WCCI/GECCO 2020 Competition Evolutionary Computation in Uncertain Environments: A Smart Grid Application

João Soares[1], Fernando Lezama[1], Bruno Canizes[1], Zita Vale[1]
[1]School of engineering (ISEP), Polytechnic of Porto, Porto, Portugal
jan@isep.ipp.pt, flz@isep.ipp.pt, brmrc@isep.ipp.pt, zav@isep.ipp.pt

IEEE CIS Task Force on 'Computational Intelligence in the Energy Domain (ci4energy), part of IEEE CIS Intelligent Systems Applications TC (http://ci4energy.uni-paderborn.de/committee/)

IEEE PES Intelligent Systems Subcommittee (ISS), part of IEEE PES Analytic Methods for Power Systems TC (http://sites.ieee.org/pes-iss/)

December 2019

Table of contents

1. Introduction

Following the success of the previous editions at WCCI 2018 and CEC 2019[1] we are launching a more challenging competition at major conferences in the field of computational intelligence. This WCCI 2020 competition proposes two test beds in the energy domain:

Testbed 1) optimization of a centralized day-ahead energy resource management problem in smart grids under environments with uncertainty. This test bed is similar to the past challenge using a challenging 500-scenario case study with high degree of uncertainty. We also add some restrictions to the initialization of initial solution and the allowed repairs and tweak-heuristics.

Testbed 2) bi-level optimization of end-users' bidding strategies in local energy markets (LM). This test bed is constructed under the same framework of the past competitions (therefore, former competitors can adapt their algorithms to this new testbed) , representing a complex bi-level problem in which competitive agents in the upper-level try to maximize their profits, modifying and depending on the price determined in the lower-level problem (i.e., the clearing price in the LM), thus resulting in a strong interdependence of their decisions.

The proposed testbeds are developed under the same framework of the past competitions with a number of adjustments.

[1] Check former competitions in http://www.gecad.isep.ipp.pt/WCCI2018-SG-COMPETITION/ and http://www.gecad.isep.ipp.pt/ERM2019-Competition

2. General description of the smart grid applications

2.1 Testbed 1

The problem of Testbed 1 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.

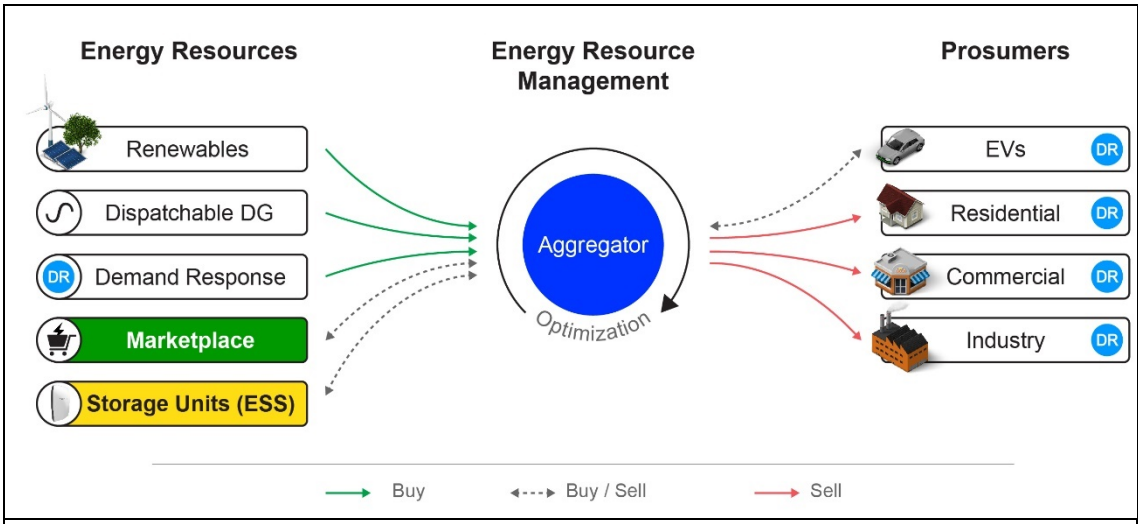

Figure 1 Overview of the aggregator energy management problem

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 [1], 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 [2] is used to maintain a reasonably small number of scenarios while keeping the main statistical characteristics of the initial scenarios' set..

2.2 Testbed 2

We consider a day-ahead LM bidding optimization problem, in which agents submit bids/offers to maximize their profits (or equivalently minimize their costs). We assume that agents of the type consumers only, small producers, and prosumers (i.e., consumers with generation capabilities) coexist in this Local Market (LM). Also, agents have access to the main grid, which works as a back-up system. Therefore, as in [3], agents can trade energy in the LM with prices between the feed-in tariff c^F and the grid electricity tariff c^G. In fact, it is assumed that $c^F<c^G$ and therefore buy/sell energy from the grid is less beneficial to agents than transacting energy in the LM. **Figure 2** illustrates the local market scenario.

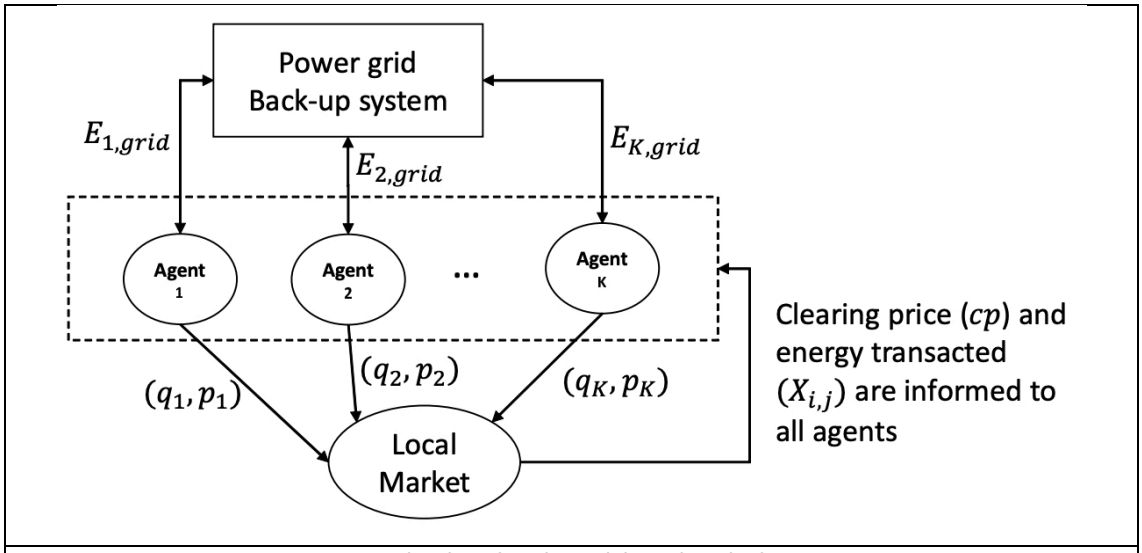

Figure 2 Considered Local Market and the grid as a back-up system

The LM bidding optimization problem can be modeled as a bi-level optimization problem. The upper-level corresponds to the maximization of agents' profits, and the lower-level problem corresponds to the maximization of energy transacted in the LM. Therefore, the solution of the lower-level (after determining the clearing price) affects the upper-level by modifying the profits of all agents.

Consider a set of consumer agents i= {1,2,...N_c}, and producer agents j= {1,2,...N_p}, where each agent i wants to minimize its costs while agents j want to maximize their profits. The upper problem, therefore, is a multi-objective problem in which each agent wants to maximize/minimize their profits/costs.

The full formulation of the problem is available in the publication [4].

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 (e.g. Differential Evolution, Particle Swarm Optimization, Vortex Search, etc.). The framework adopted in the competition is described in this document and follows the structure presented in **Figure 3**.

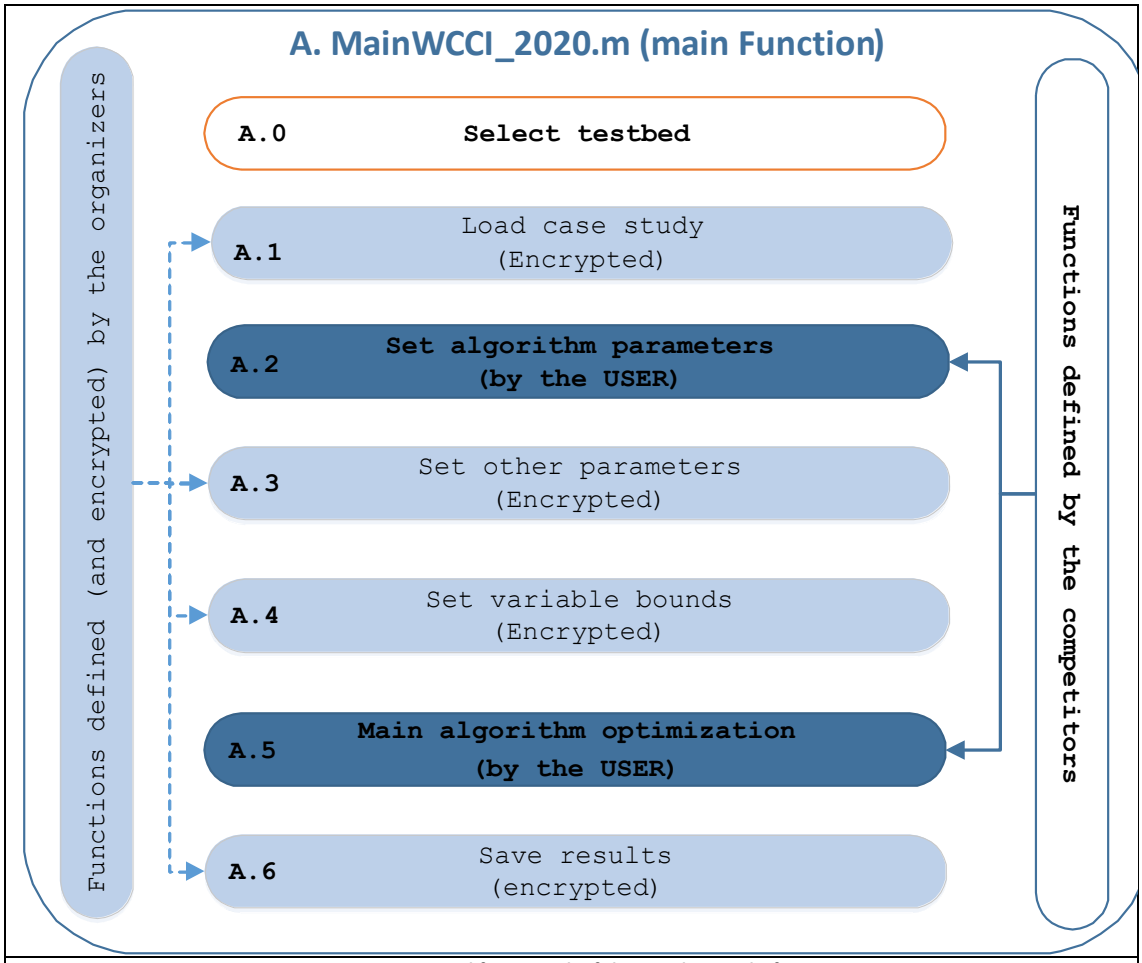

Figure 3 General framework of the simulation platform

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 3**, 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.

A maximum number of 50,000 evaluations is allowed in the competition for each testbed. Take into account that it is not the same as algorithm iterations, and that each time the fitness function is evaluated.

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 4**. The initial solutions in this edition of the competition should be initialized randomly between the upper and lower bounds of the variables. Heuristics and special tweaks are not allowed.

Testbed 1

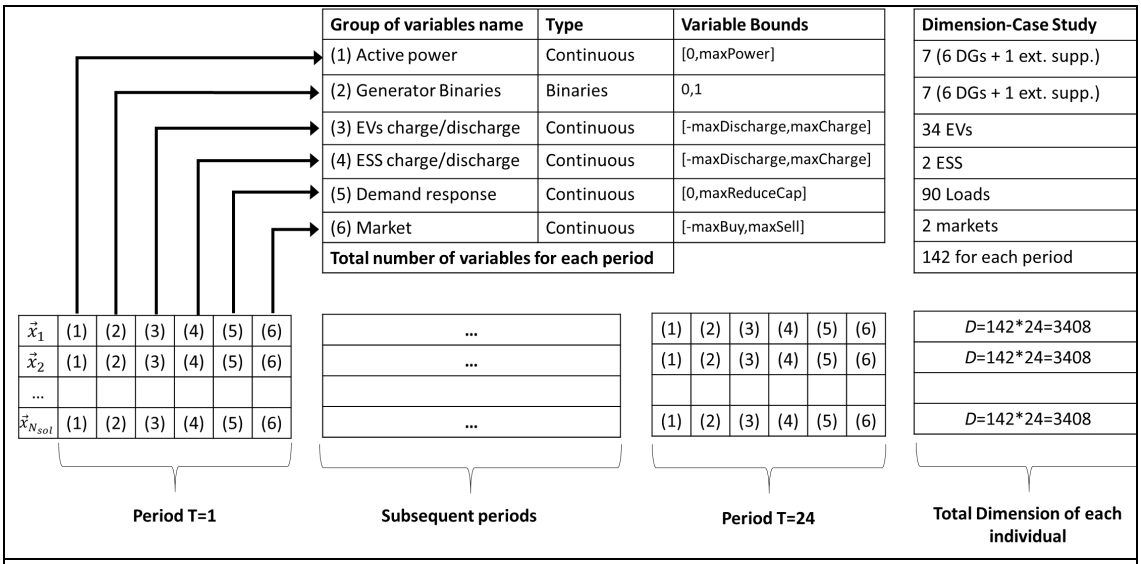

Figure 4 Solution representation in testbed 1

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). Binary variables might also present a continuous value since the fitness function internally corrects 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 notice 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 variables corresponding to PV generation (variables of group (1)) will take a specific, and thus unalterable, value depending on the considered scenario.

Testbed 2

The optimization problem, seen as a whole, searches for the optimal bidding of agents in the LM to maximize their profits. Therefore, assuming we have $K = \{1,2,...N_k\}$[1], we seek to determine the best tuple $(q_k , p_k)\forall k \in K$ representing the optimal price and quantity to bid in the LM for each agent. The bidding also should be done for all $t \in T$ periods of the optimization process (i.e., T = 24 periods in the day-ahead market). Therefore, we define a vector $\vec{x} = \{[q_{k,t}] \cup [p_{k,t}]\}$ including the bids for quantity and price the kth agent will send to the LM. To avoid separating the agents by consumer and producer types, we use a sign convention in which a positive quantity represents a bid (i.e., buying in the market), while a negative quantity represents an offer (i.e., selling in the LM). Therefore, we can control the agent action by defining variable bounds in which a consumer agent can send bids in the market within the bounds $[0, L_{max}]$ (i.e., between 0 and their maximum consumption), while producer agents can send offers within the bounds $[-P_{max}, 0]$ (i.e., between 0 and their maximum production capacity). The bounds for prices are the same for all agents and within the range $[c^F , c^G]$. **Figure 5** illustrates the structure of solutions to understand how the individual is encoded.

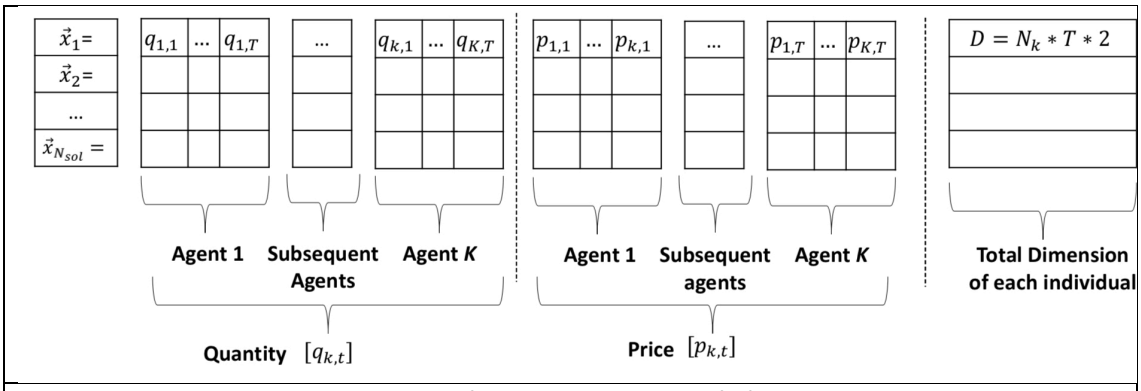

Figure 5 Solution representation in testbed 2

3.B) Fitness function

A maximum number of 50,000 function evaluations is allowed in the competition in each testbed.

Testbed 1

The fitness function f' considers the objective Z of the aggregator (see Appendix section, Eq. (10)), plus the summation of the penalties found during evaluation of the solutions:

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

where $\vec{X}$ is a solution that follows the structure showed in **Figure 4**. In this case, g_i is the value of the ith constraint (equality or inequality) and ρ 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:

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

where δ_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:

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

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

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

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 (500 for the competition).

Figure 6 shows a schematic representation of the fitness function. We developed the fitness function as a black box as shown in **Figure 6(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 500 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 6(b) shows the internal operation of the fitness function, which randomly selects N_{evalS} scenarios (N_{evalS} is a parameter not specified by the user. It is set to a default value of 10 and cannot be changed) from the N_s available ones. Notice from **Figure 6(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:

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

Recall that a maximum number of 50,000 function evaluations is allowed in the competition.

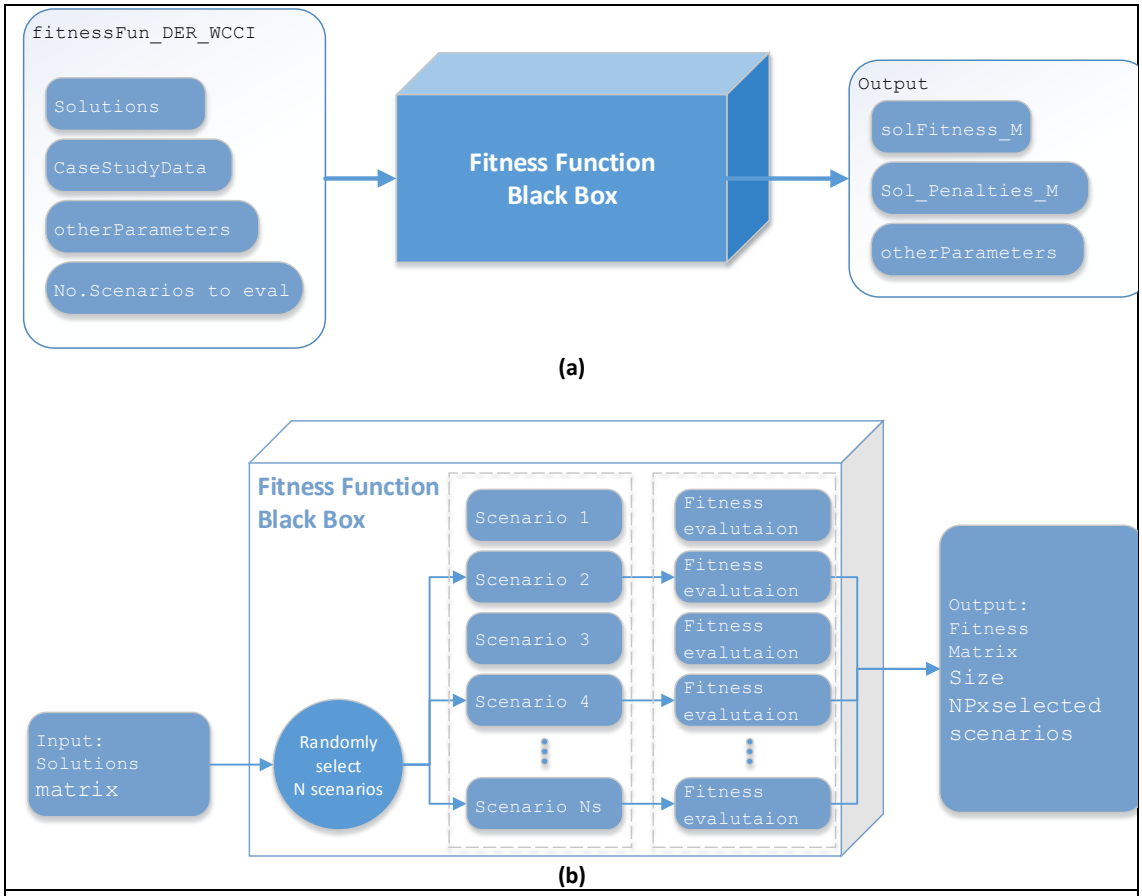

Figure 6 Fitness function. a) Black box. b) Internall functioning.

Testbed 2

Solutions in testbed 2 should be evaluated in an objective function that returns the mean average profit of all agents plus the standard deviation:

$$f'(\vec{X}) = mean(profits) + std(profits) \quad (6)$$

where *mean(Profits)* and *std(Profits*) are functions that compute the average and standard deviation (respectively) of the profits that all agents obtained considering the bids/offers encoded in the individual. The negative sign in the first term is used to transform the profits maximization problem into a minimization one. The less the value in Eq. (6), the better the mean profits achieved by all agents. The fitness function in testbed 2 acts as a black box as **Figure 6**.

3.C) Some assumptions:
Testbed 1

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 500 scenarios are generated to simulate uncertainty of EVs travels, PV generation, load variations, and market prices

3.D) Some notes on the implementation:
Testbed 1

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 500 scenarios each time the function is called.

3.E) Scenario overview
Testbed 1

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	2
	Discharge	0.03	0-16.6	
EV	Charge	-	0-111	34
	Discharge	0.06	0-111	
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	0-100	1
Market 2 (LM)	0.021-0.039	0-10	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 500 using specialized reduction techniques [5]. Regarding EVs trips, we have randomly generated 500 different forecast scheduling for each scenario using the tools in [6]. **Figure 7** shows graphically the generated scenarios.

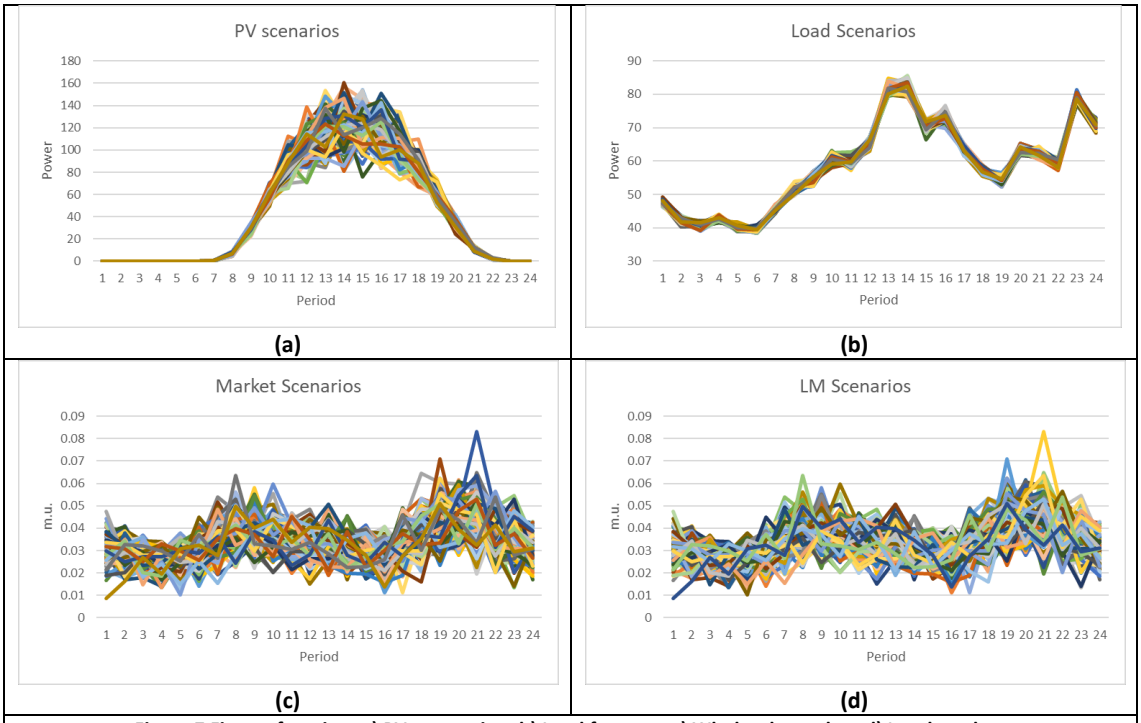

Figure 7 Fitness function. a) PV generation. b) Load forecast. c) Wholesale market. d) Local market.

Testbed 2

We adopt a case study with nine agents, in which 3 of them are consumers, 3 are prosumers (i.e., consumers with PV generation capabilities), and 3 are CHP generators. To generate case study data, sample power profiles of residential houses and PV systems are built using the open datasets available in PES ISS website[2]. We build three standard house power profiles and a PV power profile (see **Figure 8**). With these profiles, we generate agent data using a randomized function with uniform distribution, 20% around the standard profiles. **Figure 8** also provides the power ranges of the base profiles. We consider generator agents corresponding to CHPs with a maximum generation capacity of 2kW and a marginal cost calculated using the equation provided in [4] with a factor of b_CHP=0.18 EUR/kWh. Finally, feed-in and grid tariffs are set to C^F=0.12$ and C^G=0.28 EUR/kWh as in [4].

[2] Open data online at http://sites.ieee.org/pes-iss/data-sets/

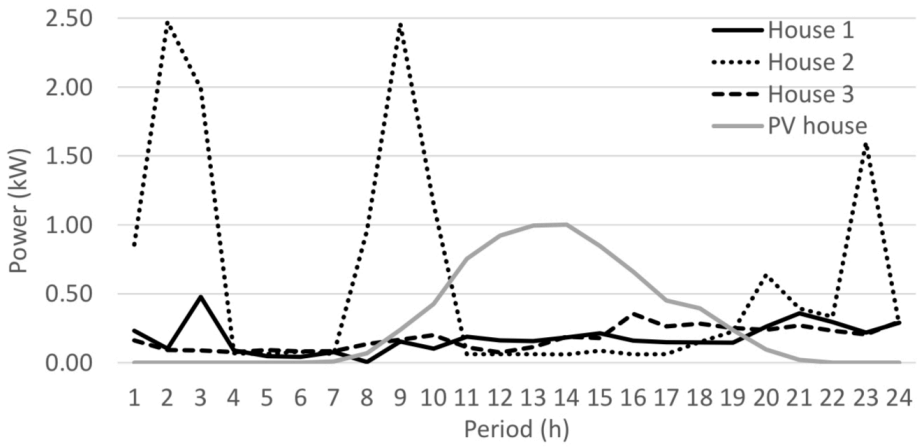

Figure 8 Profiles used in the case study. Ranges of power (in kW): house 1 [0.18-0.48], house 2 [0.06-2.50], house 3 [0.07-0.36], PV (house) [0-1].

4. Guidelines for participants

These instructions include as example the metaheuristic differential evolution (DE) [7] implemented and adapted to the present framework (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_2020.m* - Master function/script

 # mainWCCI_SG_2020.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 to perform the optimization of the problem.

mainWCCI_SG_2020.m

A.0
```
……
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Select testbed
Select_testbed=1;
%Testbed 1: Energy resource management with uncertainty
%Testbed 2: Optimal bidding in local energy markets
```
A.1
```
DB=Select_testbed;
% 1: Case study testbed 1
% 2: Case study testbed 2
[caseStudyData, DB_name]=callDatabase(DB);
```
A.2
```
Select_Algorithm=1;
%1: DE algorithm (test algorithm)
%2: Your algorithm
algorithm='DE-rand'; %'The participants should include their algorithm here'
DEparameters %Function defined by the participant
No_solutions=deParameters.I_NP; % %Notice that some algorithms are limited to one individual
```
A.3
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Set other parameters
otherParameters =setOtherParameters(caseStudyData,No_solutions, Select_testbed);
```
A.4
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Set lower/upper bounds of variables
[lowerBounds,upperBounds] = setVariablesBounds(caseStudyData,otherParameters);
```
A.5
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 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

        otherParameters.iRuns=iRuns;
            switch Select_Algorithm
                case 1
                    [ResDB(iRuns).Fit_and_p, ...
                    ResDB(iRuns).sol, ...
                    ResDB(iRuns).fitVector]= ...
deopt_simple(deParameters,caseStudyData,otherParameters,lowerB,upperB);
….
     end
```
A.6
```
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% Save the results and stats
        Save_results

….
```

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

A.0 and A.1 - # mainWCCI_SG_2020.m - Loading the testbed and case study

 # mainWCCI_SG_2020.m – This is the main framework file where you can select the testbed (either 1 or 2), which will load the caseStudyData struct (callDatabase.p – encrypted) with all the relevant dataset information depending on the selected testbed. Participants do not need to worry about the content of the case study and loading the files.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Select_testbed=1;
%Testbed 1: Energy resource management with uncertainty
%Testbed 2: Optimal bidding in local energy markets
%% Load Data base
noRuns=20; %this can be changed but final results should be based on 20 trials
DB=Select_testbed;
% 1: Case study testbed 1
% 2: Case study testbed 2
[caseStudyData, DB_name]=callDatabase(DB);
```

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. Please take into account a third parameter is added to the previous framework to consider another testbed. 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, Select_testbed);
```

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

```
[lowerBounds,upperBounds] = setVariablesBounds(caseStudyData,otherParameters,
Select_testbed);
…..
[ResDB(iRuns).Fit_and_p, ...
                    ResDB(iRuns).sol, ...
                    ResDB(iRuns).fitVector]= ...
deopt_simple(deParameters,caseStudyData,otherParameters,lowerB,upperB);
```

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. Please take into account a third parameter is added to the previous framework to consider another testbed.

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

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 (Example of testbed 1 – only works for testbed 1). 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 testbed 1:

```
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 - *#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]= ...
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.

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.6 - *#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 500 scenarios of the case study (testbed 1) and one scenario in testbed 2. 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 and #fitnessFun_WCCI2020.m* (encrypted) –** this is the fitness function to be used by participants and should be called as below. The "fnc" parameter will be assigned automatically to load the corresponding fitness function according to the selected testbed **Sect. 4.A.0**.

```
[S_val, ~]=feval(fnc,FM_pop,caseStudyData,otherParameters);
```

The function receives as input:

1. **fnc**: string with the fitness function m file name: and "fitnessFun_DER_WCCI.m" for testbed 1 and "#fitnessFun_WCCI2020.m" for testbed 2.
2. **FM_pop:** 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.5**). Only 1 individual is also possible (one row).
3. **caseStudyData:** struct with data of the case study with all the scenarios as loaded by callDatabase function (i.e., ***#callDatabase.m*, Sect. 4.A.1**).
4. **otherParameters:** Struct with additional information as loaded by ***#setOtherParameters.m* Sect. 4.A.3**).

The function returns as output:

1. **S_val**: Matrix of size N_{sol} represents the number of individuals. This matrix includes the fitness values including penalties of the solutions across different scenarios.

The ***#fitnessFun*** evaluates all the population (individuals) at once. A maximum number of 50,000 function evaluations is allowed in the competition in each testbed. The table below helps the participant to have an idea of the maximum number of iterations and population It can set without surpassing the allowed number of evals. So, take it account when designing your algorithm:

Table 1. Algorithm population/iterations limits

Size of the population	Max. iterations Testbed 1	Max. iterations Testbed 2
1	5000	50000
5	1000	10000
20	250	2500
50	100	1000
100	50	500
1000	5	50

B1. Best solution

Since ***#fitnessFun*** returns 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);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% 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
```

5. Evaluation guidelines

A ranking index will be calculated using the 20 final solutions (one for each trial) provided by each participant in each testbed 1 and 2 (T1 and T2). 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 500 scenarios in testbed 1 and the average fitness in testbed 2. The values $RI_{user(a)\_T1}$ and $RI_{user(a)\_T2}$ will then be normalized and a final ranking will be produced.

$$RI_{user(a)\_T1} = \frac{1}{N_{trials}} \cdot \left[\sum_{i=1}^{N\_trials} \left(\mu FS_a(\vec{X}_i\_T1) + \sigma FS_a(\vec{X}_i\_T1) \right) \right] \quad (7)$$

where $\mu FS_a(\vec{X}_i\_T1)$ and $\sigma FS_a(\vec{X}_i\_T1)$ 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 500 considered scenarios (See **Sect. 3.B**).

$$RI_{user(a)\_T2} = \frac{1}{N_{trials}} \cdot \left[\sum_{i=1}^{N\_trials} \left(\mu FS_a(\vec{X}_i\_T2) \right) \right] \quad (8)$$

$$RI_{user(a)\_FINAL} = \frac{1}{Norm(RI_{user(a)\_T1})} + \frac{1}{Norm(RI_{user(a)\_T2})} \quad (9)$$

Therefore, the winner of the competition will be the one that gets the maximum 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.6**) 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:

WCCI2020_testbedX_*AlgorithmName_ParticipantName*.zip
(e.g. *WCCI2020*_testbed2_*DE_Lezama*.zip).

The zipped folder must be summited to jan@isep.ipp.pt; flz@isep.ipp.pt and brmrc@isep.ipp.pt

by 31th May 2020

Appendix: Mathematical formulation of testbed 1

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:

$$minimize\ Z = OC - In \tag{10}$$

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:

$$OC = \sum_{s=1}^{N_s}\sum_{t=1}^{T}\left(\begin{array}{l}\sum_{i=1}^{T}\sum_{i=1}^{N_{DG}} P_{DG(i,t)}\cdot C_{DG(i,t)} + \sum_{t=1}^{T}\sum_{k=1}^{N_k} P_{ext(k,t)}\cdot C_{ext(k,t)} \\ \sum_{j=1}^{N_{PV-DG}} \boldsymbol{P_{PV(j,t,s)}}\cdot C_{PV(j,t)} + \sum_{e=1}^{N_e} \boldsymbol{P_{ESS^-(e,t,s)}}\cdot C_{ESS^-(e,t)} \\ \sum_{v=1}^{N_v} \boldsymbol{P_{EV^-(v,t,s)}}\cdot C_{EV^-(v,t)} + \sum_{l=1}^{N_L} \boldsymbol{P_{curt(l,t,s)}}\cdot C_{curt(l,t)} \\ \sum_{l=1}^{N_L} \boldsymbol{P_{imb^-(l,t,s)}}\cdot C_{imb^-(l,t)} + \sum_{i=1}^{N_{DG}} \boldsymbol{P_{imb^+(i,t,s)}}\cdot C_{imb^+(i,t)}\end{array}\right)\cdot \pi(s) \tag{11}$$

OC, 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).[3]

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

$$MT = \sum_{s=1}^{N_s}\sum_{t=1}^{T}\left(\sum_{m=1}^{N_m}\left(P_{buy(m,t)} - P_{sell(m,t)}\right)\cdot MP_{(m,t,s)}\right)\cdot \pi(s) \tag{12}$$

where offers and bids are allowed in two markets with distinctive characteristics, namely wholesale and local markets.

B) Constraints of the problem

The problem constraints are similar to [8]. 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:

$$\begin{array}{l}\sum_{i=1}^{N_{DG}} P_{DG(i,t)} + \sum_{k=1}^{N_k} P_{ext(k,t)} + \sum_{j=1}^{N_{PV-DG}} \boldsymbol{P_{PV(j,t,s)}} + \sum_{e=1}^{N_e}\left(\boldsymbol{P_{ESS^-(e,t,s)}} - \boldsymbol{P_{ESS^+(e,t,s)}}\right) + \sum_{v=1}^{N_v}\left(\boldsymbol{P_{EV^-(v,t,s)}} - \boldsymbol{P_{EV^+(v,t,s)}}\right) \\ + \sum_{l=1}^{N_L}\left(\boldsymbol{P_{curt(l,t,s)}} - \boldsymbol{P_{load(l,t,s)}}\right) + \sum_{m=1}^{N_m}\left(\boldsymbol{P_{buy(m,t)}} - \boldsymbol{P_{sell(m,t,s)}}\right) + \sum_{i=1}^{N_{DG}} P_{imb^+(i,t,s)} - \sum_{l=1}^{N_L} P_{imb^-(l,t,s)} \\ = 0 \qquad \forall t, \forall s\end{array} \tag{13}$$

[3] 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 [2]. 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:

$$X_s(t) = x^{forecast}(t) + x^{error,s}(t) \quad (14)$$

Where $x^{error,s}$ is a normal distribution function with zero-mean and standard deviation σ, 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 [2]).

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 500 using specialized reduction techniques [5]. Regarding EVs trips, we have randomly generated 500 different forecast scheduling for each scenario using the tools in [6].

Participants should design their algorithms to find solutions with optimal fitness and robust behavior over the 500 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] 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.

[2] 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.

[3] E. Mengelkamp, J. Gärttner, and C. Weinhardt, "Intelligent agent strategies for residential customers in local electricity markets," in *e-Energy 2018 - Proceedings of the 9th ACM International Conference on Future Energy Systems*, 2018.

[4] F. Lezama, J. Soares, and Z. Vale, "Optimal Bidding in Local Energy Markets using Evolutionary Computation," in *20th Internantional Conference on Intelligent System Applications to Power Systems*, 2019.

[5] 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.

[6] 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, 2012.

[7] 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.

[8] 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.