Black-Box Optimization Benchmarking of the Multiobjective Optimizer adaptive IBEA (ϵ -indicator) with the COCO platform

Martin BAUW
Robin DURAZ
Jiaxin GAO
Hao LIU
Luca VEYRON-FORRER

ABSTRACT

This paper discusses the implementation of a multiobjective evolutionary algorithm (MOEA). We implemented the Multiobjective Optimizer IBEA (indicator-based evolutionary algorithm) with ϵ-indicator [9] in Python and benchmarked it using the COCO platform[6]. In addition, we compared the results obtained with this algorithm with those of NSGA 2 and Random Search.

KEYWORDS

Evolutionary algorithm, Benchmarking, Black-box optimization, Bi-objective optimization, Decision Vector

1 INTRODUCTION

The adaptive IBEA we implement is dedicated to the following improvements: it requires no diversity preservation techniques within its population evolution mechanism, and it aims at taking into account arbitrary preference information thanks to its ϵ additive binary quality indicators. It operates on a set of candidates in a decision space, which will be modified thanks to selection and variation mechanisms.

The parallel with evolution can be reminded here: selection in our algorithm could be associated with the competition for reproduction and resources in nature, while variation illustrates the ability of existing living creatures to create new living beings thanks to genetic recombination and mutation. Note that due to the randomness of the variation process some individuals may not be transformed in the evolutionary mechanism.

Since we are in the case of multiobjective optimization, it is possible that several optimal objective vectors co-exist: they would be trade-offs between the different objectives we are pursuing. As we will discover in the algorithm description, our IBEA could, as any general stochastic search algorithm, be divided into three main elements: a working memory (the population of decision space candidates), a selection module, and a variation module. Among the differences with single-objective optimization, the working memory can here consider several solutions at a time. In single-objective optimizatin no mating selection is required and variation translates into modifying the current solution candidate.

None,
. None.
DOI: None

1.1 Why binary quality indicators ?

Unary quality measures have been proved to be theoretically limited: they do not allow to determine whether a Pareto set approximation is better than another one. This limitation is still valid for a finite combinatio of unary quality measures. Binary quality indicators overcome part of the limitations and can, for certain binary indicators, indicate whether a Pareto approximation set is better than another one. [10]

2 DESCRIPTION OF ALGORITHM

2.1 Description of the objects associated with the algorithm

The algorithm input consists of decision space vectors $x^i \in X$, an objective space Z and objective functions $f^j : X \to Z$. We suppose that $X \subseteq \mathbb{R}^l$ with $l \in \{2, 3, 5, 10, 20, 40\}$ and $Z \subseteq \mathbb{R}^2$.

The output of a MOEA is a set of incomparable decision vector, meaning that no member of the output set dominates another one. Domination between decision vectors is defined as follows: a decision vector x^1 is said to dominate another decision vector x^2 ($x^1 \succ x^2$), if $f_i(x^1) \leq f_i(x^2) \forall i \in \{1, ..., n\}$ and $\exists j \in \{1, ...n\}$ for which $f_j(x^1) < f_j(x^2)$.

This output will be our Pareto set approximation, and the space of Pareto set approximations will be noted as Ω. Our algorithm relies on a binary quality indicator $I : \Omega \times \Omega \to \mathbb{R}$ which associates a real number to k Pareto set approximations.

Our binary quality indicator is defined as the minimum distance by which a Pareto set approximation needs to be translated in each dimension in objective space so that another approximation (image of a decision space vector) is weakly dominated. The weak domination is defined as follows: decision vector x^1 weakly dominates x^2, written $x^1 \succeq x^2$, if x^1 dominates x^2 or the corresponding objective vectors are equal [9]. The mathematical translation of this definition consists in the following equation:

$$I_{\epsilon^+(A,B)} = min_\epsilon \{\forall x^2 \in B \, \exists x^1 \in A : f_i(x^1) - \epsilon \leq f_i(x^2) for \, i \in \{1, ..., n\}\} \quad (1)$$

2.2 Description of the algorithm adaptive IBEA

The algorithm steps is described in pseudo-code 1. The adaptive version of IBEA answers the potential issue of widely spread binary quality indicators values. Too spread out indicators values

complicate the task of determining a correct value for K, the scaling factor associated with our indicator. By adaptivaly scaling the binary quality indicators values back to a common $[-1;1]$ interval, we substantially suppress the need to adapt K to our different problems.

IBEA can be faster than other algorithms since it only compares pair of decision vectors rather than complete approximation sets.

2.3 Details regarding mating selection

Mating selection aims at picking promising solutions for variation and usually is performed in a randomized fashion. Perform binary tournament selection with replacement on P in order to fill a temporary mating pool (the variable $P\_$ in our implementation). The binary tournament consists in keeping the candidate with the best fitness value from the random pick.

2.4 Details regarding mutation

The mutation operator modifies individuals by changing small parts in the associated vectors according to a given mutation rate. The mutation rate determines how much of the considered population we will use to generate new decision space vectors. This process reminds how related evolutionary algorithms seem related to biological evolution, mimicking natural hereditary relationships. For mutation we use a polynomial mutation operator.

The operations involved in one single mutation are as follows. A random individual is picked from the population P. According to a uniform probability pick, the mutation is realized using one mathematical transformation applied to its coordinates in the decision space. Note that the transformation change from one coordinate to another, since the uniform probability pick is repeated for each. In our case, there are two forms of transformations on coordinates. In the following equations, u is uniformly picked in $[0,1]$, $ind[j]$ the $j-th$ coordinate of the already existing picked individual, $p_{mut}[j]$ the $j-th$ coordinate of the newly generated individual, Up the biggest existing coordinate value, Lo the lowest. The two latters are each defined for each decision space dimension.

- if the uniform probability pick ≤ 0.5:

$$\sigma_L = (2u)^{\frac{1}{\mu+1}} - 1 \tag{2}$$

$$p_{mut}[j] = ind[j] + \sigma_L(ind[j] - Lo) \tag{3}$$

- else:

$$\sigma_R = (2(1-u))^{\frac{1}{\mu+1}} \tag{4}$$

$$p_{mut}[j] = ind[j] + \sigma_R(Up - ind[j]) \tag{5}$$

This is implemented thanks to a loop on individuals, with a nested loop treating each decision space dimension coordinate. Mathematical formulas involving hyperparameters are based on [2].

2.5 Details regarding recombination

The recombination operator takes a certain number of parents and creates a predefined number of children by combining parts of the parents. For recombination we use a simulated binary crossover (SBX) operator. To mimic the stochastic nature of evolution, a crossover probability is associated with this operator. In our case, two parents are selected among the current population. A uniform

Algorithm 1 Adaptive IBEA

Input:
α (population size)
N (maximum number of generations)
K (fitness scaling factor)

Output:
A (Pareto set approximation)
Step 1. Initialization
generate initial population of size α
set generation counter m to 0
end Step
Step 2. Fitness assignment
for all objective function f_i **do**
lower bound $\underline{b_i} = min_{x \in P} f_i(x)$
upper bound $\overline{b_i} = max_{x \in P} f_i(x)$
end for
for all objective function f_i **do**
$f_i^{'}(x) = \frac{f_i(x) - \underline{b_i}}{\overline{b_i} - \underline{b_i}}$
end for
calculate all indicator values $I(x^1, x^2)$ with $f_i^{'}$
determine max. indicator $c = max_{x^1, x^2 \in P} |I(x^1, x^2)|$
for all $x^1 \in P$ **do**
$F(x^1) = \sum_{x^2 \in P \setminus \{x^1\}} -e^{-\frac{I(\{x^1\},\{x^2\})}{ck}}$
end for
end Step
Step 3. Environmental selection
while population P $\geq \alpha$ **do**
choose x^* such that $F(x^*) \leq F(x)$ for all $x \in P$.
remove x^* from the population
update remaining individuals fitness values, *ie* $\forall x \in P$:
$F(x) = F(x) + e^{-\frac{I(\{x^*\},\{x\})}{ck}}$
end while
end Step
Step 4. Termination
If $m \geq N$ or another stopping criterion then the output A is defined as the set of nondominated decision vectors in P
end Step
Step 5. Mating selection
Binary tournament selection with replacement on P in order to fill a temporary mating pool P'
end Step
Step 6. Variation
Apply recombination and mutation operators to P'
Add the resulting offspring to P
Increment the counter ($m = m + 1$) and go to Step 2
end Step

probability pick in $[0, 1]$ written u determines the parameter used in computing the features (decision space coordinates) of the children. In the following equations, $child0 - 1[j]$ is the $j - th$ coordinate of the generated child decision vector, $parent0 - 1[j]$ the $j - th$ coordinate of the parent associated with the child.

- if the uniform probability pick ≤ 0.5:

$$\beta_q = (2u)^{\frac{1}{\mu+1}} \tag{6}$$

- else:

$$\beta_q = \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\mu+1}} \tag{7}$$

Thanks to this stochastic parameter we can compute the children's coordinates:

- first child:

$$child0[j] = 0.5((1+\beta_q)parent0[j] + (1-\beta_q)parent1[j]) \tag{8}$$

- second child:

$$child1[j] = 0.5((1-\beta_q)parent0[j] + (1+\beta_q)parent1[j]) \tag{9}$$

Regarding the hyperparameter μ, formulas were taken from [3]. In the article this parameter is referred to as η_c. A large η_c implies an offspring close to the parents in coordinates. For a smaller η_c, children solutions tend to differ more from their parents. This parameter is therefore essential to controlling the spread of the offspring.

3 DESCRIPTION OF IMPLEMENTATION

4 CPU TIMING

In order to evaluate the CPU timing of the algorithm, we have run the with restarts on the entire bbob-biobj test suite [8] for $2D$ function evaluations according to [6]. The Python code was run on a Mac Intel(R) Core(TM) i5-2400S CPU @ 2.50GHz with 1 processor and 4 cores and (compile) options xxx. The time per function evaluation for dimensions 2, 3, 5, 10, 20, 40 equals $x.x$, $x.x$, $x.x$, xx, xxx, and xxx seconds respectively.

repeat the above for any algorithm tested

5 DISCUSSION OF THE RESULTS

Results from experiments according to [6], [4] and [1] on the benchmark functions given in [8] are presented in Figures **??**, **??**, **??** and **??** and in Tables **??** and **??**. The experiments were performed with COCO [5], version 2.0, the plots were produced with version 2.0.

The **average runtime (aRT)**, used in the tables, depends on a given quality indicator value, $I_{\text{target}} = I_{\text{ref}} + \Delta I_{\text{HV}}^{\text{COCO}}$, and is computed over all relevant trials as the number of function evaluations executed during each trial while the best indicator value did not reach I_{target}, summed over all trials and divided by the number of trials that actually reached I_{target} [6, 7]. **Statistical significance** is tested with the rank-sum test for a given target I_{target} using, for each trial, either the number of needed function evaluations to reach I_{target} (inverted and multiplied by -1), or, if the target was not reached, the best $\Delta I_{\text{HV}}^{\text{COCO}}$-value achieved, measured only up to the smallest number of overall function evaluations for any unsuccessful trial under consideration.

5.1 Results of our algorithm

5.2 Comparison with NSGA 2 and Random Search

6 CONCLUSION

REFERENCES

[1] D. Brockhoff, T. Tušar, D. Tušar, T. Wagner, N. Hansen, and A. Auger. 2016. Biobjective Performance Assessment with the COCO Platform. *ArXiv e-prints* arXiv:1605.01746 (2016).

[2] K. Deb and D. Deb. 2014. Analyzing Mutation Schemes for Real-Parameter Genetic Algorithms. *International Journal of Artificial Intelligence and Soft Computing* (2014).

[3] K. Deb, S. Karthik, and T. Okabe. 2007. Self-Adaptive Simulated Binary Crossover for Real-Parameter Optimization. *Genetic and Evolutionary Computation Conference* (2007).

[4] N. Hansen, A Auger, D. Brockhoff, D. Tušar, and T. Tušar. 2016. COCO: Performance Assessment. *ArXiv e-prints* arXiv:1605.03560 (2016).

[5] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *ArXiv e-prints* arXiv:1603.08785 (2016).

[6] N. Hansen, T. Tušar, O. Mersmann, A. Auger, and D. Brockhoff. 2016. COCO: The Experimental Procedure. *ArXiv e-prints* arXiv:1603.08776 (2016).

[7] Kenneth Price. 1997. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation*. IEEE, Piscataway, NJ, USA, 153–157. DOI: http://dx.doi.org/10.1109/ICEC.1997.592287

[8] T. Tušar, D. Brockhoff, N. Hansen, and A. Auger. 2016. COCO: The Bi-objective Black-Box Optimization Benchmarking (bbob-biobj) Test Suite. *ArXiv e-prints* arXiv:1604.00359 (2016).

[9] E. Zitzler and S. Knzli. 2004. Indicator-Based Selection in Multiobjective Search. *International conference on parallel problem solving from nature* (2004).

[10] E. Zitzler, M. Laumanns, and S. Bleuler. 2004. A Tutorial on Evolutionary Multiobjective Optimization. *Metaheuristics for Multiobjective Optimization* (2004).