

EvolPy: An open source Nature-Inspired Optimization Toolbox in Python

First Author Name¹, Second Author Name¹ and Third Author Name²

¹*Institute of Problem Solving, XYZ University, My Street, MyTown, MyCountry*

²*Department of Computing, Main University, MySecondTown, MyCountry*

{f_author, s_author}@ips.xyz.edu, t_author@dc.mu.edu

Keywords: The paper must have at least one keyword. The text must be set to 9-point font size and without the use of bold or italic font style. For more than one keyword, please use a comma as a separator. Keywords must be titlecased.

Abstract: EvolPy is an open source and cross-platform Python toolbox that implements a wide range of classical and recent nature-inspired metaheuristic algorithms. The goal of this toolbox is to facilitate the use of metaheuristic algorithms by non-specialists coming from different domains. With a simple interface and minimal dependencies, it is easier for researchers and practitioners to utilize EvolPy for optimizing and benchmarking their own defined problems using the most powerful metaheuristic optimizers in the literature. On the other hand, the design of this toolbox makes it very easy for the researchers in the domain to integrate their own optimizers and compare their performance to the state of art algorithms.

1 INTRODUCTION

Nature-inspired metaheuristic algorithms are considered to be the one of the most efficient approaches to solve optimization problems (Yang, 2013). Nature-inspired metaheuristics imitate natural phenomena such as the the behavior of the nature systems like swarm based algorithms (Swarm Intelligence Algorithms - SIA), and evolution based algorithms (Evolutionary Algorithms - EA). Over last three decades, many nature-inspired metaheuristics algorithms have been developed. Swarm intelligence algorithms simulate the natural swarms such as flocks of birds, ant colonies, and schools of fishes such as Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995), and Ant Colony Optimization (ACO) (Korošec and Šilc, 2009). Recent Swarm intelligence algorithms include Cuckoo Search (CS) (Yang and Deb, 2009), Grey Wolf Optimizer (GWO) (Mirjalili et al., 2014), Multi-Verse Optimizer (MVO) (Mirjalili et al., 2016), Moth-flame optimization (MFO) (Mirjalili, 2015), Whale Optimization Algorithm (WOA) (Mirjalili and Lewis, 2016), Bat Algorithm (BAT) (Yang, 2010b), Firefly Algorithm (FFA) (Yang, 2010a), and many others. Most of the previous algorithms reaches the best solutions by exchanging the information between the swarm's individuals.

On the other hand, evolutionary based algo-

gorithms are inspired by some concepts from the Darwinian theories about evolution and natural selection. Such algorithms include Genetic algorithm (GA) (Sivanandam and Deepa,), Genetic Programming (GP)(Koza, 1992), and Evolution Strategy (ES)(Beyer and Schwefel, 2002). These algorithms use different strategies to evolve and find good solutions for difficult problems.

In the last two decades there have been a growing interest in the applications of these algorithms for solving and optimizing complex problems in different domains.

2 RELATED WORK

Today, there exist many different nature-inspired optimization libraries and toolboxes. Some of them are developed for special type of nature-inspired optimization like GEATbx (Hartmut Pohlheim,), which is a toolbox contains many variants of the genetic algorithms, and genetic Programming. GEATbx is implemented in MATLAB environment and provides many global optimization capabilities. The authors in (Matthew Wall,) developed GALib library, which contains a set of C++ genetic algorithm tools and operators for different optimization problems. GALib is

built on UNIX platforms and supported parallel experiments.

In (Fortin et al., 2012), the authors developed a novel evolutionary computation python framework called DEAP to simplify the execution of many optimization ideas with parallelisation features. The DEAP Framework includes many nature-inspired algorithms with different variations such as: Genetic Algorithm, Genetic Programming (GP), Evolution Strategies (ES), Particle Swarm Optimization (PSO), Differential Evolution (DE), and multi-objective optimization. In addition, DEAP includes Benchmarks modules containing many test functions for evaluations. In (Wagner and Affenzeller, 2004), the author presents a generic optimization environment named HeuristicLab, which is implemented using C# language. HeuristicLab is a framework for heuristic, evolutionary algorithms, and machine learning. HeuristicLab includes many algorithms such as: Genetic Programming, Age-layered Population, Structure (ALPS), Evolution Strategy, Genetic Algorithm, Island Genetic Algorithm, Particle Swarm Optimization, Relevant Alleles Preserving GA, and many others.

In (Durillo and Nebro, 2011), jMetal is developed, which is a framework contains many metaheuristic algorithms implemented in Java programming. The authors employ the object-oriented architecture to develop the framework. jMetal includes many features such as: multi-objective algorithms, parallel algorithms, constrained problems, and different representations of the problem variables. In (Cahon et al., 2004; Humeau et al., 2013), the authors present a white-box object-oriented framework called ParadisEO. ParadisEO is developed by integrating Evolving Objects (EO) - which is an C++ evolutionary computation framework - and many distributed metaheuristics including local searches (LS), and hybridization mechanisms.

Comparing our developed framework to the frameworks listed above, all frameworks were contained traditional nature-inspired algorithms such as: GA, GP, PSO, ES, and many others. Whereas in our developed framework, the framework includes the traditional nature-inspired algorithms such as PSO and nine recent nature-inspired algorithms such as: GWO, MVO, MFO, WOA, BAT, CS, and FFA. To the best of our knowledge, EvoloPy framework is the first work contains such optimizers that developed in Python language. EvoloPy is developed in very efficient way to solve computationally expensive optimization functions with very high dimensions.

3 WHY PYTHON?

Python is a general purpose scripting language which has a clear and simple syntax. With the rapid development of mature, advanced and open source scientific computing libraries and packages, Python became as one of the most popular and powerful languages for scientific computing. Moreover, Python has cross-platform runability, which works with different operating systems, and has ability to access libraries written in different programming languages and computing environments. Python also can support small-form devices, embedded systems, and microcontrollers. In addition, Python needs very minimal setup procedure to start with. Python uses modular and object based programming, which is a popular methodology to organize classes, functions, and procedures into hierarchical namespaces. All these reason has turned Python to be a popular language in a huge community of scientists.

4 TOOLBOX OVERVIEW AND FEATURED ALGORITHMS

EvolPy provides a set of classical and recent nature-inspired metaheuristic optimizers with an easy-to-use interface. The toolbox incorporates four main components which are described as follows:

- The Optimizer: this is the main interface of the of the toolbox in which the users can select a list of optimizers to run in their experiment. Using this interface, the user can configure common parameters for the algorithms like the maximum number of iterations for each run, the size of the population and the number of runs. In addition, a list of benchmark problems can be selected.
- The metaheuristics algorithms: simply this part includes the list of implemented optimizers in the framework. At the time of writing this paper, eight algorithms are implemented. A brief description of each algorithm is given as follows:
 - PSO
 - FFA
 - GWO
 - WOA
 - MVO
 - MFO
 - BAT
 - CS

- Benchmark functions and problem definition: This part includes a set of common benchmark problems commonly used in the literature for the purpose of comparing different metaheuristic algorithms. Any new user-defined cost function could be added to this list.
- Results management and visualization: this component is responsible about exporting the results and storing them in one common CSV file.

5 DESIGN ISSUES

Most of nature inspired metaheuristics are population-based algorithms. These algorithms start by randomly initializing a set of individuals each of which represents a candidate solution. Conventionally, in most of state-of-art evolutionary algorithms frameworks, populations are implemented as 2-dimensional arrays while individuals are implemented as 1-dimensional array. For example an individual I and a population P can be represented as given in equations 1 and 2.

$$I_i = [x_i^1 \quad x_i^2 \quad \dots \quad x_i^d] \quad (1)$$

$$P = \begin{bmatrix} x_1^1 & x_1^2 & \dots & x_1^d \\ x_2^1 & x_2^2 & \dots & x_2^d \\ \vdots & \vdots & \ddots & \vdots \\ x_n^1 & x_n^2 & \dots & x_n^d \end{bmatrix} \quad (2)$$

In EvoloPy, Numpy is chosen to define and represent the populations and individuals in the optimizers. Numpy is an open-source extension to Python that provides common scientific computing and mathematical routines as pre-compiled and fast functions. NumPy also supports large multidimensional arrays and matrices and it offers a variety range of functions to handle and perform common operations on these arrays. Numpy has many things in common with Matlab. This makes it easier for researchers who are familiar with Matlab use EvoloPy.

The core data structure ndarray (abbreviation for N-dimensional array) in Numpy is used to define 1-dimensional and 2-dimensional arrays that represent individuals and populations, respectively. For example, to define a randomly generated initial population with 50 individuals and 100 dimensions the following python line of code can be used

```
initialPop = np.random.rand(500,100)
```

Moreover, with Numpy it is possible to perform vector operations in simple and compact syntax. For example, a common operation is used in many metaheuristic algorithms is to check the boundaries of the

elements of the individuals after updating them. This can be performed all at once with Numpy as follows:

```
newPop=numpy.clip(oldPop, lb, ub)
```

Where lb and ub are the upper and lower bounds respectively.

6 COMPARISON WITH MATLAB

In this section we compare the metaheuristic algorithms implemented so far in EvoloPy with their peers in Matlab based on their running time. The idea here is to measure the running time of the main loop of each algorithm in EvoloPy and Matlab with an equal number of function evaluations. The experiments were performed on a personal machine with an Intel(R) Core(TM) i5-2400 CPU at 3.10Ghz and a memory of 4 GB running Windows 7 professional 32bit operating system. In all experiments, the population size and the number of iterations was set for all algorithms to 50, and 100, respectively. In order to study the effect of the size of the problem to be solved on the running time of the algorithms, all optimizer were executed using different dimension lengths ranging from a small number of 50 elements reaching up to 20,000 in both implementations, Matlab and EvoloPy. Note that the dimension length here indicates number of elements in the single individual or in another words, number of variables in the objective function. All optimizers were applied to minimize the unimodal function given in Equation 3.

$$f(x) = \sum_{i=0}^{n-1} \left(\sum_{j=0}^{i-1} x_j \right)^2 \quad (3)$$

The results of EvoloPy and Matlab implementations are shown in Figure 1. It can be noticed that with small dimension sizes there is no big significant difference in the running time between the two implementations. In most of the optimizers and up to a dimension size of 500 the running time was less than 250 second. However, for higher sizes, the difference in the running time remarkably increases. On a dimension length of 20,000 the ration was around 1:2 for most of the optimizers.

The efficient running time of EvoloPy compared to Matlab implementation when applied for solving problems with large number of variables can be explained due to the use of the powerful N-dimensional array object NumPY to represent individuals and populations.

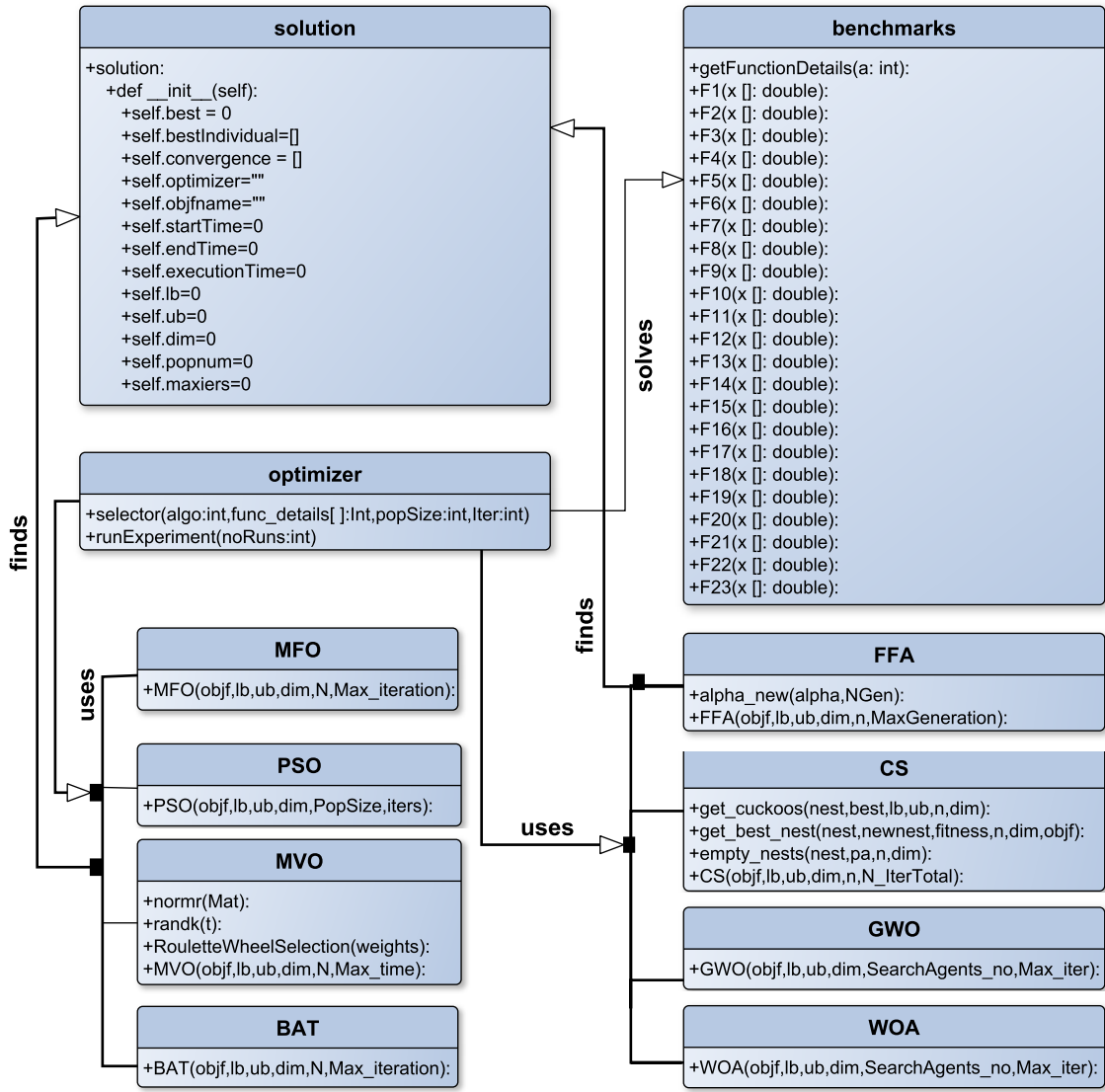


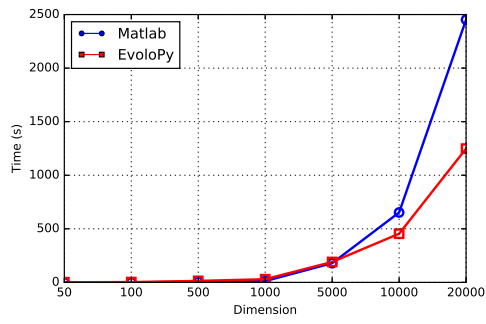
Figure 1: Class diagram of the EvoloPy framework

7 CONCLUSIONS AND FUTURE WORK

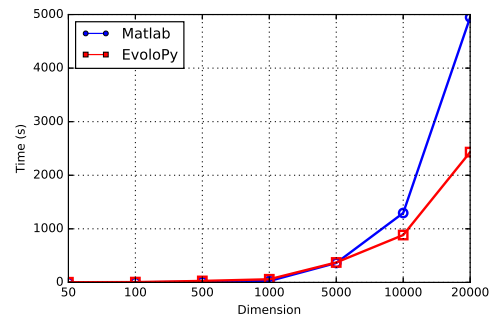
REFERENCES

- Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies – a comprehensive introduction. *Natural Computing*, 1(1):3–52.
- Cahon, S., Melab, N., and Talbi, E.-G. (2004). Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380.
- Durillo, J. J. and Nebro, A. J. (2011). jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771.
- Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175.
- Hartmut Pohlheim. Geatbx - the genetic and evolutionary algorithm toolbox for matlab.
- Humeau, J., Liefoghe, A., Talbi, E.-G., and Verel, S. (2013). ParadisEO-MO: From Fitness Landscape Analysis to Efficient Local Search Algorithms. Research Report RR-7871, INRIA.
- Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4.
- Korošec, P. and Šilc, J. (2009). A distributed ant-based algorithm for numerical optimization. In *Proceedings*

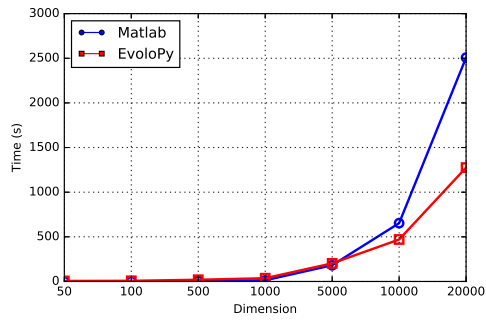
- of the 2009 workshop on Bio-inspired algorithms for distributed systems - BADS 09. Association for Computing Machinery (ACM).
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.
- Matthew Wall. Galib: A c++ library of genetic algorithm components.
- Mirjalili, S. (2015). Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-Based Systems*, 89:228 – 249.
- Mirjalili, S. and Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software*, 95:51 – 67.
- Mirjalili, S., Mirjalili, S. M., and Hatamlou, A. (2016). Multi-verse optimizer: a nature-inspired algorithm for global optimization. *Neural Computing and Applications*, 27(2):495–513.
- Mirjalili, S., Mirjalili, S. M., and Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software*, 69:46 – 61.
- Sivanandam, S. and Deepa, S. Genetic algorithms. In *Introduction to Genetic Algorithms*, pages 15–37. Springer Science Business Media.
- Wagner, S. and Affenzeller, M. (2004). The heuristiclab optimization environment. Technical report, University of Applied Sciences Upper Austria.
- Yang, X. (2010a). Firefly algorithm, stochastic test functions and design optimisation. *Int. J. Bio-Inspired Comput.*, 2(2):78–84.
- Yang, X.-S. (2010b). *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, chapter A New Metaheuristic Bat-Inspired Algorithm, pages 65–74. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Yang, X.-S. (2013). Metaheuristic optimization: Nature-inspired algorithms and applications. In *Studies in Computational Intelligence*, pages 405–420. Springer Science Business Media.
- Yang, X. S. and Deb, S. (2009). Cuckoo search via levy flights. In *Nature Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 210–214.



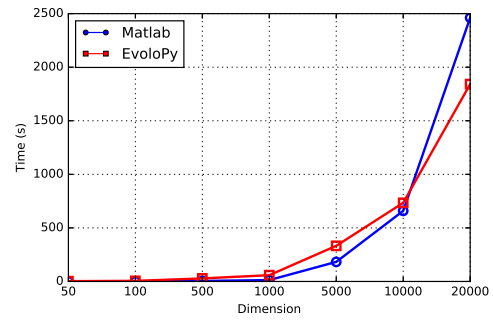
(a) BAT



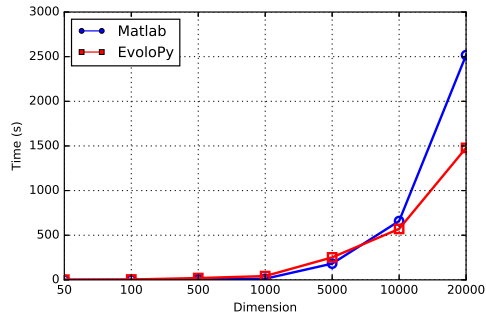
(b) CS



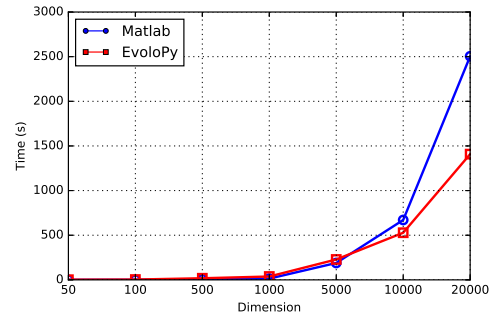
(c) FPA



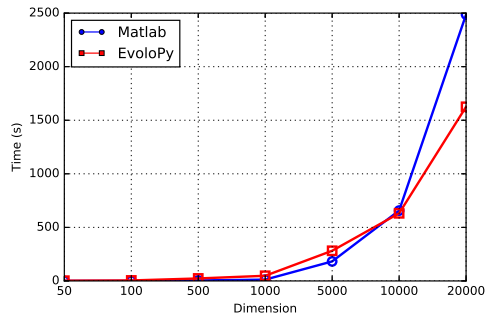
(d) GWO



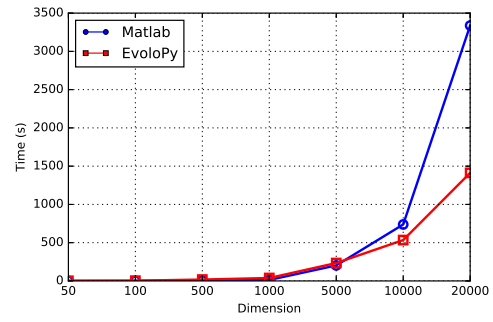
(e) MFO



(f) MVO



(g) PSO



(h) WOA

Figure 2: Comparison between the metaheuristic optimizers in EvoloPy and Matlab based on execution time of the main iteration.