DEMO Documentation
version 1.3

Tea Tušar
Department of Intelligent Systems
Jožef Stefan Institute
Ljubljana, Slovenia
tea.tusar@ijs.si

November 3, 2009

Table of Contents

1 Introduction

This is the documentation of the program DEMO, which implements the algorithm DEMO (Differential Evolution for Multiobjective Optimization) described in detail in (Tušar, 2007), (Tušar and Filipič, 2007) and (Robič and Filipič, 2005). DEMO was developed for minimizing multiple objectives of numerical functions and extern simulators (see Section 3) and can also handle constraints (although the mentioned documents do not explain this).

This document will not go into the details of the algorithm DEMO—it rather focuses on the usage of the program DEMO and requires some background knowledge of multiobjective optimization and evolutionary algorithms for multiobjective optimization.

2 Running DEMO

The program `demo.exe` must be run using the following syntax:

```
demo.exe <ini_file> [<parameter name> <parameter value>]
```

The file `ini_file` should contain all the necessary parameters and their values (see Section 4 for details). After `ini_file`, arbitrary parameters (and their values) can be listed—each such entry overwrites the corresponding parameter value read previously from the file `ini_file`. For example, if the file `ini_file.txt` contains the following lines:

```
pop_size
20
```

and DEMO is called with:

```
demo.exe ini_file.txt pop_size 50,
```

then the optimization algorithm DEMO will use a population of size 50.

The program DEMO was written in C++[1]and compiled on Windows XP. Since DEMO is a simple console application (no fancy VCL classes have been used), it should not be too difficult to compile on other operating systems. However, be careful with the usage of extern simulators, since the functions used there are specific for Windows™.

[1]Actually, we used C++ from Borland® Developer Studio for Windows™Version 10.0.2288.42451 Update 2.

3 Use of an extern evaluator

DEMO can be used for optimization of an extern evaluator/simulator. In that case, the following parameters need to be specified (see Subsection 4.2 for details on these parameters):

```
num_var
num_func
num_feat
command
archive
archive_read
archive_write
sim_in
sim_out
```

The communication between DEMO and the simulator is done through files. When evaluating an individual, DEMO first writes the individual's variables into the file `sim_in` (one variable per line). After that, DEMO calls the process defined in `command`, which must take care of running the simulator using the parameters from the `sim_in` file and (after evaluation) writing the output to the `sim_out` file. The `sim_out` file must contain information in the following format:

- 0/1 (whether the evaluation was successful),
- violation (a value representing the violation of constraints of the individual—the higher the value the worse the individual),
- list of criteria (one per line),
- list of features (one per line).

If the evaluation was not successful, the violation and criteria must nevertheless be written in the `sim_out` file. Usually, a very high value for the violation and criteria is chosen in such cases. Please note also that the flag for successful evaluation should be written as plain 0/1 and not 0.00/1.00 or similar since the program reads this as a boolean value.

If the `archive_read` parameter is set to 1 (true), DEMO checks the archive for the evaluation of the current individual before calling the extern evaluator (see Figure 1). This might speed up the whole optimization when the evaluations are time-consuming. However, beware of huge archive files since they eventually slow down the whole process!

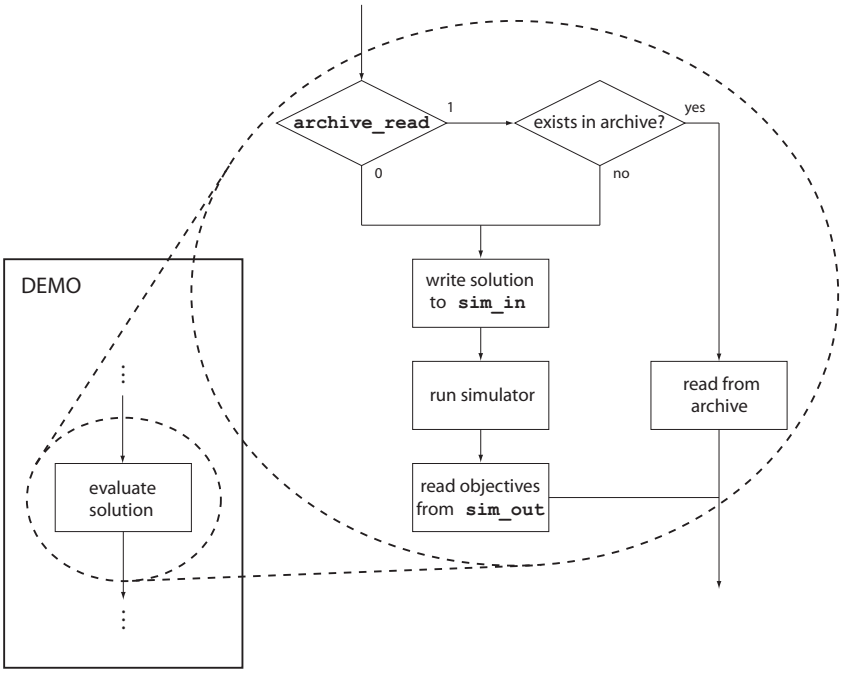

Figure 1: Extern evaluation of solutions.

4 Explanation of parameters

The program DEMO uses several parameters, which can be classified into three groups:

- parameters of the algorithm DEMO,
- parameters of the problem,
- parameters of the output.

All parameters (including the ones that do not apply for a specific problem) should be defined in the `ini_file`.

4.1 Parameters of the algorithm DEMO

The algorithm DEMO uses the following parameters:

`seed` Random generator seed. If equal to 0, the seed is chosen randomly.

`pop_size` Number of individuals in the population (at least 4).

`max_eval` Maximum number of evaluations, which serves as the stopping criterion (at least 4).

`weight` Weight F used in the creation of new individuals using the DE/rand/1/bin scheme. Usually it is set to values around 0.5, although any value in the interval $[0, 2]$ can be used. See (Price et al., 2005) for more details on the basic DE algorithm.

`cross_prob` Crossover probability CR. The efficiency of DE (and consecutively DEMO) depends upon this parameter. Although it has not been throughly studied so far, we suggest the following guidelines: if the decision space has only a few dimensions (for example, 2 or 3), the crossover probability should be high (around 0.8). If the decision space has many dimensions, the crossover probability should be low (0.3 or even lower). The values of this parameter must always lie in the interval $[0, 1]$. See (Price et al., 2005) for more details on guidelines for setting this parameter for DE.

`always_add` *(flag)* Whether to always add the candidate in the population.
0 = If the candidate dominates the parent, the candidate replaces the parent. If the parent dominates the candidate, the candidate is discarded. Otherwise, the candidate is added in the population. (The DEMO algorithm from (Robič and Filipič, 2005) and (Tušar and Filipič, 2007) uses this setting.)
1 = The candidate is always added to the population (disregarding its relation to the parent).

`sel_type` *(flag)* Type of environmental selection.
0 = DEMO$^{\text{NS-II}}$ uses environmental selection as NSGA-II (nondominated sorting and the crowding distance metric).
1 = DEMO$^{\text{IB}}$ uses environmental selection as IBEA (with indicators—see the parameter `indicator` for possible indicators).
2 = DEMO$^{\text{SP2}}$ uses environmental selection as SPEA2 (strength of the individuals and their mutual distances—see (Zitzler et al., 2001) for more information on SPEA2).
While (Robič and Filipič, 2005) used only DEMO$^{\text{NS-II}}$, in (Tušar and Filipič, 2007) all three variants are presented.

`elitism` *(flag)* This parameter applies only if the NSGA-II selection is used. It determines whether to use constrained elitism in NSGA-II's selection. See (Deb et al., 2002) for more information on NSGA-II.
0 = no
1 = yes

`elitism_r` This parameter applies only if the NSGA-II selection and the constrained elitism are used. It should lie in the interval $(0, 1]$. See (Deb and Goyal, 2000) for more information on NSGA-II's controlled elitism.

`indicator` *(flag)* This parameter applies only if the IBEA selection is used. It determines the indicator type used for fitness calculation. See (Zitzler and Künzli, 2004) for more information on IBEA.
0 = additive epsilon indicator
1 = hypervolume indicator

`rho` This parameter applies only if the IBEA selection with the hypervolume indicator is used. It determines the nadir point. For hypervolume evaluation, the criteria of all individuals in the current population are scaled to the interval $[0, 1]$. The nadir point is the reference point for the calculation of the hypervolume indicator and is usually set to 2. See (Zitzler et al., 2001) for more information on IBEA.

`kappa` This parameter applies only if the IBEA selection is used. It determines the scaling factor used in the calculation of the indicator and must lie in the interval $(0, 1]$. See (Zitzler et al., 2001) for more information on IBEA.

4.2 Parameters of the problem

Here we describe the parameters that determine the multiobjective optimization problem in question.

`func_type` *(flag)* Type of the problem function. Can have one of the following values:
0 = extern evaluation
1 = ZDT1
2 = ZDT2
3 = ZDT3
4 = ZDT4
5 = ZDT6
6 = WFG1
7 = WFG2
8 = WFG3
9 = WFG4
10 = WFG5
11 = WFG6
12 = WFG7
13 = WFG8
14 = WFG9
15 = DTLZ1
16 = DTLZ2
17 = DTLZ3
18 = DTLZ4
19 = DTLZ5

20 = DTLZ6
21 = DTLZ7
22 = DTLZ6s (shifted DTLZ6)
23 = DTLZ7s (shifted DTLZ7)
See (Zitzler et al., 2000) for more information on the ZDT problems, (Huband et al., 2006) for WFG problems, (Deb et al., 2005) for DTLZ problems and (Huang et al., 2007) for the shifted DTLZ6 and DTLZ7 variants. The parameters used for shifting the problems DTLZ6 and DTLZ7 are equal to that of the shifted DTLZ2 explained in (Huang et al., 2007).

`num_var` This parameter applies only if the extern evaluation is used. It determines the number of variables of the problem. Immediately after the number of variables, the variables' bounds and discretization must be written in the following format:

```
variable_min   variable_max   variable_discretization
```

If the variable can take continuous values, 0 is used as the discretization parameter.

`num_func` This parameter applies only if the extern evaluation is used. It determines the number of functions/criteria of the problem (at least 2). After the number of functions, the functions' lower and upper bounds must be written in the following format:

```
func_lower   func_upper
```

These bounds are used only for calculating or estimating the hypervolume of the nondominated individuals at each generation, which is output in the generation statistics. The bounds must be set in such a way, that an evaluated individual can never have the criteria outside them. Using this bounds, the criteria of all individuals are scaled to the interval $[0, 1]$ and the reference point 2 is used for calculating or estimating the hypervolume. See (While et al., 2006) for more information on the calculation of the hypervolume measure. The bounds affect only the hypervolume calculation and estimation which are part of the output information—this setting does not influence the optimization process!

`num_feat` This parameter applies only if the extern evaluation is used. It determines the number of features to be output for each individual. If 0, no features are used. Think of features as an additional information of the simulator that is useful for understanding the current solution.

`command` This parameter applies only if the extern evaluation is used. It determines the name of the executable file of the simulator.

`archive` This parameter applies only if the extern evaluation is used. It determines the name of the archive file used with extern evaluation.

`archive_read` *(flag)* This parameter applies only if the extern evaluation is used. It determines whether the archive should be checked for existence of previous evaluations of the current individual before calling the extern evaluator.
0 = no
1 = yes

`archive_write` *(flag)* This parameter applies only if the extern evaluation is used. It determines whether the individual should be written into the archive after evaluation with the extern evaluator.
0 = no
1 = yes

`sim_in` This parameter applies only if the extern evaluation is used. It determines the name of the simulator input file. See Section 3 for more information on the formats of this file.

`sim_out` This parameter applies only if the extern evaluation is used. It determines the name of the simulator output file. See Section 3 for more information on the formats of this file.

`wfg_M` This parameter applies only if the WFG or DTLZ test problems are used. It determines the number of function/criteria of the problem (at least 2). See (Huband et al., 2006) for more information on the WFG test problems and (Deb et al., 2005) for DTLZ problems.

`wfg_k` This parameter applies only if the WFG test problems are used. Together with `wfg_l` it determines the number of variables of the problem (= `wfg_k` + `wfg_l`). The following must hold for this parameter:

`wfg_k` mod (`wfg_M` - 1) == 0

See (Huband et al., 2006) for more information on the WFG test problems.

`wfg_l` This parameter applies only if the WFG test problems are used. Together with `wfg_k` it determines the number of variables of the problem (= `wfg_k` + `wfg_l`). See (Huband et al., 2006) for more information on the WFG test problems.

4.3 Parameters of the output

The parameters described here handle the output of DEMO.

`log_file_name` Name of the file with information on the progress of the optimization. The log file contains the following information:

- values of all relevant input parameters,
- information on all evaluated individuals in the order of evaluation:
  - consecutive number of the individual,
  - individual's variables,
  - whether the individual was successfully evaluated,
  - violation,
  - individual's criteria,
  - individual's features.

`log_mode` *(flag)* Type of output for the log file:
0 = do not create the log file
1 = overwrite the existing file
2 = append to the existing file

`front_file_name` Name of the file with information on the best individuals.

`front_mode` *(flag)* Type of output for the file with the best individuals:
0 = do not create the file with the best individuals
1 = overwrite the existing file
2 = append to the existing file

`front_gen` Number determining the generations, at which the best individuals are output:
1 = output the best individuals of all generations
k = output the best individuals of every k-th generation
Note that if k is greater than the number of all generations, there will be no output of the best individuals!

`front_long` *(flag)* This parameter determines the type of output of the best individuals:
0 = short front output (outputs only the criteria of the best individuals)
1 = long front output (outputs the variables, violations, criteria and features of the best individuals)

`gen_file_name` Name of the file with statistics of every generation. This file contains the following information:

- generation number,
- number of feasible individuals in the generation,
- number of unfeasible individuals in the generation,
- number of individuals that were not evaluated,
- average violation of the unfeasible individuals,
- number of nondominated individuals,
- exact hypervolume of the nondominated individuals (see also `num_func` and `gen_hyp_method`),
- hypervolume estimation of the nondominated individuals using the Monte Carlo method (see also `num_func`, `gen_hyp_method` and `gen_num_MC_points`),
- number of candidates that dominated the parents,
- number of parents that dominated the candidates,
- number of candidates that are incomparable to their parents.

`gen_mode` *(flag)* Type of output for the file with generational statistics:
0 = do not create the file with generational statistics
1 = overwrite the existing file
2 = append to the existing file

`gen_hyp_method` *(flag)* What method to use when calculating the hypervolume of every generation (this is given as an option, because the exact calculation of the hypervolume is computationally very expensive if the number of objectives is high (5 or more)):
0 = do not calculate the hypervolume and do not estimate it
1 = calculate the hypervolume exactly
2 = estimate the hypervolume using the Monte Carlo method and the number of points specified in `gen_num_MC_points`
3 = both calculate the hypervolume exactly and use the Monte Carlo method to estimate it
Note that the Monte Carlo estimation assumes that the entire objective space is feasible (if it is not, the estimation will be wrong)!

`gen_num_MC_points` The number of points used in the Monte Carlo estimation of the hypervolume (at least 100). Note that with a lot of points, the estimation is not only more accurate but also more computationally demanding.

`lines_file_name` Name of the file with information on dominating candidates. For each pair parent-candidate, where the candidate dominates the parent, an entry consisting of the objectives of both individuals is output to this file. This can be

used to draw the lines that connect the parent to the better candidate. These lines show the progress of the optimization in the objective space.

`lines_mode` *(flag)* Type of output for the file with lines:
0 = do not create the file with lines
1 = overwrite the existing file
2 = append to the existing file

5 Changes from previous versions

New in DEMO v1.3:

- The DTLZ6s and DTLZ7s benchmark problems are corrected.

New in DEMO v1.2:

- Two new problem functions are implemented: DTLZ6s and DTLZ7s (the shifted DTLZ6 and DTLZ7 functions, respectively).
- The hypervolume of feasible individuals in each generation can be calculated accurately or estimated using the Monte Carlo approach. The desired method is set using the new parameter `gen_hyp_method`, while the number of points used in the Monte Carlo estimation is set with `gen_num_MC_points`.
- A bug in the `IsEqual` function is corrected—a new function `IsEqualCrit` is added that compares two vectors from the objective space, while `IsEqual` is now used only to compare vectors from the decision space.

Acknowledgment

Thanks to the creators of the PISA environment (`http://www.tik.ee.ethz.ch/pisa`) and to the Walking Fish Group (`http://www.wfg.csse.uwa.edu.au/`) for making their code publicly available.

References

K. Deb and T. Goyal. Controlled elitist non-dominated sorting genetic algorithms for better convergence. KanGAL report 200004, Indian Institute of Technology, Kanpur, India, 2000.

K. Deb, A. Pratap, S. Agrawal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2): 182–197, 2002.

K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable test problems for evolutionary multi-objective optimization. In A. Abraham, R. Jain, and R. Goldberg, editors, *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*, chapter 6, pages 105–145. Springer, 2005.

V. L. Huang, A. K. Qin, K. Deb, E. Zitzler, P. N. Suganthan, J. J. Liang, M. Preuss, and S. Huband. Problem definitions for performance assessment of multi-objective optimization algorithms. Technical report, Nanyang Technological University, Singapore, January 2007.

S. Huband, P. Hingston, L. Barone, and L. While. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, 10(5):477–506, 2006.

K. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Springer-Verlag New York, Inc., 2005.

T. Robič and B. Filipič. DEMO: Differential evolution for multiobjective optimization. In *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, pages 520–533, March 2005.

T. Tušar. Design of an algorithm for multiobjective optimization with differential evolution, 2007.

T. Tušar and B. Filipič. Differential evolution versus genetic algorithms in multiobjective optimization. In *Proceedings of the Fourth International Conference on Evolutionary Multi-Criterion Optimization (EMO 2007)*, pages 257–271, March 2007. To appear.

L. While, P. Hingston, L. Barone, and S. Huband. A faster algorithm for calculating hypervolume. *IEEE Transactions on Evolutionary Computation*, 10(1):29–38, 2006.

E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *Proceedings of the Eighth International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pages 832–842, September 2004.

E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm. In *Proceedings of Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems (EUROGEN 2001)*, pages 95–100, September 2001.