IspsoPy - Python port of ISPSO R program

Owen Smith

May 5, 2021

GISC: 4500k
Dr. Huidae Cho

Code is available at https://github.com/ocsmit/ispsopy

Abstract

The Isolated-Speciation-based PSO is a meta-heuristic optimization algorithm initially written in R. Proposed an implemented is the porting of the algorithm to Python. Basic theory behind the ISPSO is discussed in addition to the status of the program itself.

1 Introduction & background

In the fields of mathematics and computer science, modeling the natural processes which surround us is a foundational part of theory which they encompass. The various algorithms which model real world phenomena to solve complex problems are termed 'Natural Computing' and can be separated into three primary facets; simulation for knowledge discovery, natural systems used for computing, and algorithms inspired by the natural world (Brabazon et al., 2015; De Castro, 2006). Simulation for knowledge discovery simulates natural phenomena to develop a better understanding of the natural phenomena themselves. This includes artificial life simulates and agent based modeling. The second facet utilizes the natural to enable computing in the form of DNA or molecules. The third facet are those algorithms which pull from the natural world and attempt to model its metaphoric processes to solve both real world and theoretical problems. While used in many areas such as predicting and classification, the field of optimization and meta heuristics pulls heavily from the third facet of natural computing. Examples of optimization problems include but are not limited to ant hill modeling to solve shortest paths problems (Dorigo and Di Caro, 1999; Shtovba, 2005), genetic algorithms which models evolutionary theory (Vose, 1999; Whitley, 1994), the plant propagation algorithm (Merrikh-Bayat, 2015), and particle swarm optimization algorithms (PSO).

The PSO introduced initially in 1995 takes inspiration from the acts of a flock of birds or a school of fish looking for prey, avoiding predators, finding optimal environments, etc. to solve nonlinear problems (Kennedy and Eberhart, 1995; Chazelle, 2009). As real world problems often have many different potential solutions a key issue with the continued development of the PSO is the ability to efficiently find both local and global optima (Özcan and Yılmaz, 2007; Seo et al., 2006). As such, the PSO introduced by Kennedy and Eberhart (1995) has seen a multitude of derivations and improvements.

The NichePSO algorithm (Brits et al., 2002) builds upon the original unimodal PSO functions to grow multiple sub swarms of particles capable of identifying multiple maximas. Speciation is

introduced by Li (2004) with the Species-based PSO which builds off of the ideas behind the K means clustering algorithm to group particles into various clusters, or species. An advantage of speciation over K means clustering is that the hyperparameters of k means which dictate how clustering takes place, is replaced by the speciation radius to appropriately determine subgroups. The SPSO however, is only capable of finding global optima and is therefore unsuited for multimodal problems.

The Isolated-Speciation-based PSO (ISPSO) (Cho et al., 2011) builds off the aforementioned SPSO to find multiple optima in a multimodal problem space. The ISPSO introduces the generation of new particles within the search space using Sobol sequences, which are a low-discrepancy sequence that were found to provide a more evenly spread out population than pseudo-random sequences (Maaranen et al., 2007). The original ISPSO algorithm was written in the R statistical language. While optimal for its initial conception, it potentially limits further access and usage. To help enable the usage of the ISPSO algorithm, this paper details its porting to Python.

2 Methods and materials

The ISPSO can be broken into several key portions. The initialization of the particles, the evaluation of the function, updating velocities, nesting, checking of criterion, and updating particle coordinates. The Python implementation utilizes the SciPy, NumPy, and Pandas libraries.

2.1 Intialization

The initial population group is generated with the Sobol low-discrepancy with each particle representing an individual parameter sample. The coordinates and velocities are contained within the vectors

$$x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D}) \;\; v_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D}) \tag{1}$$

which enable the sharing of personal and local information. After initialization the ISPSO algorithm will run until the optima have been found or the set number of iterations is reached.

2.2 Evaluation of the function

The ISPSO evaluates non linear functions. The initial particle coordinates, or rather sample value, is used as function inputs. If the evaluated output is less than the local best position (pbest), the particles location and respective function value is updated within the pbest topology. Likewise, if the evaluated value is greater than the particles group knowledge (gbest), the particles location and respective function value is updated within the gbest topology. After each function run the number of evaluations and the age of each particle is incremented.

2.3 Updating velocities

The update particle positions and function values within both the pbest and gbest topologies are subsequently used to inform each particles next velocity. Each particles current position is add to the lbest topology vector. Each particles position is checked to examine if any points are within the speciation radius defined in the SPSO. If so then a species has been found each particle within the radius is added to a species the next particle is evaluated. If there are any isolated particles, then they are added to their own species. Each particles location and species is added to the lbest topology. An additional species is created to both search for superior particles that might have failed to create their own species and for searching pbests which are superior within its radius.

x	f	v
0.0312	0.997	0.009
0.994	0.962	0.001
0.001	0.929	0.064
0.372	0.996	0.065
0.05	0.784	0.0218

Table 1: Example nests generated from IspsoPy.

2.4 Checking for convergence

With each species if the age of a particle is greater than the specified age, then it is checked for convergence around a solution. A halflife of each particles previous positions are created from the total population. The halflife is then subsequently tested for its normalized geometric mean which is defined as

$$NGM = \left(\Pi_{j=1}^{D} \frac{x_{ij}^{+} - x_{ij}^{-}}{x_{max,j} - x_{min,j}} \right)^{1/D} \tag{2}$$

where x_{ij}^{+} and x_{ij}^{-} are the maximum and minimum locations of the halflife particles and $x_{max,j}$ and $x_{min,j}$ are the upper and lower bounds of the full search space. If Eq. (2) is smaller than the threshold value of ϵ_x then one of three criterion for nesting is complete.

The other two criterion for nesting is the aforementioned age and the standard deviation of the particles halflife. If the standard deviation of the particles halflife is less than the threshold value of ϵ_f and the other criterion are met then a nest is created. When a nest is created then the particles within the nest are substituted for new initialized values, but not before the nests particles information is stored.

2.5 Updating positions

Adding the particle positions and velocities, we can get their next positions. Within neighborhoods, or species, particles are removed and replaced by superior particles which prey upon them. If there are any created nests, then those nests are checked as well before moving on to the nest location.

3 Testing, bugs, & results

The Python port at the time of writing is designed for testing with one dimensional functions. The function chosen to test with is

$$F4(x) = 1 - \exp\left(-2\log(2) \cdot \left(\frac{x - 0.08}{0.854} \right)^2 \right) \cdot \sin^6\left(5\pi(x^{3/4} - 0.05) \right) \tag{3}$$

which was designed by Beasley et al. (1993). The code is designed to be intuitive and follow some of the conventions as R by providing easy parameter list generation which can be seen as follows.

Listing 1: Example usage of IspsoPy

```
from ispso import ispso_parameters, ispso, diagonal
import numpy as np
import math
```

```
import matplotlib.pyplot as plt

# PARAMETERS
s = ispso_parameters()
s.f = f4
s.D = 1
s.xmin = np.array([0])
s.xmax = np.array([1])
s.S = 10 + math.floor(2 * (s.D)**0.5)
s.vmax = (s.xmax - s.xmin) * 0.1
s.vmax0 = diagonal(s)*0.001
s.exclusion_factor = 3
s.maxiter = 200
s.xeps = 0.001
s.feps = 0.1
s.rprey = diagonal(s) * 0.0001
s.age = 10
s.rspecies = diagonal(s) * 0.1
s.rnest = diagonal(s) * 0.1
s.plot_distance_to_solution = 0.01


r = ispso(s).run()
```

The current port runs efficiently for the function, however it is currently searching out maxima as opposed to minima. An example of the resulting nests can be seen in Table 1. This is believed to be caused by a something gone wrong within the halflife evaluation either due to bugs in the code or incorrect implementation all together. As the initial library was written in R, this port took steps to maintain similar data structures, such as the R data frame, for ease of use and understanding.

4 Future work

The current implementation of IspsoPy provides a solid base with which to build off of, however it needs work still. As mentioned, it needs further debugging and to be updated to allow for functions of more than one dimension to be evaluated. Only then will the current Python implementation be worthwhile and helpful for real world implementation.

5 References

Beasley, D., Bull, D. R., and Martin, R. R. (1993). A sequential niche technique for multimodal function optimization. *Evolutionary computation*, 1(2):101–125.

Brabazon, A., O'Neill, M., and McGarraghy, S. (2015). *Natural computing algorithms*, volume 554. Springer.

Brits, R., Engelbrecht, A. P., and Van den Bergh, F. (2002). A niching particle swarm optimizer. In *Proceedings of the 4th Asia-Pacific conference on simulated evolution and learning*, volume 2, pages 692–696. Citeseer.

Chazelle, B. (2009). Natural algorithms. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 422–431. SIAM.

Cho, H., Kim, D., Olivera, F., and Guikema, S. D. (2011). Enhanced speciation in particle swarm optimization for multi-modal problems. *European Journal of Operational Research*, 213(1):15–23.

De Castro, L. N. (2006). *Fundamentals of natural computing: basic concepts, algorithms, and applications*. CRC Press.

Dorigo, M. and Di Caro, G. (1999). Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477. IEEE.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE.

Li, X. (2004). Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In *Genetic and Evolutionary Computation Conference*, pages 105–116. Springer.

Maaranen, H., Miettinen, K., and Penttinen, A. (2007). On initial populations of a genetic algorithm for continuous optimization problems. *Journal of Global Optimization*, 37(3):405.

Merrikh-Bayat, F. (2015). The runner-root algorithm: a metaheuristic for solving unimodal and multimodal optimization problems inspired by runners and roots of plants in nature. *Applied Soft Computing*, 33:292–303.

Özcan, E. and Yılmaz, M. (2007). Particle swarms for multimodal optimization. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 366–375. Springer.

Seo, J.-H., Im, C.-H., Heo, C.-G., Kim, J.-K., Jung, H.-K., and Lee, C.-G. (2006). Multimodal function optimization based on particle swarm optimization. *IEEE Transactions on Magnetics*, 42(4):1095–1098.

Shtovba, S. D. (2005). Ant algorithms: theory and applications. *Programming and Computer Software*, 31(4):167–178.

Vose, M. D. (1999). *The simple genetic algorithm: foundations and theory*. MIT press.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85.