AMOEA-MAP: a framework for expensive multi-objective optimization

v.05 (2016)

Aras Ahmadi

1. Introduction

The optimization of industrial processes usually requires solving expensive multi-objective optimization problems. AMOEA-MAP aims to solve efficiently these problems with a low pre-defined computational budget (200-300 function evaluations are usually enough to deal with bi- and tri-objective problems).

Moreover, the very structure of framework is evolutionary; which is to say gradient-free, therefore adapted to complex black-box problems where the numerical approximations of gradient are usually unreliable.

AMOEA-MAP is composed of two main operators:

(i) A memory-based adaptive partitioning algorithm (MAP), aiming at a quick identification of optimal zones with fewer computational efforts by providing a dynamic self-tuning reticulation of the search space; which is to say that the partitioning granularity over the search space is refined dynamically as a function of optimization improvement.

(ii) An archived-based multi-objective evolutionary algorithm (AMOEA), tailored for expensive optimization problems, by simultaneously maintaining two populations of solutions (a large archive population and a small operational population), in order to steer and optimize the convergence speed and the diversity of the optimal solutions on the Pareto front.

2. How to use AMOEA-MAP package

2.1. Package

1. Unzip AMOEA-MAP.zip in the directory where you wish to run the optimization.
2. In Main_Program.py, general settings as well as the optimization problem can be set (see Section 2.2).
3. By defining the optimization problem (see Section 2.3) and running Main_Program.py under python the optimal Pareto front is attained. Results are stored in AMOEA_MAP_Pareto_Fronts.csv with the following order: 1) decision variables, 2) objective values, 3) constraint violation.

2.2. General settings

General settings are defined in Main_Program.py through arg dictionary:

```
arg = {
      "Population size" : 30,
      "Number of variables" : 8,
      "Number of objectives" : 2,
      "Constraints": False,
      "Memory-based Adaptive Partitioning (MAP)": True,
      "Expensive optimization": True,
      "Max function calls": 200,
      "Genenration Max" : 2000,
      "Crossover" : ["SBX", 1.],
      "Reference point for HVI(DS)" : [10,10],
   }
```

1. “Number of variables” and “Number of objectives” are to be set following the problem nature and dimension. For instance, in case of ZDT1 test function, we have 2 objectives and the problem dimension must be set to 8.
2. The population size designates the size of archive population and must be at least 3 times the number of variables. Be sure to provide a minimum population size of 20 even for low dimensional problems.
3. If the problem is constrained, turn the False value into True according to “Constraints” setting.
4. Two stopping criteria can be set: a) “Genenration Max”, which is the maximum number of generations authorized, b) “Max function calls”, which is the maximum number of function evaluations authorized (computational budget).
5. According to “Memory-based Adaptive Partitioning (MAP)” setting, users can activate or deactivate MAP by setting the corresponding logical value to True or False, respectively.
6. By activating “Expensive optimization”, archive-based multi-objective evolutionary together with IAMO mutation operator will be automatically used, which force the algorithm to find the optimal Pareto front within the pre-defined computational budget (“Max function calls” = 200 for bi-objective problems, “Max function calls” = 300 for tri-objective problems are recommended).
7. SBX crossover operator is used in this algorithm, where users can set their own crossover probability. The crossover probability is set to 1.0 by default, which represents a 100% probability for the crossover operation.

8. According to the mutation operation, both the operator and the probability are automatically determined through the choice of optimization method. For an inexpensive optimization ("Expensive optimization ": False), the polynomial mutation operator is used where the mutation probability is set to 1/number of variables. On the contrary, for an expensive optimization ("Expensive optimization": True), the IAMO mutation operator is used where the mutation probability is self-tuned.
9. Users have to fix the reference point for HyperVolume indicator (HVI) calculations via the setting item “Reference point for HVI(DS)”. The reference point must be geometrically above the optimal Pareto front. For bi-objective problems, the reference point must be a vector of size 2, whereas for tri-objective problems, the reference point has to be a vector of size 3. In general, the size of the reference point must always follow the number of objective functions.

Once, general settings are set in arg dictionary as described earlier, users can define the optimization problem by specifying the name of the benchmark to be optimized in Main_Program.py. For instance, in order to optimize the bi-objective ZDT1 benchmark, the acronym ZDT1_2D has to be used to call the benchmark as provided below. The definition of new user benchmarks and variable bounds are further described in Section 2.3.

```
Benchmark = ZDT1_2D(arg)
```

2.3. Definition of optimization problem

The package is provided with only three benchmarks available in MOO_functions_.py and listed below. However, users can develop their own benchmarks by following the same structure already provided in MOO_functions_.py.

Benchmarks	Acronym	Type	
ZDT1 (2-objectives)	ZDT1_2D	2 objectives	unconstrained
DTLZ1 (3-objectives)	DTLZ1_3D	3 objectives	unconstrained
Beam design	beam_design_2D	2 objectives	constrained

Variable bounds can to be defined or modified in MOO_functions_.py where the benchmark is defined and initialized. For instance, according to ZDT1_2D problem, bounds are set to [0,1] for all decision variables as follows,

```
for i in range(Nvar):
    Lbounds.append(float(0.0))
    Ubounds.append(float(1.0))
```

DTLZ1 (tri-objective, unconstrained)

Let's see how to optimize the multimodal 3-dimensional DTLZ1 problem. First, be sure to reset the arg dictionary by properly defining the problem dimension and the choice of optimization method:

```
arg = {
      "Population size" : 32,
      "Number of variables" : 6,
      "Number of objectives" : 3,
      "Constraints": False,
      "Memory-based Adaptive Partitioning (MAP)": True,
      "Expensive optimization": True,
      "Max function calls": 300,
      "Genenration Max" : 2000,
      "Crossover" : ["SBX", 1.],
      "Reference point for HVI(DS)" : [10,10,10],
   }
Benchmark = DTLZ1_3D(arg)
```

The choice of benchmark has been made through the acronym DTLZ1_3D. Accordingly, variable bounds are set to [0,1] for all decision variables in MOO_functions_.py:

```
for i in range(Nvar):
    Lbounds.append(float(0.0))
    Ubounds.append(float(1.0))
```

DTLZ1 problem includes three objective functions; therefore "Number of objectives" was set to 3. The problem dimension is 6, so "Number of variables" was set to 6. The Expensive optimization approach was chosen; therefore both "Expensive optimization" and "Memory-based Adaptive Partitioning (MAP)" options were activated. Since the problem is tri-objective, a maximum number of 300 function calls was set as computational budget. A 3-dimensional reference point above the optimal Pareto front was set (Reference point for HVI(DS)": [10,10,10]) for hypervolume indicator calculations.

By running the algorithm the following optimal Pareto front can be attained (Fig. 1), where the optimal Pareto set and the objective values can be found in *AMOEA_MAP_Pareto_Fronts.csv*, respectively. The problem being unconstrained, the constraint violations will be reported zero by default.

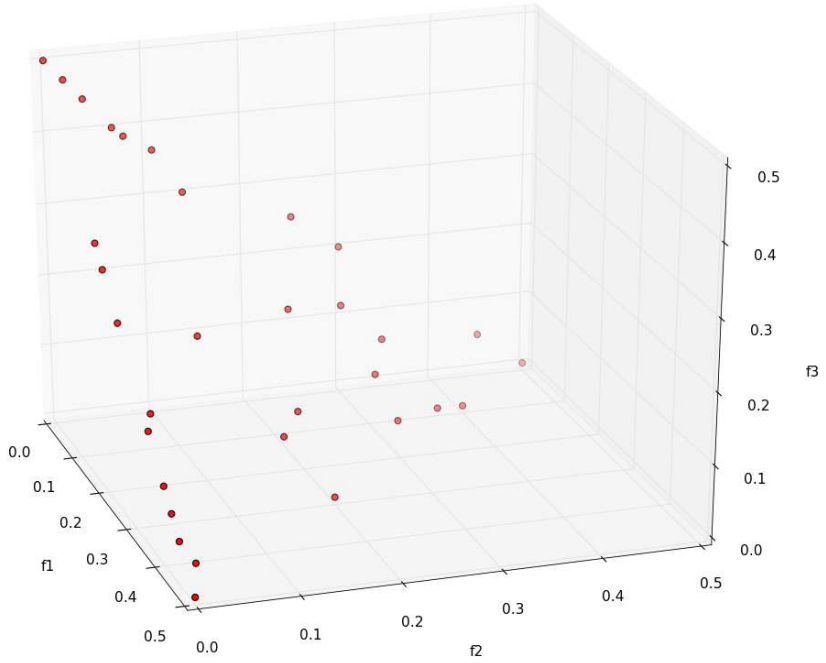

Fig 1: Optimal Pareto front for tri-objective DTLZ1 benchmark

Beam design (bi-objective, constrained)

Beam design is a stiff constrained multi-objective problem. First, be sure to set correctly the arg dictionary by defining properly the problem dimension (4 decision variables, 2 objectives), the choice of constraints ("Constraints": True), and the choice of optimization method ("Memory-based Adaptive Partitioning (MAP)": True, "Expensive optimization": True):

```
arg = {
      "Population size" : 32,
      "Number of variables" : 4,
      "Number of objectives" : 2,
      "Constraints": True,
      "Memory-based Adaptive Partitioning (MAP)": True,
      "Expensive optimization": True,
      "Max function calls": 200,
      "Genenration Max" : 2000,
      "Crossover" : ["SBX", 1.],
      "Reference point for HVI(DS)" : [100,100],
   }
Benchmark = beam_design_2D(arg)
```

The choice of optimization problem is made through the acronym beam_design_2D. Bounds for the 4 decision variables are set independently to $0.125 \leq x_1 \leq 5,\ 0.1 \leq x_2 \leq 10,\ 0.1 \leq x_3 \leq 10,\ 0.125 \leq x_4 \leq 5$, throughout the following scripts in MOO_functions_py:

```
Lbounds.append(float(0.125))
Lbounds.append(float(0.1))
Lbounds.append(float(0.1))
Lbounds.append(float(0.125))
Ubounds.append(float(5.0))
Ubounds.append(float(10.0))
Ubounds.append(float(10.0))
Ubounds.append(float(5.0))
```

The problem includes two objective functions; therefore the Number of objectives was set to 2. The problem dimension is 4, so “Number of variables” was set to 4. The Expensive optimization approach was chosen; therefore both Expensive and MAP options were activated. Since the problem is bi-objective, a maximum number of 200 function calls was set as computational budget. An appropriate 2-dimensional reference point above the optimal Pareto front (Reference point for HVI(DS)”: [100,100]) is chosen for hypervolume indicator calculations.

By running the algorithm, the following convex optimal Pareto front can be attained (Fig. 2), where the optimal Pareto set, the objective values, as well as the constraint violations can be found in AMOEA_MAP_Pareto_Fronts.csv, respectively.

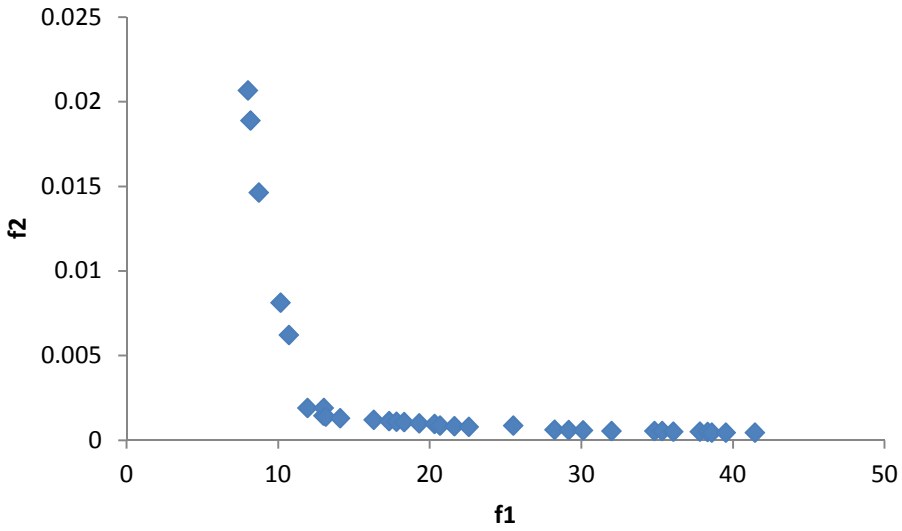

Fig 2: Optimal Pareto front for Beam design benchmark (constrained)

3. Python requirements

The code has been provided under python 2.7. While running the optimization code in your own environment, make sure that the following python packages are already functional: numpy, math, json, csv, os, sys, random, copy.

4. Publications, License and Contact

4.1. Citations

Please use the following references to cite AMOEA-MAP framework and MAP algorithm, respectively:

[1] Ahmadi et al., 2016. *An archive-based multi-objective evolutionary algorithm with adaptive search space partitioning to deal with expensive optimization problems: application to process eco-design*, Computers and Chemical Engineering 87 (2016) 95–110.

[2] Ahmadi A., 2016. *Memory-based Adaptive Partitioning (MAP) of search space for the enhancement of convergence in Pareto-based multi-objective evolutionary algorithms*, Journal of Applied Soft Computing 41 (2016) 400–417.

4.2. License

4.3. Contact

Please feel free to send any comments or questions to the following contact:

+33 (0)5 61 55 97 82
Aras.Ahmadi@insa-toulouse.fr

INSA Toulouse, LISBP
135 avenue de Rangueil
31077 Toulouse CEDEX 04
France