Task Force on Modern Heuristic Optimization Test Beds
Working Group on Modern Heuristic Optimization
Intelligent Systems Subcommittee
Analytic Methods in Power Systems Committee

2017 Competition
Evaluating the Performance of Modern Heuristic Optimizers on Smart Grid Operation Problems

Test bed 2:
Optimal scheduling of distributed energy resources

Zita Vale, João Soares
School of engineering (ISEP), Polytechnic of Porto, Porto, Portugal
zav@isep.ipp.pt, joaps@isep.ipp.pt

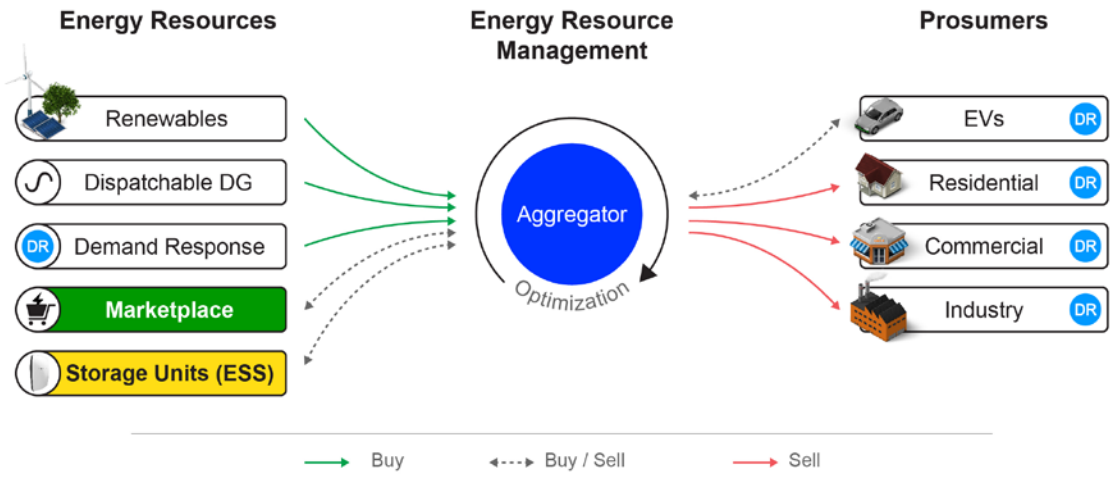
December 2016

Table of contents

Introduction and general description

GECAD – Polytechnic of Porto – proposes the optimization of two large-scale centralized Day-Ahead energy resource scenarios. The aim is to use stochastic optimization (e.g. PSO, GA, SA, ABC, etc.) to mitigate the exponential execution time using traditional mathematical tools. Even with state-of-the-art solvers' technology these scenarios use considerable amount of time to solve. We wish to solve these scenarios fast, reliable and with satisfactory solutions. The use of stochastic optimization may provide interesting answers and further discussion in the community.

The energy aggregator can procure energy needs from several resources and the electricity market and makes revenue from reselling energy to its customers. In addition, it may use its own assets, e.g. storage units, to supply the load demand. The energy aggregator establishes energy contracts with those who seek electricity supply, e.g. residential and industry customers. It is designated here as a bilateral contract, i.e. between the aggregator and the final end-user. In this case, it is assumed that the aggregator establishes a fixed price for fixed loads and EVs charging. The fixed price is set independently for each consumer, based on single-tariffs. The main idea is that the optimization software can perform the energy resource scheduling of the involved resources in the day-ahead context for the 24 hours of the following day. In addition, V2G is also possible.



Overview of the aggregator energy management problem

1.1. Objective function

The envisaged problem is a hard combinatorial Mixed-Integer Non-Linear Programming (MINLP) problem due to high number of continuous, discrete and binary variables and network non-linear equations. The objective of the aggregator is to maximize profits, i.e. income (*In*) minus operation cost (*OC*). This can be rewritten as minimization function *Z*, as show in (1).

$$Minimize\, Z = OC - In \tag{1}$$

The minimum value of Z (hopefully negative) is the profit of the energy aggregator. If *Z* is negative it is expected to have a profit, otherwise *OC* are higher than *In*come. Thus, the profit is *P*=-*Z*, where *P* is the profit. Nevertheless, for the goal in optimization terms is to obtain the minimum value of *Z* in the metaheuristics form.

The aggregator can receive his income (*In*) from four sources, as illustrated in (2): the revenue from the consumers demand; the energy to sell to the electricity market; the revenue from the charging process of storage units and the charging of EVs.

$$In = \sum_{t=1}^{T}\left[\left(\begin{array}{l}\sum_{L=1}^{N_L} P_{Load(L,t)} \cdot MP_{Load(L,t)} + \sum_{M=1}^{N_M} P_{Sell(M,t)} \cdot MP_{Sell(M,t)} + \\ \sum_{E=1}^{N_E} P_{Charge(E,t)} \cdot MP_{Charge(E,t)} + \sum_{V=1}^{N_V} P_{Charge(V,t)} \cdot MP_{Charge(V,t)}\end{array}\right) \times \Delta t\right] \quad (2)$$

The parameters are described by: N_E is the number of ESSs; N_L is the number of loads; N_M is the number of markets; N_V is the number of EVs; $MP_{Load(L,t)}$ is the price of load L in period t (m.u.); $MP_{Sell(M,t)}$ is the price that market M pays in period t (m.u.); $MP_{Charge(E,t)}$ is the price for the charge process of ESS E in period t (m.u.); $MP_{Charge(V,t)}$ is the price for the charge process of EV V in period t (m.u.)

The variables are described by: In is the aggregator income (m.u.); $P_{Charge(E,t,)}$ is the active power charge of ESS E in period t (MW); $P_{Charge(V,t)}$ is the active power charge of EV V in period t (MW); $P_{Load(L,t)}$ is the active power demand of load L in period t (MW); $P_{Sell(M,t)}$ is the active power sale to market M in period t (MW)

Function OC, defined in (3), represents the operation cost of the resources managed by the VPP. It considers the cost with Distributed Generation (DG), external suppliers, discharge of storage and EVs, Demand Response (DR) programs, penalization with non-supplied demand and penalization with DG units' generation curtailment.

$$OC = \sum_{t=1}^{T}\left[\left(\begin{array}{l}\sum_{I=1}^{N_{DG}} P_{DG(I,t)} \cdot c_{DG(I,t)} + \sum_{S=1}^{N_S} P_{Supplier(S,t)} \cdot c_{Supplier(S,t)} + \\ \sum_{L=1}^{N_L} P_{LoadDR(L,t)} \cdot c_{LoadDR(L,t)} + \sum_{M=1}^{N_M} P_{Buy(M,t)} \cdot MP_{Buy(M,t)} + \\ \sum_{V=1}^{N_V} P_{Discharge(V,t)} \cdot c_{Discharge(V,t)} + \sum_{E=1}^{N_E} P_{Discharge(E,t)} \cdot c_{Discharge(E,t)} + \\ \sum_{L=1}^{N_L} P_{NSD(L,t)} \cdot c_{NSD(L,t)} + \sum_{I=1}^{N_{DG}} P_{GCP(I,t)} \cdot c_{GCP(I,t)}\end{array}\right)\right] \quad (3)$$

The indices are represented by: E is an index of ESSs; I is an index of DG units; L is an index of loads; M is an index of market/energy buyer; S is an index of external suppliers; t is an index of time periods; V is an index of EVs;

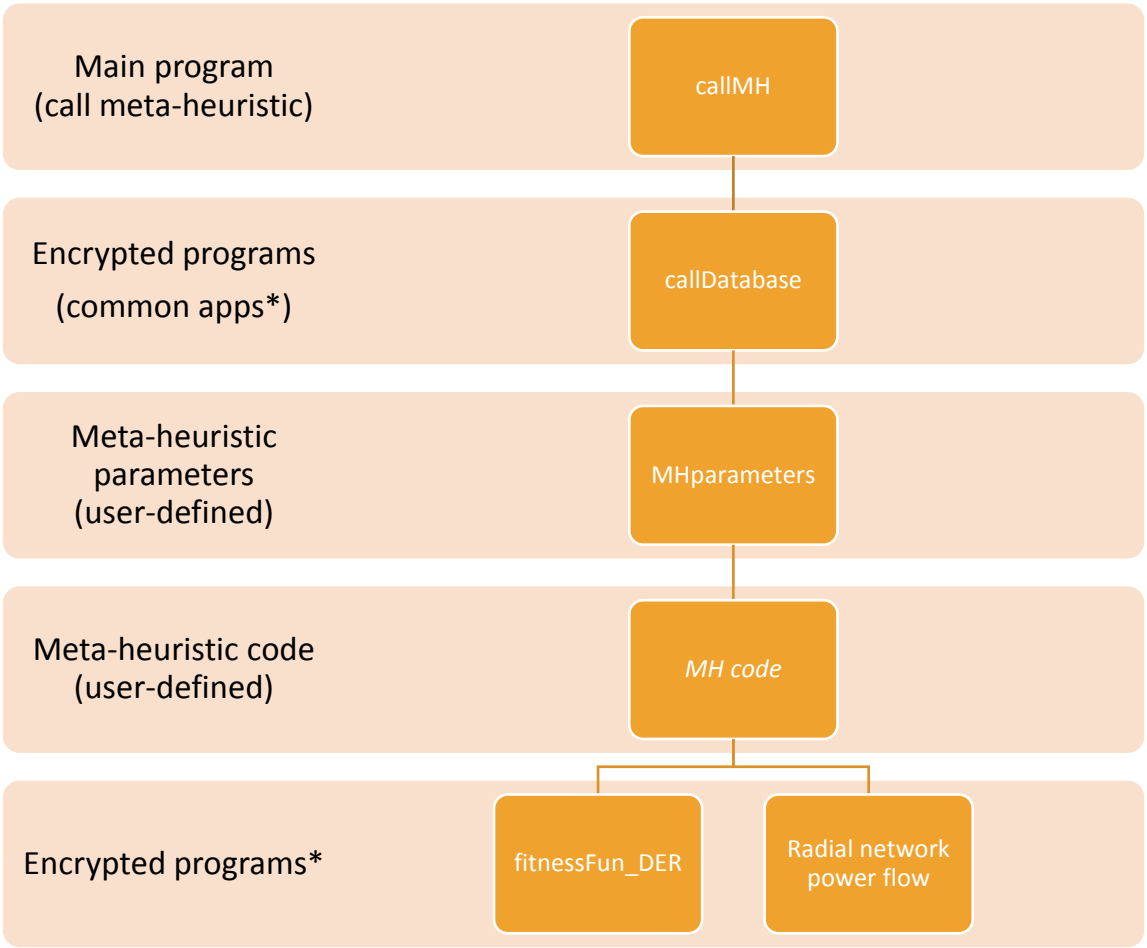
The parameters are described by: N_{DG} is the number of DG units; N_S is the number of external electricity suppliers; $C_{DG(I,t)}$ is the generation price of DG unit I in period t (m.u.); $C_{Supplier(S,t)}$ is the energy price of external supplier S in period t (m.u.); $C_{LoadDR(L,t)}$ is the load reduction (DR) cost of load L in period t (m.u.); $C_{Discharge(E,t)}$ is the discharging cost of ESS E in period t (m.u.); $C_{Discharge(V,t)}$ is the discharging cost of EV V in period t (m.u.); $C_{NSD(L,t)}$ is the non-supplied demand (NSD) cost of load L in period t (m.u.); $C_{GCP(I,t)}$ is the curtailment cost of DG unit I in period t (m.u.).

The variables are described by: OC is the total operation cost (m.u); $P_{DG(I,t)}$ is the active power generation of DG unit *I* in period *t* (MW); $P_{Supplier(S,t,)}$ is the active power generation of the external supplier *S* in period *t* (MW); $P_{LoadDR(L,t)}$ is the active power reduction of load L in period *t* (MW); $P_{NSD(L,t)}$ is the active power of Non-supplied demand for load L in period *t* (MW); $P_{GCP(I,t)}$ is the generation curtailment power of DG unit I in period *t* (MW);

The problem constraints are similar to [1]. The problem is mainly constrained by the network equations, namely active and reactive powers balance, voltage and angle limits, 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. A full AC power flow is used to check the network conditions [2].

1.2. Metaheuristic method framework

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



*the code should/cannot be changed

1.3. Fitness function

The fitness function *f'* (4) considers the objective *Z* of the aggregator (see (1)), plus the summation of the penalties found during evaluation of the solution. In this case g_i is the value of the i-th constraint (equality and inequality) and ρ is the configurable penalty factor (high value). See section 3 for instructions regarding fitness function and how penalties work.

$$f' = Z + \rho \sum_{i=1}^{n} \max\left[0, g_i\right] \tag{4}$$

1.4. Some assumptions of the energy scheduling problem:

1) The aggregator maximizes its profits (income minus costs)
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 to the electricity market.
7) The market in which the aggregator participates can accept any bid amount
8) Only one market is considered in the scenarios
9) Forecast scenarios of EVs travels, wind/PV and other renewables are known in advance (1 scenario)

1.5. Some notes on the implementation of the problem:

1) Internally in the fitness function, it is assumed that the charge/discharge variables for the EVs are the same, but positive is charge value and negative is discharge to save computational memory
2) The same principle describe above for EV applies for the stationary storage variables
3) The market in which the aggregator participates can accept any bid amount
4) Internally, the market value is positive for an offer (sale) and negative for a buy bid
5) Direct repair of solution is used in the fitness function (see section 3.6)

A maximum number of 50,000 evaluations is allowed in the competition. (Take into account that it is not the same as algorithm iterations).

2. Scenarios overview

This section briefly describes the scenarios prepared for the competition.

2.1. 33-bus scenario

The first scenario considers a 12.66 kV 33-bus MV distribution network with 66 distributed generation units (DG), 10 external suppliers, 1 large wind unit, 15 storage units, 1800 gridable vehicles, 1 market and 32 loads with demand response (reduce).

- 33 –bus MV distribution network

  66 DGs

  10 external Suppliers

  1 large wind unit

  15 storage units

  1800 gridable EVs (V2G)

  1 market

  32 aggregated loads with demand response reduce program

The above scenario using traditional tools (GAMS/MINLP) takes about 19 hours to solve in a state-of-the-art workstation (Intel(R) Xeon(R) CPU- E5-2620 v2 @ 2.10GHz with 16 GB RAM). The total number of equations reported by the software is 280,729 with 234,541 continuous and 88,380 binary variables.

EQUATIONS 280,729
SINGLE VARIABLES 234,541
DISCRETE VARIABLES 88,380

Total execution time: **~19 hours**

2.2. 180-bus scenario

The second scenario considers a 180-bus 30 kV MV distribution network. With 116 DGs , 1 external suppliers, 7 storage units, 6000 EVs, 1 market and 90 loads with demand response.

- 180-bus MV distribution network

  116 DGs

  1 external Suppliers

  7 storage units

  6000 gridable EVs (V2G)

  1 market

  90 aggregated loads with demand response reduce program

The above scenario using traditional tools (GAMS/MINLP) takes more than 168 hours to solve in a state-of-the-art workstation (Intel(R) Xeon(R) CPU- E5-2620 v2 @ 2.10GHz with 16 GB RAM) The total number of equations reported by the software is 910,033 with 763,033 continuous and 290,568 binary variables.

EQUATIONS 910,033
SINGLE VARIABLES 763,033
DISCRETE VARIABLES 290,568

Total execution time: more than **168 hours (1 week)**

3. Instructions for participants:

These instructions include as example the metaheuristic differential search algorithm [3] implemented and adapted to the energy resource management (It has been modified by GECAD).

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

3.1. Master function /script

callDSA.m (**this is the main file, the name should be similar for the participants, e.g. callMH.m or masterScript.m**)

The code arrangement is up to the competitor but of course it should follow a very similar structure, like some functions that are mandatory to be used.

```
clear all
clc
tTotalTime=tic; % lets track total computational time


callDatabase % script to load the caseStudyData
noRuns = 3; % Number of trials here
% get DSA parameters from DSAparameters.m file
DSAparameters

otherParameters =
setOtherParameters(caseStudyData,dsaParameters.nParticles);
% set penalties
otherParameters.ensPenalty=10000; % insufficient generation / energy
not supplied
otherParameters.voltagePenalty=10000; % bus voltage violations
otherParameters.linesPenalty=10000; % rate capacity of lines violated

[lowerBounds,upperBounds] =
setVariablesBounds(caseStudyData,otherParameters);

for iRuns=1:noRuns
    tOpt=tic;
    rand('state',sum(noRuns*100*clock))% ensure stochastic indpt
trials
    [ResDB(iRuns).solution, ResDB(iRuns).fitMaxVector, ...
    ResDB(iRuns).objMaxVector, ResDB(iRuns).otherParameters] =...

DSA(dsaParameters,caseStudyData,otherParameters,lowerBounds,upperBound
s);
    ResDB(iRuns).tOpt=toc(tOpt); % time of each trial

   writeBenchTables % changed (20/02/2017)



end

tTotalTime=toc(tTotalTime)
```

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

- *solution*: best candidate solution found (size: 1 x noVariables) vector

- *fitMaxVector*: the value of the fitness over the iterations (size: 1xnoIterations) vector

- *objMaxVector*: the actual value of the cost function without penalties (size: 2x noIterations) vector

- otherParameters returned from the fitness function (check

If the participant wants to change the default assigned penalties regarding constraint violations, please add these lines and change the value accordingly. A tweak is accepted in the competition as some optimal penalties (i.e. penalties that adapt during the optimizations) may be suggested by the participants. These penalties should be 1 or higher or the fitness function will not accept it. If you don't want to change the default penalties just remove those lines from the master. Default penalties are 10,000 per each violation found.

```
otherParameters.ensPenalty=10000; % insufficient generation / energy
not supplied
otherParameters.voltagePenalty=10000; % bus voltage violations
otherParameters.linesPenalty=10000; % rate capacity of lines violated
```

3.2. Loading the case study datasets

#callDatabase script – important to load the caseStudyData struct with all the relevant dataset from the case study scenario. Participants don't need to worry about the content of the case study and loading the files. It is already done. Just need to select in this file the scenario to test (33-bus) or (180-bus). The actual loading file for each case is encrypted. The data set variable is **caseStudyData.**

```
% available scenarios
% 33 for 33-bus network with 1800 EVs
% 180 for 180-bus network with 6000 EVs

scenario = 180; % choose 33 or 180

switch scenario
    case 33
        callDatabase_33bus % encrypted file
    case 180
        callDatabase_180bus % encrypted file
end
```

3.3. Set parameters of the metaheuristic

DSAparameters.m file – **file specific to the metaheuristic implemented by the participant**

```
dsaParameters.nParticles= 10; % population DSA, size_of_superorganism
dsaParameters.maxIterations = 2000; % number of max iterations/epochs
dsaParameters.method = 4; % 4: Elitist DSA (strategy 2) (E2-DSA)
dsaParameters.fnc='fitnessFun_DER';
dsaParameters.p1_multiplier=0.3;
dsaParameters.p2_multiplier=0.3;
dsaParameters.scale_factor=1./gamrnd(1,0.5);   % pseudo-stable walk
```

```
dsaParameters.noIterationsToGap=400; % iterations to wait for improve
dsaParameters.minIterations = 500; % number of min iterations/epochs
dsaParameters.threshold = 1e-9; % threshold for fitness improvement
```

NOTE: 50,000 evaluations per trial should be made.

3.4. Set other necessary parameters and struct

setOtherParameters.m **(encrypted) –** this file should not be changed and it is encrypted, it just sets parameters and data needed for the fitness function to work. It is a mandatory function and should be run as illustrated in main function section.

Participants have to pass the otherParameters struct to the fitnessFunction provided

3.5. Set bounds of variables

setVariablesBounds.m **(encrypted) –** this file should not be changed and it is encrypted, it just sets the bounds of the problem variables.

The outputs of this function [lowerBounds,upperBounds] – should be used by your MH to generate the initial solution and validate if the bounds are being respected in each iteration. You need to use lower/upper bounds vector to generate the in

The order of the variables in the implemented codes are case-study dependent. However, they should have this order to respect the fitness function and given bounds:

Variables name	Length
Generator active power (1)	Number of DG generators and external suppliers
Generator reactive powers (2)	Number of DG generators and external suppliers
Generator binaries (3)	Number of DG generators and external suppliers
EVs charge/discharge (4)	Number of electric vehicles
Demand response for each load (5)	Number of loads
Storage charge/discharge (6)	Number of ESS units
Market (7)	Number of markets (one)
Substation transformer tap (8)	One

These variables are repeated for each period in the solutions matrix (*individual x periods*):

	Period 1								**…**								**Period *T***							
1	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
2	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
…	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
N	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8

The following parameters are used to identifty the ids of each type of variables. These ids are used to locate the type of variables in solutions matrix (ids correspond to the columns while individuals to the rows). One individual is possible but not very common.

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

```
otherParameters.ids.idsTAP
```

Example of use:

```
periods = caseStudyData.parameterData.numPeriods;
nParticles = size(solutions,1);
nVariables = size(solutions,2);
idsV2G= otherParameters.ids.idsV2G;
getPeriod = 2;
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
```

3.6. Fitness function (evaluation)

fitnessFun_DER.m **(encrypted) –** this is the evaluation function to be used by participants. Participants must use this function to evaluate the solutions of the metaheuristic.

The fitnessFun_DER evaluates all the population at once, i.e. solutions matrix is a NxD dimesion matrix, in which N (rows) represents the number of individuals and D (columns) represent the number of variables of the optimization problem.

```
function [solFitness, solObjFun, solutions, otherParameters] =
fitnessFun_DER(solutions,caseStudyData,otherParameters)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%              Energy Resources Management fitness function
% This function evaluates meta-heuristics solutions
% The objective function is: maximize the profits of DER management.
% "solutions" contains all the variables of the optimization problem
% "solutions" is a NxD dimension matrix in which N represents the
% number of particles (i.e. in the case of PSO) and D represents the
% number of variables of the optimization problem (dimension of the
% particle).
% "caseStudyData" contains the data of the scenario
% "otherParameters" contains auxiliary data such as ids for the sol
vector
%                              V4.0
%                     25-11-2016  (last update)
%   GECAD, ISEP, Polytechnic of Porto, 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Note: Please refer to section 3.6.2 regarding the magnitude of each constraint violation.

3.6.1. Direct *repair* of solutions:

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

- Charge/discharge rates of EVs/ESS are adjusted taken into account the energy remaining the battery and the maximum capacity of the batteries (no need for penalties, as the correction are guaranteed)

- The demand/generation balance is made using a merit sort process
- A power flow is run and the total system losses are compensated by generators
- If the balance is not possible a penalty is thrown.

Inputs:

Solutions is a matrix with the variables of the problem (columns). Individuals are by rows. This variable comes from the MH population. Only 1 individual is also possible (one row).

caseStudyData: as loaded by callDatabase function (see section 3.2)

otherParameters: as loaded by setOtherParameters function (see subsection 3.3)

Outputs:

solFitness provides a row-vector with a fitness value for each candidate solution.

```
solFitness: 10x1 double =

   1.0e+05 *

    0.4311
    0.7301
    0.6298
    1.0295
    0.8288
    0.7301
    0.9305
    0.7298
    0.6301
    0.9287
```

solFitness output for 10 candidate solutions evaluated with the fitnessFun_DER.m

solObjFun provides a matrix with the actual objective function (without penalties) for each candidate solution

```
solObjFun: 10x2 double =

   1.0e+04 *

    4.9986    4.3095
    5.0374    4.3387
    5.0308    4.3287
    5.0529    4.3484
    5.0480    4.3361
    5.0360    4.3367
    4.9876    4.2922
    5.0553    4.3535
    5.0265    4.3276
    5.0522    4.3387
```

solObjFun output for 10 candidate solutions evaluated with the fitnessFun_DER.m

Note: As direct repair is being used in fitness function as outlined before, the repaired solutions vector is also returned (third). The participants must update their population in the MH side.

Finally, otherParameters struct is used to return some valuable information captured during the evaluation phase in fitnessFun_DER.

```
                genCosts: [10x24 double]
            loadDRcosts: [10x24 double]
         v2gChargeCosts: [10x24 double]
      v2gDischargeCosts: [10x24 double]
             v2gBalance: [10x6000x24 double]
    storageChargeCosts: [10x24 double]
 storageDischargeCosts: [10x24 double]
              stBalance: [10x7x24 double]
          pensVoltageU: [10x24 double]
          pensVoltageL: [10x24 double]
        pensMaxSLines: [10x24 double]
          penSlackBus: [10x24 double]
                pfDB: [24x10 struct]
```

Example output of otherParameters struct returned with the fitnessFun_DER.m

3.6.2. Penalties[1] for the last *candidate* solutions evaluated by the fitness function

- otherParameters.pensVoltageU: penalties regarding overvoltage (by solution and period)
- otherParameters.pensVoltageL: penalties regarding undervoltage (by solution in each period)
- otherParameters.pensMaxSlines: penalties regarding thermal capacity violation in lines (by solution and period)
- otherParameters.penSlackBus: if generation is not enough, i.e. there is energy not supplied, a penalty is assigned (by candidate solution and period)

In the MH function, please add some code like the one provided below after determining the best candidate solution, i.e. after evaluating the solutions. In the sample code below *indexbest* is the index for the best candidate solution (determined on your side – not in the fitnessFun_DER: Please store *indexbest* in otherParameters.idBestParticle.

```
% store other information
otherParameters.idBestParticle = indexbest;
otherParameters.pfFinal = otherParameters.pfDB(:,indexbest);
otherParameters.genCostsFinal = otherParameters.genCosts(indexbest,:);
otherParameters.loadDRcostsFinal = otherParameters.loadDRcosts(indexbest,:);
otherParameters.v2gChargeCostsFinal = otherParameters.v2gChargeCosts(indexbest,:);
otherParameters.v2gDischargeCostsFinal =otherParameters.v2gDischargeCosts(indexbest,:);
otherParameters.storageChargeCostsFinal =
otherParameters.storageChargeCosts(indexbest,:);
otherParameters.storageDischargeCostsFinal =
otherParameters.storageDischargeCosts(indexbest,:);
otherParameters.stBalanceFinal = otherParameters.stBalance(indexbest,:,:);
otherParameters.v2gBalanceFinal = otherParameters.v2gBalance(indexbest,:,:);
otherParameters.pensVoltageUFinal =  otherParameters.pensVoltageU(indexbest,:);
otherParameters.pensVoltageLFinal = otherParameters.pensVoltageL(indexbest,:);
otherParameters.pensMaxSLinesFinal = otherParameters.pensMaxSLines(indexbest,:);
otherParameters.penSlackBusFinal = otherParameters.penSlackBus(indexbest,:);
```

3.7. Benchmark results (text-files)

[1] Voltages deviations allowed in the scenarios are 5% (ΔV % ≤ 5%). Line flow cannot be higher than the maximum capacity (Max S - MVA rate).

The output is written to text-file using the *writeBenchTables.m* 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	
…	
Run31	

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

	Best Fit	Avg Convergence Rate	Penalties Balance	Penalties Max S Lines	Penalties Voltage (Lower bound)	Penalties Voltage (Upper bound)
Run1						
Run2						
Run3						
…						
Run31						

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

	Best fitness	Worse fitness	Median	Mode	Standard deviation	Variance

A number 31 trials should be made.

50,000 evaluations per trial should be made.

Material to be submitted to the organizers:

For each scenario, these 3 benchmark text files results 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

output_data_case_implementation_name.zip

(e.g. output_data_OSDER_PSOAlgorithm_Smith).

The zipped folder must be summited to joaps@isep.ipp.pt and j.l.ruedatorres@tudelft.nl by 30th March 2017

Bibliography

[1] 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.

[2] D. Thukaram, H. M. Wijekoon Banda, and J. Jerome, “A robust three phase power flow algorithm for radial distribution systems,” *Electr. Power Syst. Res.*, vol. 50, no. 3, pp. 227–236, 1999.

[3] P. Civicioglu, “Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm,” *Comput. Geosci.*, vol. 46, pp. 229–247, 2012.