Genetic Programming and Evolvable Machines (2020) 21:269–272
https://doi.org/10.1007/s10710-019-09367-z

SOFTWARE REVIEW

Inspyred: Bio-inspired algorithms in Python

Alberto Tonda[1]

Published online: 2 November 2019
© Springer Science+Business Media, LLC, part of Springer Nature 2019

Inspyred 1.0 is an open-source, freely available Python module developed by Dr. Aaron Garrett of Wofford College, Spartanburg, South Carolina, USA (https://aarongarrett.github.io/wofford-webs/). The project is under active development, with the latest updates released in January 2019.

Inspyred provides Python implementations for some of the most commonly used Evolutionary Algorithms (Genetic Algorithms, Evolutionary Strategies, Differential Evolution, Pareto Archived Evolutionary Strategy, and NSGA-II) and other bio-inspired optimization techniques (ant colony optimization, particle swarm optimization, simulated annealing, and swarm intelligence).

While Inspyred's tools can be used as out-of-the-box optimization resources, its most commendable feature is its design methodology for EAs, explicitly "inspired" (pun intended) by De Jong's 2006 book *Evolutionary Computation: A Unified Approach*. Inspyred implements a generic Evolutionary Computation as a series of components/Python functions:

Problem-specific components

1. A **generator** that defines how solutions are created
2. An **evaluator** that defines how fitness values are calculated for solutions

Algorithm-specific evolutionary operators

1. An **observer** that defines how the user can monitor the state of the evolution
2. A **terminator** that determines whether the evolution should end
3. A **selector** that determines which individuals should become parents
4. A **variator** that determines how offspring are created from existing individuals
5. A **replacer** that determines which individuals should survive into the next generation
6. A **migrator** that defines how solutions are transferred among different populations

✉ Alberto Tonda
alberto.tonda@inra.fr

[1] UMR 782 GMPA, INRA, Université Paris-Saclay, Thiverval-Grignon, France

7. An **archiver** that defines how existing solutions are stored outside of the current population

What makes Inspyred particularly useful for both students and practitioners of Evolutionary Algorithms is the possibility of easily replacing any of these components with either an existing solution provided by Inspyred or a custom user-written Python function. This allows users to mix and match techniques, easily experiment with new ways of (for example) performing individual selection, variation or replacement, and develop custom evolutionary algorithms with relatively little coding effort. For example, you could create a version of NSGA-II that evolves individuals with non-fixed genome size, by just replacing the generator, evaluator, and variator, and keeping the rest of the Inspyred functions as they are.

Several popular functions/components are already provided: for example, among the possible **selectors** we can find roulette wheel, tournament, rank, truncation and uniform selection. The **variators** include the "usual suspects" among genetic operators. These range from n-point crossover to Gaussian mutation, for both genomes with continuous floating point and discrete (Boolean/int) values. The **replacers** decide how populations are updated from one generation to the next. Several popular strategies are provided, including the basic "comma", where parents are replaced by offspring (also known as generational) and "plus" strategies, in which parents and offspring compete with each other. **Observers** are basically functions called at the end of each generation. They can be used to save the current state of the algorithm (for example by writing all individuals in the population to a file), or to visualize information on the screen (Fig. 1).

When compared to other Evolutionary Algorithms Python packages, such as DEAP [1] this structure makes Inspyred more elegant and easier to modify for a specific purpose.

The documentation (https://pythonhosted.org/inspyred/) is well written. It includes several tutorials that show how different components can be replaced by user supplied functions and examples that show how to develop custom components for a specific application. Finally, a few "recipes", in the form of snippets of code, show how to tweak Inspyred to introduce advanced concepts, such as lexicographic ordering for comparing individuals, or a migrator of individuals between islands deployed on parallel machines connected by a network.

One of the most useful features of Inspyred is the ease of passing custom arguments between the components. When calling the **evolve** method of any instance of the Evolutionary Computation object, all arguments passed to the method that are not listed among its standard arguments will be added as accessible entries in a Python dictionary called **args.** (**args** is passed to all functions/components.) This means if your evaluation function needs access to an extra data structure, you can just pass it as additional argument, e.g. my_structure, when you call **evolve**. Then all functions and components will be able to access the data structure as **args["my_structure"]**. A useful, but not well-documented feature, is that **args["_ec"]** stores a reference to the instance of the currently running evolutionary algorithm, so that all of its parameters can be accessed from inside user-defined functions.

```
# example of code with inspyred
import inspyred
import time
...
rand = Random()
rand.seed(int(time())) # initialize pseudo-random number generator
...
my_ea = inspyred.ec.EvolutionaryComputation(rand) # generic evolutionary algorithm
my_ea.selector = inspyred.ec.selectors.tournament_selection # tournament selection
my_ea.variator = [inspyred.ec.variators.uniform_crossover, # genetic operators: crossover
inspyred.ec.variators.gaussian_mutation] # genetic operator: gaussian mutation
my_ea.replacer = inspyred.ec.replacers.plus_replacement # "plus" replacement
my_ea.observer = my_observer # defined by the user (not shown in this figure)
my_ea.terminator = inspyred.ec.terminators.generation_termination # stop after a number of generations
...
final_pop = my_ec.evolve(generator=my_generator, evaluator=my_evaluator, pop_size=100,
     max_generations=100,                    # parameter for generation_termination
     num_selected=2,                         # parameter for tournament_selection
     mutation_rate=0.25,                     # parameter for gaussian_mutation
     my_variable_1=3,                        # all variables that are not arguments of "evolve"
     my_variable_2=["list", "of", "strings"])   # will be stored as entries in the "args" dictionary
# Sort and print the best individual, who will be at index 0.
final_pop=sorted(final_pop, reversed=True)
print('Terminated due to {0}.'.format(my_ea.termination_cause))
print(final_pop[0])
...
def my_generator(random, args) :
    # function that generates random individuals at the beginning of the evolution
    ...
    return new_individual

def my_evaluator(candidates, args) :
    # evaluator: all individuals to be evaluated are in the list 'candidates'
    ...
    return fitness
```

Fig. 1 Example of code snippet using Inspyred. The components "my_generator", "my_evaluator" and "my_observer" are defined by the user, written as functions in the same file

While Inspyred natively supports concurrent evaluation of the individuals through multiple processes (fork()), it does not support multithreading: this could be an issue when evaluations require storing large data structures. However, it is always possible to write a custom multi-threaded "evaluator" for your problem, and manage the threads using the **threading** Python module (Inspyred does not specifically support either SIMD or GPUs).

Inspyred can be easily installed through pip (sudo pip install inspyred), or cloned from its GitHub repository. On Windows, it can be installed through Anaconda/Anaconda Cloud distributions.

At the moment Inspyred does not support genetic programming. Whereas genetic programming is fully supported in DEAP, its main competitor among Python modules.

Using Inspyred requires basic competence in Python. A general understanding of object-oriented programming is helpful, but not strictly necessary. I would recommend Inspyred both as a tool for fast deployment and testing of ideas for researchers, and for introducing graduate and undergraduate students to Evolutionary Algorithms, even if they had no previous experience with EAs. I personally used it for both purposes, with good results.

GitHub repository: https://github.com/inspyred/inspyred.
Documentation (including tutorials): https://pythonhosted.org/inspyred/.
Google groups discussion board: https://groups.google.com/forum/#!forum/inspyred.

Reference

1. J. Kim, S. Yoo, Software review: DEAP (Distributed Evolutionary Algorithm in Python) library. Genet. Program Evolvable Mach. **20**, 139 (2019). https://doi.org/10.1007/s10710-018-9341-4

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.