Guidelines for WCCI/GECCO 2026 Competition Evolutionary Computation in the Energy Domain: Fairness-aware Pricing in Energy Communities

Fernando Lezama, José Almeida, João Soares, Filipe Sousa, Zita Vale
School of engineering (ISEP), Polytechnic of Porto, Porto, Portugal
flz@isep.ipp.pt, jorga@isep.ipp.pt, jan@isep.ipp.pt, bmc@isep.ipp.pt, ffeso@isep.ipp.pt, zav@isep.ipp.pt

----

Cite this document as:
Lezama, F., Almeida, J., Soares, J., Sousa, F., Vale, Z. (2026, January). *Guidelines for WCCI/GECCO 2026 Competition Evolutionary Computation in the Energy Domain: Fairness-aware Pricing in Energy Communities*.

IEEE CIS Task Force on 'Computational Intelligence in the Energy Domain (ci4energy), part of IEEE CIS Intelligent Systems Applications TC (https://www.gecad.isep.ipp.pt/ci4energy/)

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

IEEE PES Working Group on Modern Heuristic Optimization (https://site.ieee.org/psace-mho/)

January 2026

Table of contents

1. Introduction

Following the success of the previous editions at PES GM (2017,2021), GECCO (since 2018 to 2025), SSCI 2025, and WCCI and CEC (since 2017 to 2025)[1], we are launching a new edition of our algorithm competition at major conferences in the field of Computational Intelligence. This **2026 competition** proposes one track in the energy domain:

Track 1) ***Fairness-aware Pricing Optimization in Local Energy Communities (LECs).***

This track focuses on optimizing local electricity prices to achieve a fair and equitable distribution of costs and benefits among community members. Participants are required to develop metaheuristic algorithms capable of determining an optimal price profile that maximizes fairness according to a "FairMix metric", a composite fairness function that integrates three complementary factors:

- **Equity** (1−J), based on Jain's index, rewarding uniform positive outcomes across participants.
- **Balance** (CV), capturing the coefficient of variation among those who benefit to ensure equity within the "winners."
- **Loss moderation** (L_norm), which penalizes large relative losses among participants with negative outcomes.

The competition framework follows the same structure as in previous editions, allowing former participants to easily adapt their algorithms. The number of function evaluations is also included in the ranking index, together with the final fitness value, rewarding algorithms that achieve fairer outcomes efficiently.

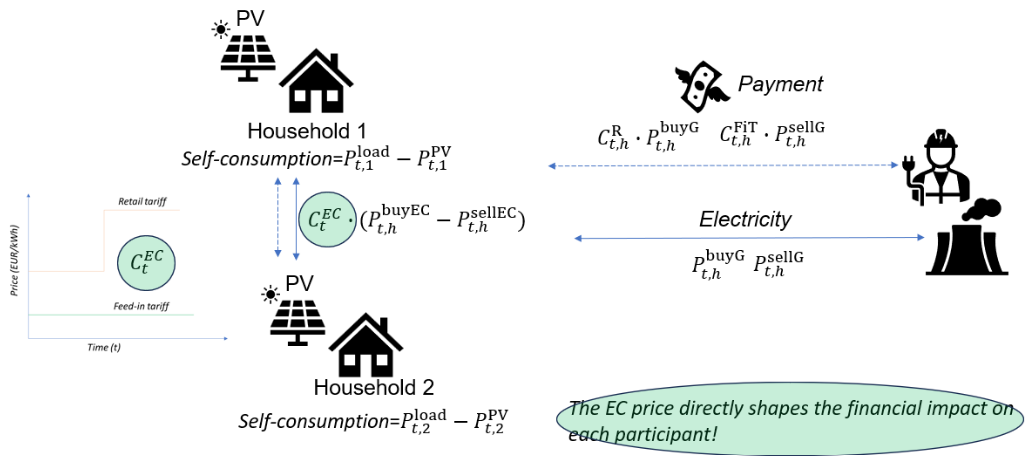
Tip: The most important section for a quick participation in this competition is **Section 4**, which explains how to implement your heuristic and treat the problem as a black-box optimization task. The earlier sections (Sections 2–3) describe the background and mathematical formulation of the problem.

----

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

2. General Description of the Fair Pricing Problem in Local Energy Communities

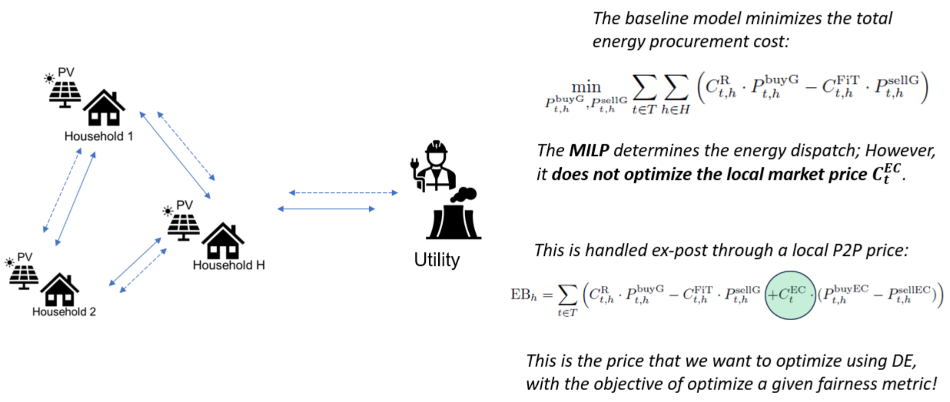
Local Energy Communities (LECs) enable coordinated management of distributed resources, fostering self-consumption, sharing of surplus photovoltaic energy, and active peer-to-peer (P2P) exchanges (see Figure 1). However, as demonstrated in recent studies, the economic outcomes resulting from these exchanges can be uneven, even when the underlying operational scheduling is technically optimal. Mixed-integer linear programming (MILP) models typically determine energy flows that minimize global system costs, but they do not guarantee that the resulting financial allocation among community members is fair or socially acceptable. This limitation has been highlighted in scalable LEC management models such as the three-stage framework of Lezama et al. [1], where pricing is applied ex-post and may not reflect the diversity of roles or contributions within the community. Similarly, Gonçalves et al. [2] show that pricing choices strongly influence fairness, especially when the community integrates storage, variable PV profiles, and heterogeneous user behaviors.


PV
Household 1
Self-consumption=$P^{\text{load}}_{t,1} - P^{\text{PV}}_{t,1}$
$C^{EC}_t \cdot (P^{\text{buyEC}}_{t,h} - P^{\text{sellEC}}_{t,h})$
Retail tariff
C^{EC}_t
Feed-in tariff
Price (EUR/kWh)
Time (t)
PV
Household 2
Self-consumption=$P^{\text{load}}_{t,2} - P^{\text{PV}}_{t,2}$
Payment
$C^{\text{R}}_{t,h} \cdot P^{\text{buyG}}_{t,h}$ $C^{\text{FiT}}_{t,h} \cdot P^{\text{sellG}}_{t,h}$
Electricity
$P^{\text{buyG}}_{t,h}$ $P^{\text{sellG}}_{t,h}$

The EC price directly shapes the financial impact on each participant!

Figure 1 The energy sharing concept

In this competition, the fair pricing problem is framed as an **ex-post optimization challenge**: after the MILP has fixed all energy transactions and ensured technical feasibility, participants must compute a vector of local prices that leads to a fair distribution of costs among households. This strategy follows the approach introduced in Lezama et al. [3], where evolutionary algorithms optimize fairness metrics without altering the operational dispatch. The dispatch is therefore identical across all algorithms; only the economic redistribution changes. This separation mimics real-world LECs, where pricing is often governed by cooperative agreements or market rules rather than physical constraints (See Figure 2).



The baseline model minimizes the total energy procurement cost:

$$\min_{P^{\text{buyG}}_{t,h},\, P^{\text{sellG}}_{t,h}} \sum_{t\in T}\sum_{h\in H}\left(C^{\text{R}}_{t,h}\cdot P^{\text{buyG}}_{t,h} - C^{\text{FiT}}_{t,h}\cdot P^{\text{sellG}}_{t,h}\right)$$

*The **MILP** determines the energy dispatch; However, it **does not optimize the local market price C^{EC}_t**.*

This is handled ex-post through a local P2P price:

$$\text{EB}_h = \sum_{t\in T}\left(C^{\text{R}}_{t,h}\cdot P^{\text{buyG}}_{t,h} - C^{\text{FiT}}_{t,h}\cdot P^{\text{sellG}}_{t,h} + C^{\text{EC}}_t\cdot\left(P^{\text{buyEC}}_{t,h} - P^{\text{sellEC}}_{t,h}\right)\right)$$

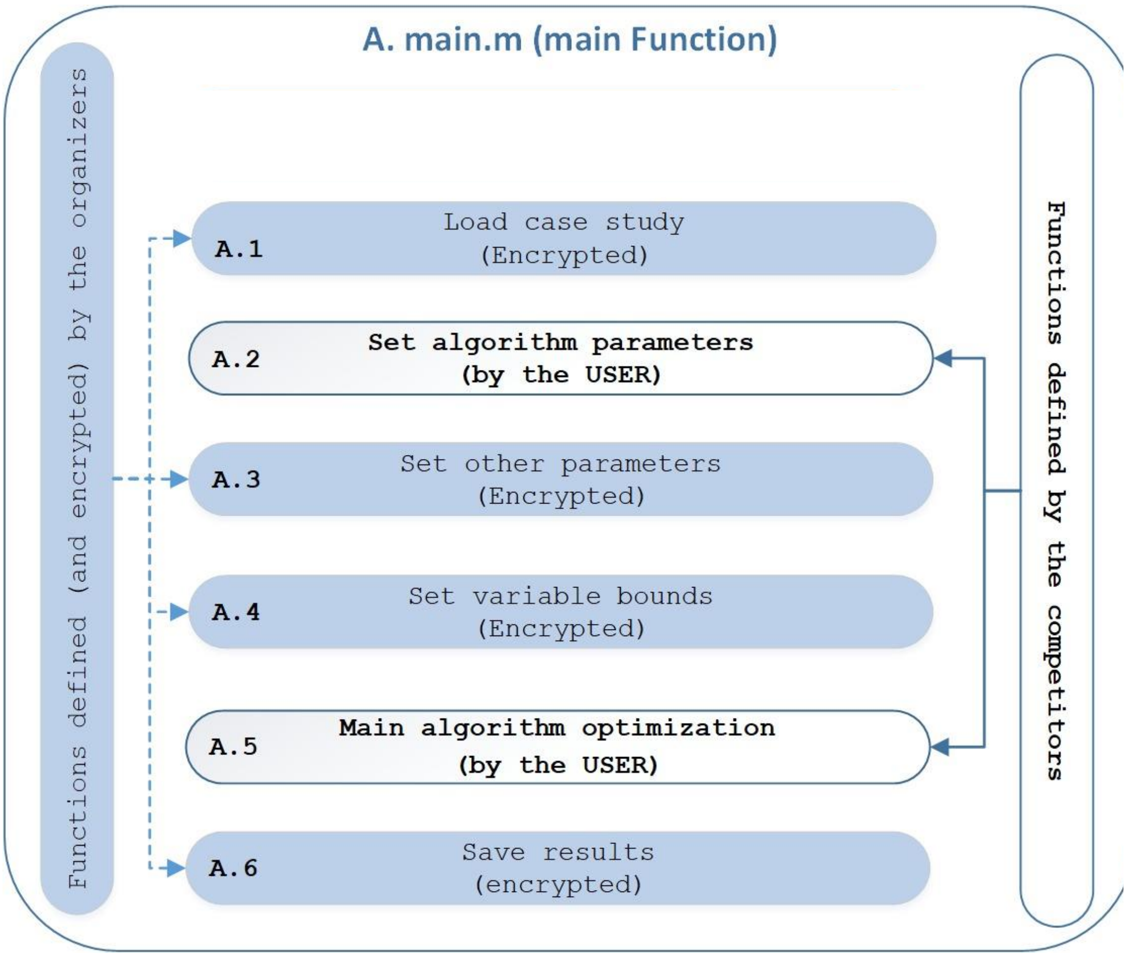
This is the price that we want to optimize using DE, with the objective of optimize a given fairness metric!

Figure 2 MILP Model for Energy Exchanges and the ex-post p2p price problem

Participants must therefore design metaheuristic algorithms capable of exploring a high-dimensional continuous search space (one price per time step) and producing a price profile that improves fairness without modifying the physical dispatch or increasing system costs. The problem is strongly nonlinear, multi-perspective, and sensitive to community heterogeneity, making it an ideal benchmark for evolutionary computation research.

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.**



A. main.m (main Function)
Functions defined (and encrypted) by the organizers
A.1 Load case study (Encrypted)
A.2 Set algorithm parameters (by the USER)
A.3 Set other parameters (Encrypted)
A.4 Set variable bounds (Encrypted)
A.5 Main algorithm optimization (by the USER)
A.6 Save results (encrypted)
Functions defined by the competitors

Figure 3 General framework of the simulation platform

The simulation platform has been implemented in MATLAB© 2021 64-bit and consists of different scripts with specific roles 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.**

A maximum of **5,000 function evaluations** is allowed for each run of the optimizer. Importantly, the number of function evaluations is included in the competition's ranking index, meaning that participants benefit from developing algorithms that converge using fewer evaluations. One function evaluation corresponds to a single evaluation of the fitness function, which computes the mixed fairness metric for a candidate price vector. This is distinct from algorithm iterations, because an iteration may generate and evaluate several solutions depending on the metaheuristic design.

3.A) Encoding of the individual

A fundamental component of any population-based metaheuristic is the encoding of candidate solutions. In this competition, each solution represents a complete **vector of local electricity prices** for the scheduling horizon of the LEC. Since the operational dispatch is already fixed by the MILP model, the optimizer only manipulates the **price profile**, which directly influences the fairness metric.

Each solution is encoded as a **continuous-valued vector**: $\vec{p} = [p_1, p_2, \dots, p_T]$

where each element p_t corresponds to the local energy price at time step t, and T is the number of periods (typically $T = 96$ for a 15-minute resolution over 24 hours).

All decision variables are continuous values restricted by problem-specific bounds:

$$C_t^{\mathrm{FiT}} \leq p_t \leq C_t^{\mathrm{grid}}$$

where C_t^{FiT} is the feed-in tariff (lower bound) and C_t^{grid} is the retail/grid price (upper bound). These bounds ensure realistic price formation and reflect typical regulatory and market constraints.

Heuristic or problem-specific initialization strategies are not allowed, since submitted algorithms will also be evaluated on unseen LEC instances with different sizes and characteristics. This guarantees generality and fairness across participants.

Figure 4 illustrates the solution representation. Each individual in the population is a one-dimensional vector of length $T = 96$ for a 15-minute resolution over 24 hours, with no binary or integer components. Because the dimensionality scales linearly with the number of time periods, the encoding remains simple and computationally efficient, ensuring broad compatibility across evolutionary approaches such as DE, PSO, GA, CMA-ES, and others.

$$\begin{aligned} \overrightarrow{x_1} &= [p_{1,1} \quad p_{1,2} \quad \cdots \quad p_{1,T}] \\ \overrightarrow{x_2} &= [p_{2,1} \quad p_{2,2} \quad \cdots \quad p_{2,T}] \\ &\vdots \\ \overrightarrow{x_{NP}} &= [p_{NP,1} \quad p_{NP,2} \cdots p_{NP,T}] \end{aligned}$$

Variable name	Type	Bounds	Dimension
LEC price per period	Continuous	[FiT, Grid]	96

Figure 4 Solution representation.

The metaheuristic's initial solutions should be created randomly between the maximum and minimum values set for each variable. Each solution is composed of a group of sequentially repeated variables per each of the 96 periods if 15 minutes (t=1, t=2, ..., t=96). The metaheuristic evaluates each individual by computing the **mixed fairness metric** based on the resulting household energy bills. A solution is therefore considered better when its price vector yields a more equitable cost distribution across community members, according to the competition's fairness metric.

3.B) Fitness function

A maximum of **5,000 fitness evaluations** is allowed per run in this competition. As in previous editions, the number of fitness evaluations is explicitly incorporated into the ranking index. The problem can be treated as a **pure black-box optimization task**: each time a candidate price vector is evaluated, the framework returns a scalar fairness score, as illustrated in **Figure** 5.

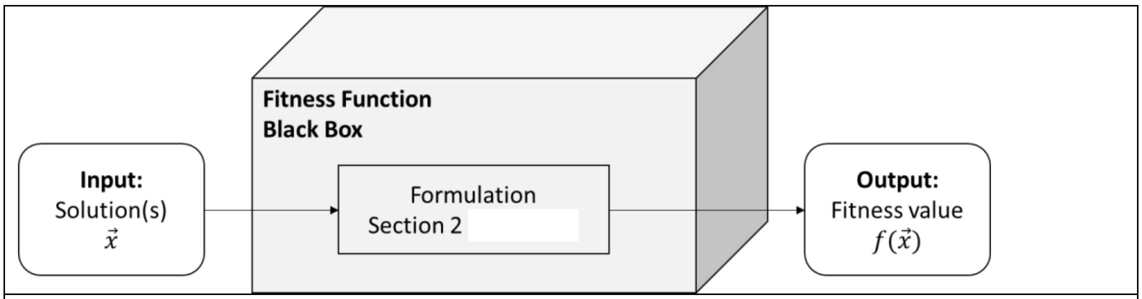


Figure 5 Fitness function as black box.

Internally, the fitness function computes household energy bills based on the fixed MILP dispatch and evaluates the fairness of the proposed price vector using the **FairMix metric (see** [3] **for more details)**. This metric synthesizes three complementary fairness dimensions, capturing both economic balance and perceived equity among community members.

FairMix Fairness Metric

Given a price vector $\vec{p}$, the resulting household bills $B_h(\vec{p})$ are used to compute three components:

1. **Jain's Fairness Index**, which increases when a larger number of participants experience positive gains distributed uniformly:

$$J = \frac{(\sum_h G_h)^2}{n \cdot \sum_h G_h} \quad (1)$$

where $G_h = \max(B_h^{BAU} - B_h, 0)$represents each household's economic gain relative to its Business-as-Usual bill.

2. **Coefficient of Variation (CV)** of gains among users who benefit:

$$CV = \frac{\sigma_G}{\mu_G} \quad (2)$$

where a lower value indicates balanced benefits among winners.

3. Normalized Loss Indicator **for households experiencing losses:**

$$L_{norm} = \frac{\mu_L}{\mu_{BAU}} \quad (3)$$

where $\mu_L$is the mean loss among losing participants and $\mu_{BAU}$is the community's average BAU bill. Lower values correspond to smaller relative burdens placed on losing members.

These components are combined into the **FairMix** objective:

$$\text{FairMix} = \alpha(1 - J) + \beta CV + \gamma L_{norm} \quad (4)$$

where $\alpha, \beta, \gamma \geq 0$weight the importance of each fairness dimension. The default values provided in the platform are $\alpha = 0.4, \beta = 0.4, \gamma = 0.2$ for this application.

The metaheuristic algorithm must **minimize** this score. Lower FairMix values indicate better overall fairness, meaning gains are widespread and uniform, losses are small, and the distribution is considered socially balanced.

Operational Summary

- Participants do *not* need to compute any of these values manually.
- The MATLAB framework evaluates each candidate solution and returns the scalar FairMix score.
- Each function evaluation corresponds to one full internal computation of household bills and fairness metrics.
- The goal is to minimize FairMix while requiring as few function evaluations as possible.

This fitness design ensures that participants focus on developing effective and generalizable optimization algorithms, while the fairness logic and economic modeling remain fully encapsulated within the competition's black-box evaluation framework.

3.C) Scenario overview

The competition evaluates algorithms on a **Local Energy Community (LEC)** composed of distributed photovoltaic (PV) generation, battery storage, and heterogeneous consumption profiles. The base case study consists of **ten active households**, operating over a **single day with 96 time steps** (15-minute resolution). All profiles derive from real measurements collected at **GECAD Building N, Porto, Portugal**, and reflect realistic variability in both generation and demand [2], [3], [4].

Community Composition

The base configuration includes:

- **Six prosumers** equipped with rooftop PV systems.
- **Four consumers** without local generation.
- **Optional battery storage systems**, included in some instances and managed centrally in the MILP.

The PV households are designed to mimic diverse socio-economic and installation conditions:

- **Households 1–2:** low-capacity PV, peak around **4 kW**.
- **Households 3–4:** medium PV systems, peaks around **8 kW**.
- **Households 5–6:** high-capacity PV, peaks near **12 kW**.

Households **7 to 10** operate purely as consumers, representing users without generation capability.

Energy exchanges occur internally between community members and externally through the utility supplier. Figure 1 illustrates the basic energy-sharing structure used in this scenario.

Operational Baseline

The **Business-as-Usual (BAU)** benchmark assumes that the local price is the **mid-market price**, i.e., the price between the grid tariff and the feed-in tariff.
Each household purchases imported electricity at the grid tariff and sells excess PV at a fixed **Feed-in Tariff (FiT)**. This BAU bill is used to compute individual gains and losses, which serve as the foundation for the fairness metric.

MILP-Based Dispatch

Before pricing optimization, a **Mixed-Integer Linear Programming** (MILP) model computes:

- Internal P2P exchanges
- Imports and exports
- Storage charge–discharge decisions
- Energy flows satisfying technical constraints

This dispatch is **identical for all algorithms** and remains fixed throughout the pricing optimization phase. Participants optimize *only* the price vector, not the physical operation.

Extended Instance

To evaluate scalability and algorithmic robustness, the framework also includes a **100-household testbed**. This extended community preserves the same structural logic: heterogeneous PV penetration, consumer-only households, realistic profiles, and the same BAU and MILP setup.

Participants' algorithms must generalize across both community sizes without requiring manual tuning.

Both instances provide a challenging yet well-structured environment for fairness-aware optimization, combining high-resolution temporal data, heterogeneous user behavior, and nonlinear economic interactions.

4. Guidelines for participants

These instructions include as example the metaheuristic hybrid-adaptive differential evolution (HyDE) [5] implemented and adapted to the present fair-pricing framework. Participants must follow the structure described below to ensure compatibility with the supplied dataset and evaluation script.

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.

main.m

```
……
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GECAD CEC/GECCO 2026 Competition: Evolutionary Computation in the Energy Domain: 2026
Edition of the fair-price scheduling
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Load Data base                                                                A.1
[caseStudyData, DB_name]=callDatabase();

Select_algorithm=1;
%1: HyDE algorithm (test algorithm)
%2: Your algorithm
%%%%%%%%%%%%%%%%%% FEs %%%%%%%%%%%%%%%%%%
No_evals=5000; % These are the number of function evaluations per trial. The lower the
value, more points for the algorithm!
noRuns=20; %this can be changed but final results should be based on 20 trials

switch Select_algorithm
       case 1                                                                    A.2
          addpath('HyDE')
          algorithm='HyDE-Algorithm'; %'Participants can include their algorithm here'
          DEparameters %Function defined by the participant
          No_solutions=deParameters.I_NP; %Some algorithms are limited to 1 solution
       case 2
          %%%%%%%%%%%%%%%%%%%%%%% Or here - Your algorithm %%%%%%%%%%%%%%%%%%%%%%%%%%%
    otherwise
          fprintf(1,'No algorithm selected\n');
end

%% Set other parameters                                                          A.3
otherParameters =setOtherParameters(caseStudyData,No_solutions);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Set lower/upper bounds of variables                                           A.4
[lowerBounds,upperBounds] = setVariablesBounds(caseStudyData,otherParameters);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Call the MH for optimizationclear
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                                                      A.5
        case 1
            [ResDB(iRuns).Fit_and_p, ...
            ResDB(iRuns).sol, ...
            ResDB(iRuns).fitVector]= ...
            HyDE(deParameters,caseStudyData,otherParameters,lowerB,upperB,No_evals);
        case 2
          %%%%%%%%%%%%%%%%%%%%%%% Your algorithm %%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% Save the results and stats                                            A.6
        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.1 - *# main.m - Loading the case study*

# main– This is the main framework file which will load the caseStudyData struct (callDatabase.p – encrypted) with all the relevant dataset information. Participants do not need to worry about the content of the case study and loading the files.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% Load Data base
[caseStudyData, DB_name]=callDatabase();
```

A.2 - *#No_evals, No_runs, and #DEparameters.m - Set parameters of the metaheuristic and number of evaluations*

# No_evals – The authors should modify this parameter considering that a lower value will guarantee a better algorithm classification.

# NoRuns – This is the number of trial. Can be reduced for testing new algorithms, but 20 trails will be considered for ranking the participants.

# 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.

```
No_evals=5000; % These are the number of function evaluations per trial. The lower the
value, more points for the algorithm!
noRuns=20; %this can be changed but final results should be based on 20 trials


%%%%%%%%%%%%%% DE.parameters.m file %%%%%%%%%%
% Size of the population in DE
deParameters.I_NP= 10; % Size of the population in DE
deParameters.F_weight= 0.3; %Mutation factor
deParameters.F_CR= 0.5; %Recombination constant
deParameters.I_itermax= 500; % 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);
…..
[ResDB(iRuns).Fit_and_p, ...
                    ResDB(iRuns).sol, ...
                    ResDB(iRuns).fitVector]=                                       ...
HyDE(deParameters,caseStudyData,otherParameters,lowerB,upperB,No_evals);
```

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.

A.5 - *#HyDE.m* - *Algorithm proposed by the competitor*

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

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

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
5. **No_evals:** number of function evaluations

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).noEvals:** array of size 1 with the number of function evaluations needed to achieve the reported result

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. 1_time_T1.txt: Computing time spent for all optimization trials (benchmark_Time.txt)

	timeSpent (s)
Run1	
Run2	
Run3	
…	
Run20	

Table 2. 1_fitness_T1: Individual benchmark of the fitness in each iteration (benchmark_Fitness.txt)

	fitVector_1	fitVector_2	…	FitVector_itermax
Run1				
Run2				
Run3				
…				
Run20				

Table 3. 1_nfe_T1: Number of function evaluations at each trial (this should be the same for each trial, and should be as lower as possible to get more points for your algorithm. It it defined by `No_evals=x in main script`)

	NFE
Run1	
Run2	
Run3	
…	
Run20	

Table 4. 1_benchmark_Summary_T1: Summary statistics or the trials (benchmark_Summary.txt)

Provisional RankingIndex	RankingIndex_1	RankingIndex_2	StdOF	StdEvals	AvgTime	ValidationCode
RankingIndex1+ RankingIndex2	Related to fitness value	Related to number of function evaluations	Standard deviation of fitness	Standard deviation of NFE		

In addition, this function should automatically generate the file "`best_solutions.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. 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 "`1_send2Organizers_T1.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 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.**
***A maximum of 5,000 evaluations per trial are allowed. Again, in this year's competition the number of fitness evaluations are considered in the calculation of the ranking index.**

4.B) Fitness function evaluation

***# fitnessFun_riskERM.m and #fitnessFun_transEP* (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, no_evals);
```

The function receives as input:

1. **fnc**: string with the fitness function m file name: "fitnessFun_fairLECprice.m".
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**).
5. `no_evals`: **The defined number of function evaluations. Remember, the lower, the better.**

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.

The ***#fitnessFun*** evaluates all the population (individuals) at once. A maximum number of 5,000 function evaluations is set for this competition. The table below helps the participant to have an idea of the maximum number of iterations and population. They can set their number of function evaluations without surpassing the shown limits. So, take it account when designing your algorithm:

Table 7. Algorithm population/iterations limits

Size of the population	Max. iterations Track 1
1	5,000
5	1,000
20	250
50	100
100	50
1000	5

5. Evaluation guidelines

A ranking index is computed using the 20 final solutions provided by each participant. For each participant a, two quantities are evaluated across the 20 trials:

1. The average fitness value obtained using the **FairMix** metric.

2. The average number of function evaluations required to obtain the reported solution.

These two values are normalized across all participants and summed to produce the final ranking index. Lower scores indicate better performance. Winsorization[2] is applied during normalization to reduce the influence of extreme outliers.

Let

- $\text{Fit}_a(\vec{X}_i)$ denote the FairMix fitness value obtained by participant $a$in trial i,
- $\text{Nevals}_a(\vec{X}_i)$ denote the number of fitness evaluations used in trial i, and
- $N_{\text{trials}} = 20.$

The ranking components are:

$$RI_{a,1} = \frac{1}{N_{\text{trials}}} \sum_{i=1}^{N_{\text{trials}}} \text{Fit}_a(\vec{X}_i) \qquad (5)$$

$$RI_{a,2} = \frac{1}{N_{\text{trials}}} \sum_{i=1}^{N_{\text{trials}}} \text{Nevals}_a(\vec{X}_i) \qquad (6)$$

Both values are then normalized across all participants using min-max scaling with winsorization:

$$RI_a^{\text{total}} = N(RI_{a,1}) + N(RI_{a,2}) \qquad (7)$$

The winner is the participant achieving the **minimum** value of

$$RI_a^{\text{total}}.$$

This approach ensures a balanced evaluation that rewards both **fairness-oriented pricing performance** (via the FairMix metric) and **computational efficiency** (via the number of evaluations). Participants must take both aspects into account when designing their metaheuristic strategies.

[2] Winsorization is the practice of substituting statistical data's extreme values to reduce the impact of outliers on computations or conclusions drawn from the data.

6. Material to be submitted to the organizers

For the validation of the results, the 4 benchmark text files and the "`send2Organizers_T1.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 (Note: this evaluation could consider the modification of the case study and number and the encoding of variables, so that algorithms should be designed generally enough to handle different case studies of the same problem). 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:

WCCI_GECCO2026_*AlgorithmName*_*ParticipantName*.zip
(e.g., WCCI_GECCO2026_*DE*_*Lezama*.zip).

The zipped folder must be summited to flz@isep.ipp.pt, jorga@isep.ipp.pt, jan@isep.ipp.pt

by 1st June 2026 (anywhere on Earth)

Appendix: Related Literature on the Competition Topics

A systematic review of explainability in computational intelligence for optimization [6]

Review on fairness in local energy systems [7]

Modern distribution system expansion planning considering new market designs: Review and future directions [8]

Bibliography

[1] F. Lezama *et al.*, "A scalable three-stage model for local energy community management and pricing," *CSEE J. Power Energy Syst.*, 2025.

[2] C. Goncalves, F. Lezama, and Z. Vale, "Towards Fair Energy Communities: Integrating Storage, Sharing and Pricing Strategies," *IFAC-Pap.*, vol. 58, no. 13, pp. 320–325, 2024, doi: 10.1016/j.ifacol.2024.07.502.

[3] F. Lezama, F. Doria, J. Almeida, J. Soares, and Z. Vale, "Fair Pricing Optimization in Energy Communities with Differential Evolution," in *Proceedings of the International Workshop on AI Systems for the Environment (AISE-2025*, 2025, pp. 61–70. [Online]. Available: https://eprints.soton.ac.uk/505165/1/Proceedings_of_the_International_Workshop_on_AI_Systems_for_the_Environment_AISE-2025_.pdf

[4] C. Goncalves, F. Lezama, and Z. Vale, "Analyzing the Impact of Pricing Strategies on Economic Fairness in Energy Communities," in *2024 IEEE Power & Energy Society General Meeting (PESGM)*, Seattle, WA, USA: IEEE, July 2024, pp. 1–5. doi: 10.1109/PESGM51994.2024.10689031.

[5] F. Lezama, J. Soares, R. Faia, T. Pinto, and Z. Vale, "A New Hybrid-Adaptive Differential Evolution for a Smart Grid Application Under Uncertainty," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, Rio de Janeiro: IEEE, July 2018, pp. 1–8. doi: 10.1109/CEC.2018.8477808.

[6] J. Almeida, J. Soares, F. Lezama, S. Limmer, T. Rodemann, and Z. Vale, "A systematic review of explainability in computational intelligence for optimization," *Comput. Sci. Rev.*, vol. 57, p. 100764, Aug. 2025, doi: 10.1016/j.cosrev.2025.100764.

[7] J. Soares *et al.*, "Review on fairness in local energy systems," *Appl. Energy*, vol. 374, p. 123933, Nov. 2024, doi: 10.1016/j.apenergy.2024.123933.

[8] T. D. De Lima, F. Lezama, J. Soares, J. F. Franco, and Z. Vale, "Modern distribution system expansion planning considering new market designs: Review and future directions," *Renew. Sustain. Energy Rev.*, vol. 202, p. 114709, Sept. 2024, doi: 10.1016/j.rser.2024.114709.