CEC/GECCO 2021 Competition
Evolutionary Computation in Uncertain Environments: A Smart Grid Application

Fernando Lezama[1], João Soares[1], Bruno Canizes[1], Zita Vale[1], Ruben Romero[2]
[1]School of engineering (ISEP), Polytechnic of Porto, Porto, Portugal
[2]UNESP, Ilha Solteira, São Paulo, Brazil
flz@isep.ipp.pt, jan@isep.ipp.pt, brmrc@isep.ipp.pt, zav@isep.ipp.pt, ruben.romero@unesp.br

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/)

January 2021

Table of contents

1. Introduction

Following the success of the previous editions at PES GM 2017, WCCI 2018, CEC 2019 and WCCI 2020[1] we are launching a new edition of our algorithm competition at major conferences in the field of computational intelligence. This CEC/GECCO 2021 competition proposes two tracks in the energy domain:

Track 1) Bi-level optimization of bidding strategies in local energy markets (LEM). This testbed is constructed under the same framework of the past competitions (therefore, former competitors can adapt their algorithms to this new track), 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 LEM), thus resulting in a strong interdependence of their decisions.

Track 2) Flexibility management of home appliances to support DSO requests. A model for aggregators flexibility provision in distribution networks that takes advantage of load flexibility resources allowing the re-schedule of shifting/real-time home-appliances to provision a request from a distribution system operator (DSO) is proposed. The problem can be modeled as a Mixed-Integer Non-Linear Programming (MINLP) in which the aggregator strives to match a flexibility request from the DSO/BRP, paying a remuneration to the households participating in the DR program according to their preferences and the modification of their baseline profile.

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

Tip: The most important part for a quick participation in this competition is section 4 of this guidelines, i.e., if you want to just implement your heuristic and treat your problems as pure black box items. Former sections (2-3) are introduction and explanation of the problem being optimized.

[1] Check former competitions in http://www.gecad.isep.ipp.pt/ERM-Competitions

2. General description of the smart grid applications

2.1 Track 1 - Bi-level optimization of bidding strategies in local energy markets

In track 1, a day-ahead local energy market (LEM) bidding optimization problem is considered, 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 LEM. Also, agents have access to the WS market through an aggregator. Therefore, similar to [1], agents can trade energy in the LM with prices between the feed-in tariff c^F and the wholesale market plus an aggregator fee c^{AGG}. In fact, it is assumed that $c^F<c^{AGG}$ and therefore buy/sell energy from the grid/aggregator is less beneficial to agents than transacting energy in the LEM. **Figure 1** illustrates the local market scenario.

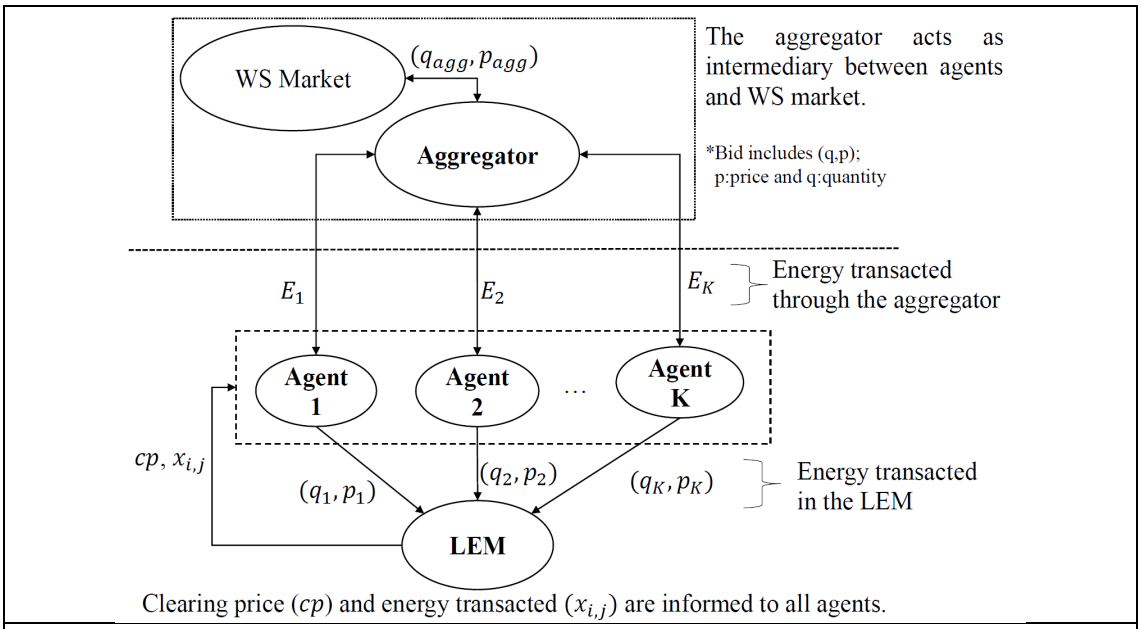

Figure 1 Considered Local Market and market participants

The LEM 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 $i = \{1,2, \dots, N_c\}$, and producer agents $j = \{1,2, \dots, N_p\}$, 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.

More details about the formulation of the problem are available in the publication [2].

2.2 Track 2 - Flexibility management of home appliances to support DSO requests

In the track 2, the optimization problem is the flexibility management of home appliances to support DSO requests. The problem considers an aggregator which is in control of the management of devices with DR capabilities [3]. The users register voluntarily for participation in flexibility provision receiving monetary compensation if their baseline profile is modified.

Features and assumptions of the optimization model in track 2:

- The perspective of an aggregator in charge of a HEMS with different devices with DR capabilities.

- Two types of devices are considered for DR, namely devices which consumption can be shifted to a different period, and devices with real-time control capabilities.
- The aggregator is prepared to respond to a flexibility request from a DSO or a BRP, who pays monetary compensation for each power unit (p.u.) of flexibility provisioned.
- The aggregator makes use of a flexibility management system to re-schedule some appliances and match, as close as possible, the flexibility curve procured by the DSO.
- End-users have the capability of registering devices for flexibility provision and configuring their preferences regarding allowed shiftable times, expected remuneration due to flexibility activation, a priority of the available devices for activation, among others.
- We assume that the required infrastructure for achieving such management and control (e.g., smart metering systems, communication lines, HEMS) is in place.
- Both, the DSO/BPR and the aggregator, have access to forecast baseline power consumption provided by a third party (the baseline represents normal consumption in case no DR is activated)

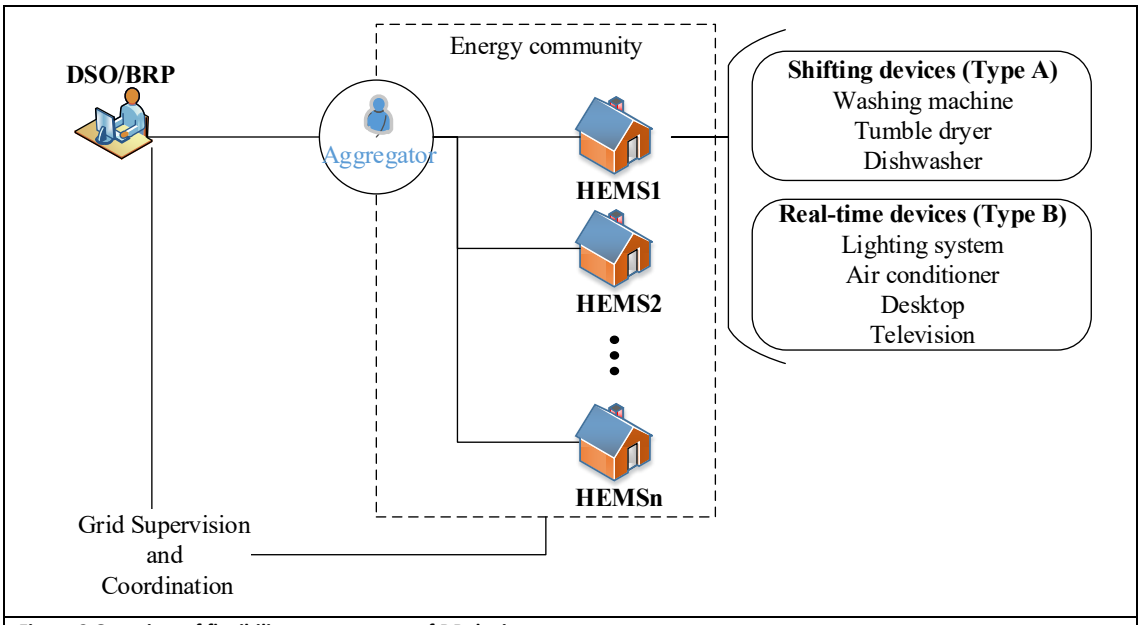

Figure 2 Overview of flexibility management of DR devices

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, Hybrid approaches, etc.). The framework adopted in the competition is described in this document and follows the structure presented in **Figure 3**.

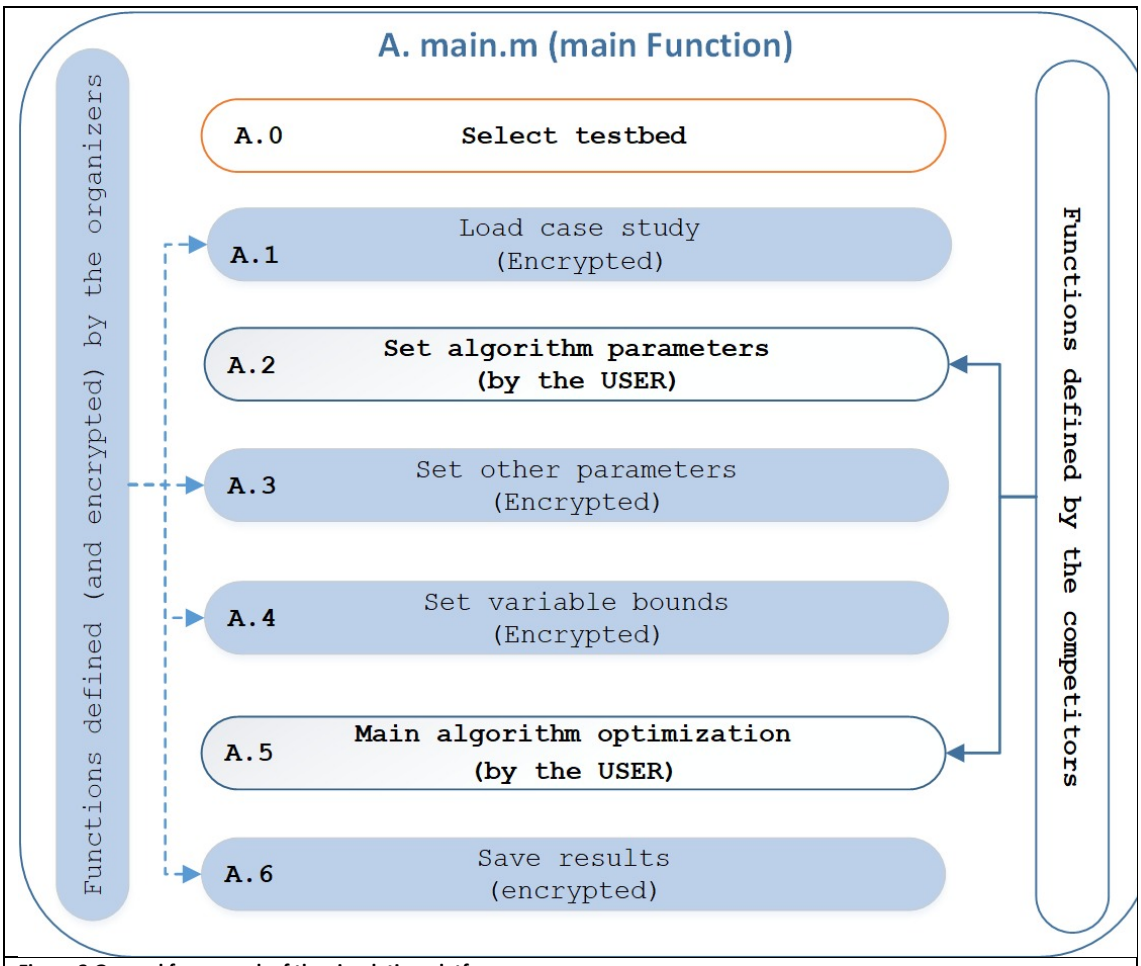

Figure 3 General framework of the simulation platform

The simulation platform has been implemented in MATLAB© 2018 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 10,000 function evaluations for track1, and 100,000 function evaluations for track2 are 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.

3.A) Encoding of the individual

One fundamental aspect of population-based algorithms is the encoding of the solutions. Depending on the problem, particles/vectors must contain all the information associated with a solution and should be evaluated in a fitness function in order to measure their performance.

The initial solutions should be initialized randomly between the upper and lower bounds of the variables. Heuristics and special tweaks are not allowed. This rule is valid for both tracks.

Track 1

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 of track 1 problem follows the vector representation showed in **Figure 4**.

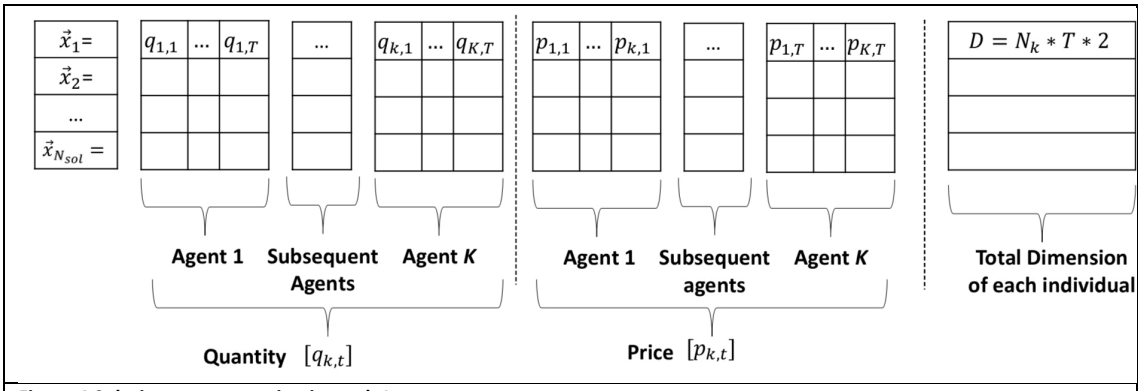

Figure 4 Solution representation in track 1

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,...Nk}[1], we seek to determine the best tuple (qk , pk)∀k ∈ K representing the optimal price and quantity to bid in the LM for each agent. The bidding also should be done for all t ∈ T periods of the optimization process (i.e., T = 24 periods in the day-ahead market). Therefore, we define a vector $\vec{x}$ = {[qk,t] ∪ [pk,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^{AGG}]. **Figure 5** illustrates the structure of solutions to understand how the individual is encoded.

Track 2

For the specific problem a given solution must contain the all the new starting periods of appliances with shifting capabilities (type A) and the new intensities of real-time appliances (type B).

Type A appliances:

The information related with appliances type A can be encoded in a vector that indicates the new starting times of each appliance:

$$x_{shift} = [T_{new(1)}, T_{new(2)}, \dots, T_{new(i)}]$$

x_{shift} contains the decision variables$T_{new(i)}$ corresponding to the new starting period of appliance *i* ∈ A (Type A).

Type B appliances:

New intensities for appliances typeB should be defined for all of their operation periods and are encoded as:

$$x_{int} = [Int_{new(1,1)}, \dots, Int_{new(1,NT)}, Int_{new(2,1)}, \dots, Int_{new(2,NT)}, \dots, Int_{new(Nj,1)}, \dots, Int_{new(Nj,NT)}]$$

where x_{int} contains the decision variables $Int_{new(j,t)}$ that represents the new intensity of the jth appliance in the operation period t

The vectors x_{shift} and x_{int} are concatenated to form a new vector that represents a solution (e.g. a particle in PSO).

$X = [x_{shift}, x_{int}]$

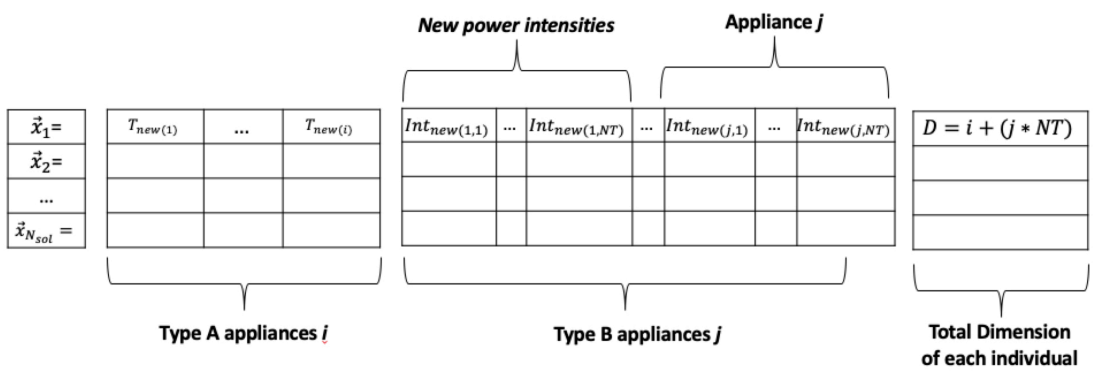

Figure 5 Solution representation in track 2

3.B) Fitness function

A maximum number of 10,000 function evaluations for track1 and 100,000 for track 2, are allowed in the competition.

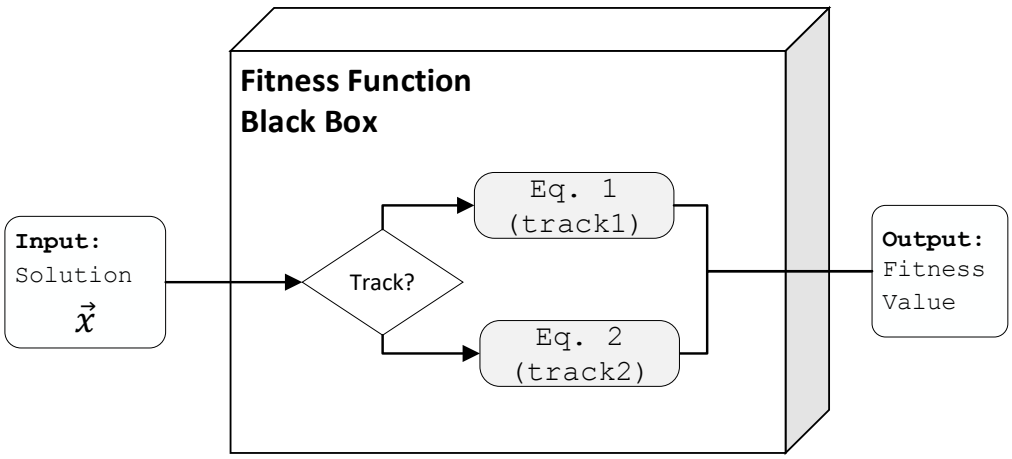

Figure 6 Fitness function as black box .

Track 1

Track 1 should be evaluated in an objective function that returns the mean average profit of all agents plus the standard deviation:

$$f'(\vec{X}) = mean(profits) + std(profits) \quad (1)$$

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. (1), the better the mean profits achieved by all agents. The fitness function in track 1 acts as a black box as **Figure 6**.

Track 2

Solutions in track 2 should be evaluated with fitness function (2). In order to maximize the aggregator profits, the fitness function can be modelled as the minimization of the remuneration to be paid to the households plus a penalty for the mismatch of flexibility procured by the DSO/BRP (F_{match}). The fitness function *f(x)* in track 2 acts as a black box as **Figure 6**.

$$\text{Minimize } f = \left(\sum_{i=1}^{N_I} Rem_{A(i)} + \sum_{j=1}^{N_J} Rem_{B(j)} \right) + C_{DSO} \cdot F_{match} \tag{2}$$

$$Rem_{A(i)} = \begin{cases} C_{A(i)} & if \quad t_{start(i)} \neq t_{new(i)} \\ 0 & \text{otherwise} \end{cases}$$

$$Rem_{B(j)} = C_{B(i)} \cdot \sum_{t=1}^{N_T} \left| B_{base(j,t)} - B_{flex(j,t)} \right|$$

$$F_{match} = \sum_{t=1}^{N_T} \left| F_{agg(t)} - F_{DSO(t)} \right|$$

where the first term of Eq. (2) corresponds to the monetary compensation paid for shifting device *i* (a flat payment $C_{A(i)}$ in EUR is considered despite how many periods the device is shifted); the second term corresponds to the remuneration given for the modification of the baseline profile of devices type B (where $C_{B(j)}$ is a compensation paid in EUR/kWh modification); and the third term corresponds to a penalty, C_{DSO} in EUR/kWh, paid for the mismatch between the flexibility procured by the DSO ($F_{DSO(t)}$) and the flexibility provided by the aggregator ($F_{agg(t)}$) in each period *t*.

3.C) Scenario overview

Track 1

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 7**). With these profiles, we generate agent data using a randomized function with uniform distribution, 20% around the standard profiles. **Figure 7** 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 [2] with a factor of b_CHP=0.1 EUR/kWh. Concerning the bounding tariffs of the LEM, a flat feed-in tariff of C^F=0.095 for all *t* has been set considering the Portuguese scenario, whereas the aggregator tariff C^{AGG} has been set considering an aggregator fee of 0.15 EUR$/$kWh plus the average wholesale price through the day (which in average was 0.05 EUR$/$kWh in the MIBEL market during the week of 5-9/08/2019).

[2] Open data online at http://sites.ieee.org/pes-iss/data-sets/ and https://fernandolezama.github.io/publication

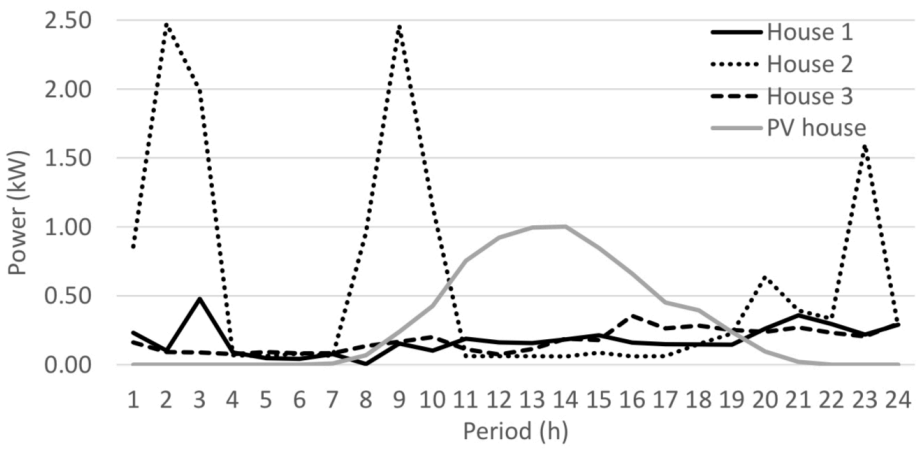

Figure 7 Profiles used in the case study. Ranges of power (in kW): house 1 [0.18-0.48], house 2 [0.06-2.50], house 3 [0.07-0.36], PV (house) [0-1].

Track 2

The case study considers houses equipped with appliances belonging to one of the categories aforementioned. Specifically, we assume houses with the following main appliances for energy consumption: A) shifting devices: dishwashers, washing machines and tumble dryers; B) real-time devices: lighting devices, televisions, and computers. A brief description and the characteristic consumption profile are summarized below:

- Washing Machine: present in the majority of every house in Europe, it is normal its integration in this case study. About 80-90% of the energy used by a washing machine is used to heat the water unless the washing machine is specifically connected to a hot water connection.
- Tumble Dryer: the abundance of tumble dryers is lower in European households compared to washing machines. Tumble dryers use a considerable amount of energy due to the need for a spin to produce heat.
- Dishwasher: The ratios of these devices is increasing in all European countries since the most modern dishwashers are water and energy efficient.
- LED lighting system: these devices are an energy efficient choice for lighting, overshadowing compact fluorescent lamp (CFL), concerning energy efficiency and longevity, at a similar cost.
- Television: TV displays or computer monitors consumption depends on its operating characteristics, namely on the display technology used, manufacturer and build quality, screen size, images showed (e.g., fixed versus moving images), screen brightness and power-saving settings. Despite their low consumption, a considerable number of aggregated TVs might provide some degree of flexibility that the aggregator can use.
- Laptop: generally used less power than desktop computers due to their design to run on battery power. The power consumption of a laptop depends on the size of the screen, varying from 20 watts up to 100 watts when the battery is running out.
- Air conditioning system: used to cool or heat a particular location, the consumption of an Air conditioning system vary in size and power.

The typical consumption profiles of above devices are shown in the figure below:

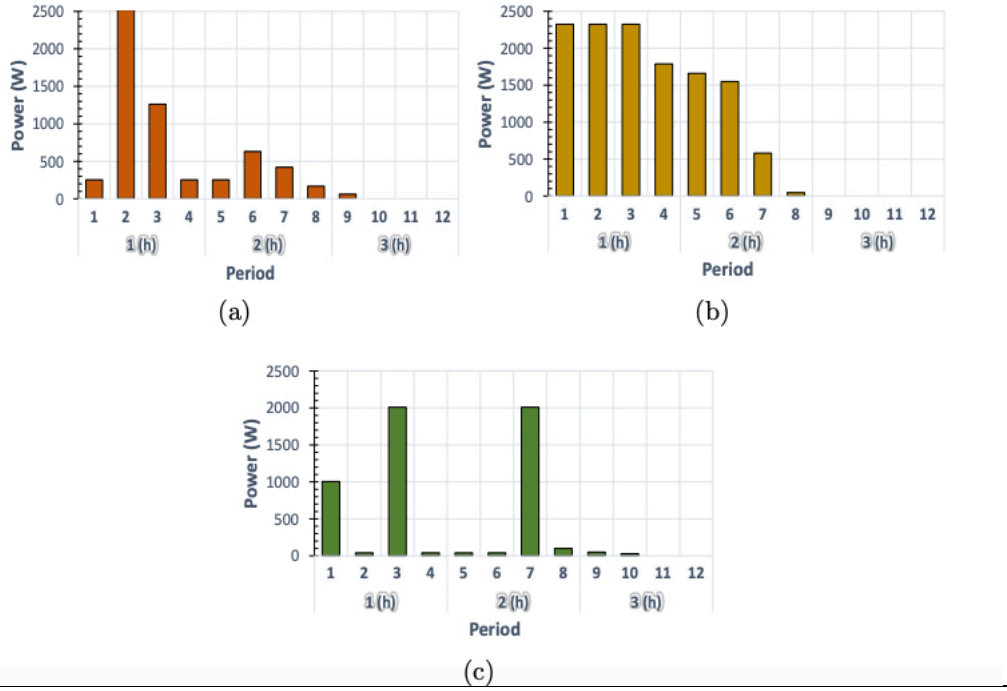

Figure 8 Typical consumption profiles of appliances with shifting capabilities (each 15 mins). (a) Washing Machine (9 periods=2.25 hrs) (b) Tumble Dryer (8 periods=2 hrs). (c) Dishwasher (10 periods=2.5 hrs).

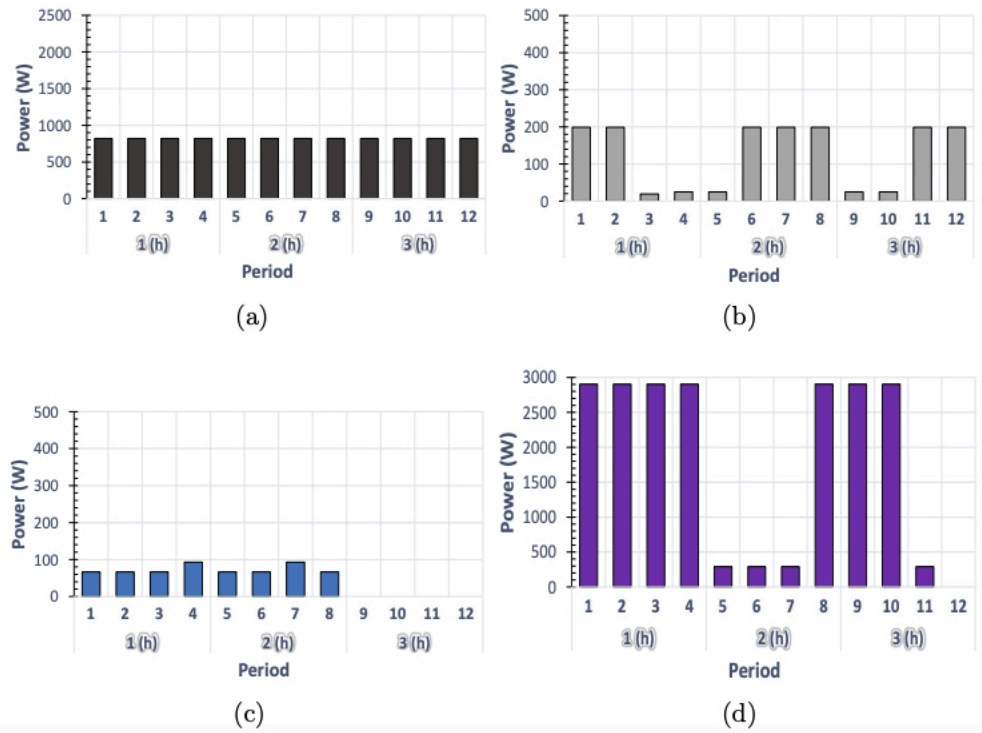

Figure 9 Typical consumption profiles of appliances with real-time reduction capabilities (each 15 mins). (a) Lighting system (12 periods=3 hrs). (b) Television (12 periods=3 hrs). (c) Laptop (8 periods=2 hrs). (d) Air conditioning (11 periods= 2.75 hours).

4. Guidelines for participants

These instructions include as example the metaheuristic differential evolution (DE) [4] 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) *main.m* - Master function/script

 # main.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.

A.0

main.m

```
……
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Select testbed
Select_testbed=1;
%Testbed 1: Optimal bidding in local energy markets
%Testbed 2: Flexibility management of home appliances to support DSO requests
```

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 - # main.m - Loading the testbed and case study

 # main– 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: Optimal bidding in local energy markets
%Testbed 2: Flexibility management of home appliances to support DSO requests
%% 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 track. 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 track.

```
%% 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

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 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.**
***10,000 evaluations per trial are allowed for track 1 and 100,000 for track 2.**

4.B) Fitness function evaluation

***# fitnessFun_bilevel.m and #fitnessFun_flexprob.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_bilevel.m" for track 1 and "#fitnessFun_flexprob.m" for track 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*** (fitnessFun_biLevel.m" and "fitnessFun_flexProb.m") evaluates all the population (individuals) at once. A maximum number of 10,000 function evaluations for track 1, and 100,000 function evaluations for track 2 are allowed in the competition. 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 Track 1	Max. iterations Track 2
1	10,000	100,000
5	2,000	20,000
20	500	5000
50	200	2000
100	100	1000
1000	10	100

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_XXX_track(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

Important note: Tracks evaluation are independent. A winner per each track will be announced.

A ranking index will be calculated using the 20 final solutions (one for each trial) provided by each participant in each track (T1 and T2). With these solutions, the organizers will calculate the ranking index (RI_{user}) for each participant a based on the average fitness of solutions in track 1 and track 2. The values $RI_{user(a)\_T1}$ and $RI_{user(a)\_T2}$ will then be normalized and a final ranking will be produced.

$$RI_{user(a)\_T1} = \frac{1}{N_{trials}} \cdot \left[\sum_{i=1}^{N_{trials}} \left(Fit_a(\vec{X}_{i_{T1}}) \right) \right] \quad (3)$$

where $Fit_a(\vec{X}_i\_T1)$ is a function that returns the fitness value of the solution found in trial i (i.e., $\vec{X}_i$) by participant a (See **Sect. 3.B**).

$$RI_{user(a)\_T2} = \frac{1}{N_{trials}} \cdot \left[\sum_{i=1}^{N\_trials} \left(Fit_a(\vec{X}_{i_{T2}}) \right) \right] \quad (4)$$

where $Fit_a(\vec{X}_i\_T2)$ is a function that returns the fitness value of the solution found in trial i (i.e., $\vec{X}_i$) by participant a (See **Sect. 3.B**).

Therefore, the winner of each track will be the one that gets the minimum value of RI_{user} (minimization problems). The participants must consider this criterion while selecting the best search strategy in their algorithms.

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:

trackX_*AlgorithmName_ParticipantName*.zip
(e.g., track2_*DE_Lezama*.zip).

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

by 31th May 2021

Bibliography

[1] E. Mengelkamp, J. Gärttner, and C. Weinhardt, “Intelligent agent strategies for residential customers in local electricity markets,” 2018, doi: 10.1145/3208903.3208907.

[2] F. Lezama, J. Soares, and Z. Vale, “Optimal Bidding in Local Energy Markets using Evolutionary Computation,” 2019.

[3] F. Lezama, J. Soares, B. Canizes, and Z. Vale, “Flexibility management model of home appliances to support DSO requests in smart grids,” *Sustain. Cities Soc.*, vol. 55, p. 102048, Apr. 2020, doi: 10.1016/j.scs.2020.102048.

[4] 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,” 2017.