization trials (benchmark_Time.txt)

|  | timeSpent (s) |
|---|---|
| Run1 |  |
| Run2 |  |
| Run3 |  |
| … |  |
| Run20 |  |

**Table 2. Table_Fit:** Individual benchmark of the trials (benchmark_Fitness.txt)

|  | AvgFit | StdFit | MinFit | MaxFit | varFit | ConvergenceRate | Penalties |
|---|---|---|---|---|---|---|---|
| Run1 |  |  |  |  |  |  |  |
| Run2 |  |  |  |  |  |  |  |
| Run3 |  |  |  |  |  |  |  |
| … |  |  |  |  |  |  |  |
| Run20 |  |  |  |  |  |  |  |

**Table 3. Table_TrialStats:** Summary statistics or the trials (benchmark_Summary.txt)

| Ranking Index | Average | Standard deviation | Minimum | Maximum | Variance | Code |
|---|---|---|---|---|---|---|
| RankingIndex | PAvgFit | PstdFit | PminFit | PmaxFit | PvarFit | validationCode |

In addition, this function should automatically generate the file "Send2Organizer.mat", which should include the best solutions found in each of the trials. That file will be used to double-check the reported results by validating all the solutions contained there over the 100 scenarios of the case study. For that reason, it is important that the participants put special care in returning the best solutions from their algorithms and stored in "ResDB.sol" (see Sect. 4.A.6).

To clarify, the "Send2Organizer.mat" file will include a matrix called *"solutions"* with the solutions stored in "ResDB.sol".  The solutions there will be evaluated according to Sect. 5 in order to double check the ranking index of each participant. The lower the ranking index, the better the performance of a participant.

**\*A number 20 trials should be made.**
**\*50,000 evaluations per trial should be made.**

## 4.B) Fitness function evaluation
***# fitnessFun_DER_WCCI.m* (encrypted) –** this is the fitness function to be used by participants:

```
function [solFitness_M,solPenalties_M, ExtraInfo] =
fitnessFun_DER_WCCI(solutions,caseStudyData,otherParameters,No_eval_Scenarios)
```

The function receives as input:

1. **solutions:** matrix of size $N_{sol} \times D$, in which $N_{sol}$ (rows) represents the number of individuals/solutions in an array, and $D$ (columns) represents the dimension (i.e., number of variables) of the optimization problem. This variable should be encoded in the metaheuristic algorithm proposed by participants (e.g., ***#MHalgorithm.m*, Sect. 4.A.6**). Only 1 individual is also possible (one row).
2. **caseStudyData:** struct with data of the case study with all the scenarios (100 for this competition) as loaded by callDatabase function (i.e., ***#callDatabase.m*, Sect. 4.A.1**).
3. **otherParameters:** Struct with additional information as loaded by ***#setOtherParameters.m* Sect. 4.A.3**).
4. $N_{evalS}$: The user can specify how many scenarios, from the available 100 in the case study, wants to evaluate. The fitness function selects randomly a subset of "$N_{evalS}$" from the total scenarios (See **Sect 3.B**).

The function returns as output:

1. $solFitness_M$: Matrix of size $N_{sol} \times N_{evalS}$, in which $N_{sol}$ (rows) represents the number of individuals and $N_{evalS}$ (columns) represents the number of evaluated scenarios. This matrix includes the fitness values of the solutions across different scenarios.



**Figure 6 Example of $solFitness_M$. Fitness of $N_{sol} = 5$ individuals over $N_{evalS} = 6$ scenarios.**

2. $solPenalties_M$: Matrix of size $N_{sol} \times N_{evalS}$, in which $N_{sol}$ (rows) represents the number of individuals and $N_{evalS}$s (columns) represents the number of evaluated scenarios. This matrix includes the penalty values of the solutions across different scenarios. (If direct repair is used, the penalties are always zero).



**Figure 7 Example of $solPenalties_M$. Fitness of $N_{sol} = 5$ individuals over $N_{evalS} = 6$ scenarios.**

3. **ExtraInfo:** Struct with additional valuable information captured during the evaluation phase.



**Figure 8 Example output of ExtraInfo struct returned by *#fitnessFun_DER.m***

The ***#fitnessFun_DER_WCCI.m*** evaluates all the population at once. Notice that the criterion to select the winner in this competition will be the average value over the total 100 scenarios plus the standard deviation (see **Sect. 5**). However, selecting 100 scenarios each time that a solution is evaluated will increase the computational cost and the number of fitness functions evaluations (see **Sect. 3.B**) which has a limit of 50000.

## *B1. Direct repair of solutions:*

The ***#fitnessFun_DER_WCCI.m*** as some mechanisms to provide a fast convergence of the solutions, namely direct repair of solutions. This means that the solutions returned by the function are changed if some constraints are not satisfied according to a heuristic mechanism. Some notes are explained:

1. Charge/discharge rates of EVs/ESS are adjusted considering the energy remaining the battery and the maximum capacity of the batteries (no need for penalties, as the correction are guaranteed).
2. direct repair balance: the demand/generation balance is made using a merit sort process and using the market to guarantee balance.

We have included the option of deactivate direct repair balance. The participant can control whether to use this feature by modifying line 28 in the main scrip:

```
         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         %% Some parameters that can be modified by the user
Line 28: otherParameters.DirectMEthod=2;
         %1:without direct repair
         %2:With direct repairs (No violations is guaranteed)
         otherParameters.ensPenalty=100;
         % Penalty factor:insufficient generation / energy not supplied
```

Note: For the validation of results, organizers will perform fitness evaluations always considering direct repair balance, i.e., `otherParameters.DirectMEthod=2` will always set to this value.

## B2. Best solution and storing additional information

Since *#fitnessFun_DER_WCCI.m* does not return a single value associated to an individual, but the evaluation of individuals across scenarios, the participants should select a criterion to determine which is the best individual in their population, or how they want to perform the search. The criteria for selecting the best individual could vary from worst-case performance, mean fitness value, best fitness value, etcetera. Here, we provide an example of selecting the best individual based on the worst-case performance:

```
[solFitness_M, solPenalties_M,Struct_Eval]=
fitnessFun_DER_WCCI(solutions,caseStudyData,otherParameters,No_eval_Scenarios);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The user should decide which is the best criterion to optimize.
% In this example, we optimize worst-case performance
[S_val, worstS]=max(solFitness_M,[],2); %Find worst-case performance
[~,I_best_index] = min(S_val); %Select the best amount the worst-case performances

FVr_bestmemit = FM_pop(I_best_index,:); % best member of current iteration
```

In your MH function, please add some code like the one provided below after determining the best candidate solution:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% store other information
Best_otherInfo.idBestParticle = I_best_index;
Best_otherInfo.genCostsFinal =
Struct_Eval(worstS(I_best_index)).otherParameters.genCosts(I_best_index,:);
Best_otherInfo.loadDRcostsFinal =
Struct_Eval(worstS(I_best_index)).otherParameters.loadDRcosts(I_best_index,:);
Best_otherInfo.v2gChargeCostsFinal =
Struct_Eval(worstS(I_best_index)).otherParameters.v2gChargeCosts(I_best_index,:);
Best_otherInfo.v2gDischargeCostsFinal
=Struct_Eval(worstS(I_best_index)).otherParameters.v2gDischargeCosts(I_best_index,:);
Best_otherInfo.storageChargeCostsFinal =
Struct_Eval(worstS(I_best_index)).otherParameters.storageChargeCosts(I_best_index,:);
Best_otherInfo.storageDischargeCostsFinal =
Struct_Eval(worstS(I_best_index)).otherParameters.storageDischargeCosts(I_best_index,:);
Best_otherInfo.stBalanceFinal =
Struct_Eval(worstS(I_best_index)).otherParameters.stBalance(I_best_index,:,:);
Best_otherInfo.v2gBalanceFinal =
Struct_Eval(worstS(I_best_index)).otherParameters.v2gBalance(I_best_index,:,:);
Best_otherInfo.penSlackBusFinal =
Struct_Eval(worstS(I_best_index)).otherParameters.penSlackBus(I_best_index,:);
```

*indexbest* is the index for the best candidate solution (determined by the participant according to the preferred method). Please store *indexbest* in *Best_otherInfo.idBestParticle*.

# 5. Evaluation guidelines

A ranking index will be calculated using the 20 final solutions (one for each trial) provided by each participant. With these solutions, the organizers will calculate the ranking index ($RI_{user}$) for each participant $a$ based on the average fitness and standard deviation of each solution across the 100 scenarios:

$$RI_{user(a)} = \frac{1}{N_{trials}} \cdot \left[ \sum_{i=1}^{N\_trials} \left( \mu FS_a(\vec{X_i}) + \sigma FS_a(\vec{X_i}) \right) \right] \qquad (6)$$

where $\mu FS_a(\vec{X_i})$ and $\sigma FS_a(\vec{X_i})$ are functions that return the average value and standard deviation of the solution found in trial $i$ (i.e., $\vec{X_i}$) by participant $a$ across the 100 considered scenarios (See **Sect. 3.B**).

Therefore, the winner of the competition will be the one that gets the minimum value of $RI_{user}$. The participants must consider this criterion while selecting the best search strategy in their algorithms. With this performance measurement, we are considering not only the best mean expected value, but also the robustness of the solution.

# 6. Material to be submitted to the organizers

For the validation of the results, the 3 benchmark text files and the "Send2Organizer.mat" file produced by **# Save_results.m** (see **Sect. 4.A.7**) should be submitted to the organizers. The implementation codes of each algorithm entering the competition must also be submitted along with final results for full consideration in the evaluation. The submitted codes will be used for further tests, which are intended to crosscheck the submitted results. The submitted codes will be in the public domain and no intellectual property claims should be made.

Each participant is kindly requested to put the text files corresponding to final results, as well as the implementation files (codes), obtained by using a specific optimizer, into a zipped folder named

WCCI2018_SG_*AlgorithmName_ParticipantName*.zip
(e.g. WCCI2018_SG_*DE_Lezama*.zip).

The zipped folder must be summited to flzcl@isep.ipp.pt and joaps@isep.ipp.pt

by 15th May 2018

# Appendix: Mathematical formulation

We divide this section in three parts for better understanding: A) Objective function, B) Constraints of the problem, and C) Uncertainty modelling.

## A) Objective function

The envisaged problem can be modelled as a combinatorial Mixed-Integer Linear Programming (MILP) problem due to the presence of a high number of continuous, discrete and binary variables. The objective of the aggregator is to minimize operational costs ($OC$) while maximizing incomes ($In$). This can be rewritten as minimization function $Z$:

$$minimize\ Z = OC - In \tag{7}$$

The minimum value of $Z$ is the total cost (or profits if negative) for the energy aggregator. Therefore, the goal in optimization terms is to obtain the minimum value of $Z$ in the metaheuristics form.

The aggregator looks for the minimization of the operational costs ($OC$) associated with the management of resources as follows:

$$OC = \sum_{s=1}^{N_s} \sum_{t=1}^{T} \begin{pmatrix} \sum_{t=1}^{T}\sum_{i=1}^{N_{DG}} P_{DG(i,t)} \cdot C_{DG(i,t)} + \sum_{t=1}^{T}\sum_{k=1}^{N_k} P_{ext(k,t)} \cdot C_{ext(k,t)} \\ \sum_{j=1}^{N_{PV-DG}} \boldsymbol{P_{PV(j,t,s)}} \cdot C_{PV(j,t)} + \sum_{e=1}^{N_e} \boldsymbol{P_{ESS^-(e,t,s)}} \cdot C_{ESS^-(e,t)} \\ \sum_{v=1}^{N_v} \boldsymbol{P_{EV^-(v,t,s)}} \cdot C_{EV^-(v,t)} + \sum_{l=1}^{N_L} \boldsymbol{P_{curt(l,t,s)}} \cdot C_{curt(l,t)} \\ \sum_{l=1}^{N_L} \boldsymbol{P_{imb^-(l,t,s)}} \cdot C_{imb^-(l,t)} + \sum_{i=1}^{N_{DG}} \boldsymbol{P_{imb^+(i,t,s)}} \cdot C_{imb^+(i,t)} \end{pmatrix} \cdot \pi(s) \tag{8}$$

Eq. **Error! Reference source not found.** considers the cost associated with Distributed Generation (DG), external suppliers, discharge of ESS and EVs, DR by direct load control programs (curtailable loads), penalization of non-supplied demand (negative imbalance) and penalization for excess of DG units' generation (positive imbalance).[1]

On the other hand, the aggregator can receive its incomes ($In$) from market transactions as follows:

$$MT = \sum_{s=1}^{N_s} \sum_{t=1}^{T} \left( \sum_{m=1}^{N_m} \left( P_{buy(m,t)} - P_{sell(m,t)} \right) \cdot MP_{(m,t,s)} \right) \cdot \pi(s) \tag{9}$$

where offers and bids are allowed in two markets with distinctive characteristics, namely wholesale and local markets.**Error! Bookmark not defined.**

## B) Constraints of the problem

The problem constraints are similar to [6]. The problem is mainly constrained by the energy balance constraint (Eq. 8), DG generation and supplier limits in each period, ESS capacity, charge and discharge rate limits, EVs capacity, EVs' trips requirements, charge and discharge efficiency and rate limits. For the competition, to simplify the problem we have neglected the network constraints regarding reactive powers balance, voltage and angle limits.

The main constraint to fulfill in the formulation is the active power balance constraint which states that the amount of generated energy should be equal to the amount of consumed energy at every instant $t$:

$$\sum_{i=1}^{N_{DG}} P_{DG(i,t)} + \sum_{k=1}^{N_k} P_{ext(k,t)} + \sum_{j=1}^{N_{PV-DG}} \boldsymbol{P_{PV(j,t,s)}} + \sum_{e=1}^{N_e}\left(\boldsymbol{P_{ESS^-(e,t,s)}} - \boldsymbol{P_{ESS^+(e,t,s)}}\right) + \sum_{v=1}^{N_v}\left(\boldsymbol{P_{EV^-(v,t,s)}} - \boldsymbol{P_{EV^+(v,t,s)}}\right)$$
$$+ \sum_{l=1}^{N_L}\left(\boldsymbol{P_{curt(l,t,s)}} - \boldsymbol{P_{load(l,t,s)}}\right) + \sum_{m=1}^{N_m}\left(\boldsymbol{P_{buy(m,t)}} - \boldsymbol{P_{sell(m,t)}}\right) + \sum_{i=1}^{N_{DG}} P_{imb^+(i,t,s)} - \sum_{l=1}^{N_L} P_{imb^-(l,t,s)}$$
$$= 0 \qquad \forall t, \forall s \tag{10}$$

---

[1] See nomenclature at the end of this subsection.

It can be noticed that the balance constraint must be satisfied for all the possible uncertain scenarios $s$, which require solutions that are robust to the variations of uncertain variables/parameters.

## C) Uncertainty representation

We assume that a correct set of scenarios that simulate real-world conditions can be generated considering forecast and associated errors based on historical data or previous experiences. The uncertainty in this problem comes from: i) PV renewable sources, ii) load profiles, iii) EVs' scheduling, and iv) market prices for wholesale and local markets.

We apply the technique for scenario generation (and scenario reduction) used in [1]. In a first step, a large number of scenarios is generated by Monte Carlo Simulation (MCS). The MCS uses the probability distribution function of the forecasted errors (which can be obtained from historical data) to create a number of scenarios according to:

$$X_s(t) = x^{forecast}(t) + x^{error,s}(t) \qquad (11)$$

Where $x^{error,s}$ is a normal distribution function with zero-mean and standard deviation $\sigma$, and $x^{forecast}(t)$ is the forecasted valued of variable $x$ at time $t$. To simplify, all forecast errors for the uncertain inputs are represented by a normal distribution function. In a second step, a standard scenario reduction technique is applied that excludes scenarios with low probabilities and combines those that are close to each other in terms of statics metrics (for a complete description of these techniques see [1]).

For the competition, we created 5000 scenarios for PV generation, load consumption and market price variations. For the PV uncertainty generation, an error of 15% was used. Regarding the load forecasted and market prices, errors of 10% and 20% were used respectively. In a second step, the number of scenarios was reduced to 100 using specialized reduction techniques [7]. Regarding EVs trips, we have randomly generated 100 different forecast scheduling for each scenario using the tools in [8].

Participants should design their algorithms to find solutions with optimal fitness and robust behavior over the 100 provided scenarios.

## Nomenclature

| Indices | | Parameters | |
|---|---|---|---|
| $i$ | Distributed generation (DG) units | $N_{DG}$ | Number of DG |
| $j$ | PV units | $N_{PV}$ | Number of PV |
| $k$ | External suppliers | $N_k$ | Number of external suppliers |
| $e$ | Energy storage systems (ESSs) | $N_e$ | Number of ESSs |
| $v$ | Electric vehicles (EVs) | $N_v$ | Number of EVs |
| $l$ | Loads | $N_L$ | Number of loads |
| $m$ | Markets | $N_m$ | Number of markets |
| $s$ | Scenarios | $N_s$ | Number of scenarios |
| $t$ | Periods | $T$ | Number of periods |
| **Variables** | | $C_{DG}$ | Generation cost of DG (m.u./kWh) |
| $P_{DG}$ | Active power generation (kW) | $C_{ext}$ | Cost of external supplier (m.u./kWh) |
| $P_{ext}$ | External supplied power (kW) | $C_{PV}$ | Cost of PV generation (m.u./kWh) |
| $P_{ESS^-}$ | Discharge power of ESS (kW) | $C_{ESS^-}$ | Discharging cost of ESS (m.u./kWh) |
| $P_{EV^-}$ | Discharge power of EV (kW) | $C_{EV^-}$ | Discharging cost of EV (m.u./kWh) |
| $P_{ESS^+}$ | Charge power of ESS (kW) | $C_{curt}$ | Load curtailment cost (m.u./kWh) |
| $P_{EV^+}$ | Charge power of EV (kW) | $C_{imb}$ | Grid imbalance cost (m.u./kWh) |
| $P_{curt}$ | Power reduction of load (kW) | | |
| $P_{imb^-}$ | Non-supplied power for load (kW) | $\pi(s)$ | Probability of scenario $s$ |
| $P_{imb^+}$ | Exceeded power of DG unit (kW) | $P_{PV}$ | Photovoltaic generation (kW) |
| $P_{buy}$ | Power buy to the market (kW) | $P_{load}$ | Forecasted load |
| $P_{sell}$ | Power sell to the market (kW) | $MP$ | Electricity market price (m.u./kWh) |
| $x_{DG}$ | Binary variable for DG status | | |

**Bibliography**

[1]     J. Soares, B. Canizes, M. A. Fotouhi Gazvhini, Z. Vale, and G. K. Venayagamoorthy, "Two-stage Stochastic Model using Benders' Decomposition for Large-scale Energy Resources Management in Smart grids," *IEEE Trans. Ind. Appl.*, pp. 1–1, 2017.

[2]     F. Lezama, J. Soares, E. Munoz de Cote, L. E. Sucar, and Z. Vale, "Differential Evolution Strategies for Large-Scale Energy Resource Management in Smart Grids," in *GECCO '17: Genetic and Evolutionary Computation Conference Companion Proceedings*, 2017.

[3]     W. L. Theo, J. S. Lim, W. S. Ho, H. Hashim, and C. T. Lee, "Review of distributed generation (DG) system planning and optimisation techniques: Comparison of numerical and mathematical modelling methods," *Renew. Sustain. Energy Rev.*, vol. 67, pp. 531–573, Jan. 2017.

[4]     "http://sites.ieee.org/psace-mho/2017-smart-grid-operation-problems-competition-panel/," 2017. .

[5]     Y. Jin and J. Branke, "Evolutionary Optimization in Uncertain Environments—A Survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, Jun. 2005.

[6]     J. Soares, C. Lobo, M. Silva, H. Morais, and Z. Vale, "Relaxation of non-convex problem as an initial solution of meta-heuristics for energy resource management," in *2015 IEEE Power & Energy Society General Meeting*, 2015, pp. 1–5.

[7]     N. Gröwe-Kuska, H. Heitsch, and W. Römisch, "Scenario reduction and scenario tree construction for power management problems," in *2003 IEEE Bologna PowerTech - Conference Proceedings*, 2003, vol. 3, pp. 152–158.

[8]     J. Soares, B. Canizes, C. Lobo, Z. Vale, and H. Morais, "Electric Vehicle Scenario Simulator Tool for Smart Grid Operators," *Energies*, vol. 5, no. 12, pp. 1881–1899, Jun. 2012.