Genetics Algorithm in python

Jamiro Screti Mat: 656848

September 2019

Contents

1 Introduction

1.1 What is a genetic algorithm?

A genetic algorithm(**GA**) is a meta-heuristic inspired by Charles Darwin's theory of natural evolution (selection), that belongs into a larger class of evolutionary algorithms (**EA**).
The most common usage of GAs is to generate solutions to *optimization* and *search problems*, by relying on biologically-inspired operators, which are:

- **Mutation**: Used to maintain genetic diversity inside the population, altering one or more genes from its initial state, which may result in entirely different solutions from the previous ones;
- **Cross-over** : also called *recombination*, is used to combine the genetic information of two-parent when generating the new offspring. The new offspring can be generated by cloning an existing solution, but before being added into the new population, cloned solution are typically mutated;

- **Selection**: A stage of a generic GA in which the individual genomes are being chosen from the population, using a specifically created function called *fitness* , that determines which genomes can be chosen for later breeding (using the crossover operator).

1.2 How does a basic Genetic algorithm works

The most basic functioning of a GA is the following:

1. Individual in population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring than others
3. Genes from "fittest" parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent, thus each successive generation is more suited for their environment.
4. Repeat from step 2 until the condition to stop the algorithm are met (limit of generations reached, best solution reached(if known), extinction, endless evolution, etc..)

If we want to be more specific in the explaining of a GA, we need to clarify how each step work:

1. Step 1:
    - To define an initial population, random string is generated (the specific type of those strings does not matter when designing the algorithm, only when you need to implement the operators)
    - the number of generated string is set during the design phase
    - the number of individuals is constant during each iteration(generation) of the program, that means that the number of offspring individuals generated is similar to the number of the individuals that didn't survive
    - if the knowledge on the domain is enough, you can avoid the random generation and using a custom set of individuals.
2. Step 2:
    - you define a function that determines which individual is fit to survive the next generation and can mate with other individuals. This function takes the name of **fitness**. This function is applied to every individual on each generation and returns a value in the range of [0,1] for each individual, and this value represents the probability to survive and create an offspring in the next generation.

- you define another function, called **objective**, which determines where there is an individual that fits the criteria in the best way possible, and that is marked as a correct solution for the GA, therefore stopping it, if necessary.

3. Step 3:

   - During the selection process, a group of individuals that passed the fitness function are grouped inside a **mating pool**, where they are chosen in random pairs, for the reproduction part of the GA. There are several different reproduction methods, more or less plausible biologically. We are going to describe the most common selection algorithms:

     (a) proportional selection to fitness **(FPS)** : is the most common method used for selection : we denote with f_i the fitness associated to the i-Th element selected for reproduction with the chance of:

     $$p_i = \frac{f_i}{\sum_j}$$

     This method is by far the most plausible biologically speaking, but it has some problems :

     - **Premature convergence**: if an item has a fitness much greater then the average, it will be always selected
     - **Stagnation**: after some generations are passed, all elements will be electable for the selection, so no one gets discriminated against.
     - **Slow execution**: Since the algorithm has to calculate the fitness of every item each generation, the result will be a slow execution of the program itself.

     (b) **elite selection**: The elite selection method requires that at least one copy of the best individual is preserved through a generation. This speeds up the execution of the selection using its fast convergence to find a solution, but, if the fittest item has some dominant characters, it will be always chosen and you can get caught up in local highs.

     (c) **Ranked selection**: this method provides for the organization of individuals is based on fitness, so that $f_i \geq f_j$ for $i < j$

     A decreasing probability is assigned as a function of the position in ranking, regardless of the value of fitness, and The worst individuals are heavily penalized (or even discarded). This solves the problems regarding the elite selection, but, such a method, is not biological in the slightest and is very heavy on the computational side of the algorithm

     (d) **Tournament selection**: The tournament selection method involves the creation of a group of N (N > 2) individuals chosen at random: the ones with the most fitness gets chosen and can reproduce. This method is similar to the rank selection, but it doesn't need any kind of ordering between the items.

- After the selection method, the resulting items inside the mating pool get paired and a mating algorithm is applied to them.

Now we are going to talk about these concepts and ideas using python.

1.3 How the research was carried out

The information on this research are obtained from different sources, such as Wikipedia (for the first chapter), and the documentation for these libraries available on GitHub, PyPI, and their respective libraries sites.

2 Deap

2.1 Description of the library

Developed by François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, Christian Gagné at Université Laval DEAP is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It seeks to make algorithms explicit and data structures transparent. It works in perfect harmony with parallelization mechanisms such as multiprocessing and SCOOP (Scalable Concurrent Operations in Python, allowing parallel programming on various environments).

2.2 License of use

DEAP is licensed under the GNU Lesser General Public License v3.0

2.3 Features

- Genetic algorithm using any imaginable representation
- Genetic programming using prefix trees

  Loosely typed, Strongly typed

  Automatically defined functions
- Evolution strategies (including CMA-ES)
- Multi-objective optimization (NSGA-II, NSGA-III, SPEA2, MO-CMA-ES)
- Co-evolution (cooperative and competitive) of multiple populations
- Parallelization of the evaluations (and more) Hall of Fame of the best individuals that lived in the population

- Checkpoints that take snapshots of a system regularly
- Benchmarks module containing most common test functions
- Genealogy of evolution (that is compatible with NetworkX)
- Examples of alternative algorithms: Particle Swarm Optimization, Differential Evolution, Estimation of Distribution Algorithm

2.4 Python compatibility (2.x and 3.x)

The most basic features of DEAP requires Python2.6. To combine the toolbox and the multiprocessing module Python2.7 is needed for its support to pickle partial functions. Since version 0.8, DEAP is compatible out of the box with Python 3. The installation procedure automatically translates the source to Python 3 with 2to3.

2.5 Dependencies

- CMA-ES requires Numpy
- matplotlib for visualization of results as it is fully compatible with DEAP's API.

2.6 Genetic operators available

DEAP provides a plethora of genetic operators for each phase of the algorithm (initialization, selection, cross-over, mutation, and even migration between 2 populations). For a complete list of the operators you can find it here.

2.7 How genotypes are represented

DEAP does not come with generic representations of genotypes, because there is an extremely large variety of individuals that can be represented, making it impossible to create all types, but instead you can create custom individuals using their functions. Here's a simple list:

- List of floats
- Permutations
- Prefix tree of mathematical expressions
- Evolution strategies(2 arrays: 1 for the population, 1 for the possible mutations)
- Custom one created by the user.

2.8 exploitation of parallelism

DEAP implements and exploit the concept of parallelism using :

- *Scalable Concurrent Operations in Python* **(SCOOP)**,a distributed task module allowing concurrent parallel programming on various environments.
- Multiprocessing Module available in python

2.9 Guides and tutorials

DEAP provides a detailed documentation concerning the use of the library, as well a section concerning the benchmark algorithms

2.10 learning curve in using the library

DEAP is quite a straightforward and easy to understand library, thanks to the documentation provided by the authors, but it requires a good knowledge of the python language to begin with since it requires to create a custom population and fitness functions most of the times.

2.11 Conda environment

An Anaconda environment will be provided named *DEAP-CONDA*, using python 3.7.4 (the latest version since the project was created), implementing the knapsack algorithm found on the docs.

2.12 Publications

- François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau and Christian Gagné, "DEAP – Enabling Nimbler Evolutions", SIGEVOlution, vol. 6, no 2, pp. 17-26, February 2014. Link
- Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau and Christian Gagné, "DEAP: Evolutionary Algorithms Made Easy", Journal of Machine Learning Research, vol. 13, pp. 2171-2175, jul 2012. Link
- François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau and Christian Gagné, "DEAP: A Python Framework for Evolutionary Algorithms", in !EvoSoft Workshop, Companion proc. of the Genetic and Evolutionary Computation Conference (GECCO 2012), July 07-11 2012. Link

2.13 Projects using this library

- Ribaric, T., & Houghten, S. (2017, June). Genetic programming for improved cryptanalysis of elliptic curve cryptosystems. In 2017 IEEE Congress on Evolutionary Computation (CEC) (pp. 419-426). IEEE.

- Ellefsen, Kai Olav, Herman Augusto Lepikson, and Jan C. Albiez. "Multi-objective coverage path planning: Enabling automated inspection of complex, real-world structures." Applied Soft Computing 61 (2017): 264-282.
- S. Chardon, B. Brangeon, E. Bozonnet, C. Inard (2016), Construction cost and energy performance of single family houses : From integrated design to automated optimization, Automation in Construction, Volume 70, p.1-13.
- B. Brangeon, E. Bozonnet, C. Inard (2016), Integrated refurbishment of collective housing and optimization process with real products databases, Building Simulation Optimization, pp. 531–538 Newcastle, England.
- Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore (2016). Automating biomedical data science through tree-based pipeline optimization. Applications of Evolutionary Computation, pages 123-137.
- Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore (2016). Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. Proceedings of GECCO 2016, pages 485-492.
- Van Geit W, Gevaert M, Chindemi G, Rössert C, Courcol J, Muller EB, Schürmann F, Segev I and Markram H (2016). BluePyOpt: Leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience. Front. Neuroinform. 10:17.
- Lara-Cabrera, R., Cotta, C. and Fernández-Leiva, A.J. (2014). Geometrical vs topological measures for the evolution of aesthetic maps in a rts game, Entertainment Computing,
- Macret, M. and Pasquier, P. (2013). Automatic Tuning of the OP-1 Synthesizer Using a Multi-objective Genetic Algorithm. In Proceedings of the 10th Sound and Music Computing Conference (SMC). (pp 614-621).
- Fortin, F. A., Grenier, S., & Parizeau, M. (2013, July). Generalizing the improved run-time complexity algorithm for non-dominated sorting. In Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference (pp. 615-622). ACM.
- Fortin, F. A., & Parizeau, M. (2013, July). Revisiting the NSGA-II crowding-distance computation. In Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference (pp. 623-630). ACM.
- Marc-André Gardner, Christian Gagné, and Marc Parizeau. Estimation of Distribution Algorithm based on Hidden Markov Models for Combinatorial Optimization. in Comp. Proc. Genetic and Evolutionary Computation Conference (GECCO 2013), July 2013.

- J. T. Zhai, M. A. Bamakhrama, and T. Stefanov. ”Exploiting Just-enough Parallelism when Mapping Streaming Applications in Hard Real-time Systems”. Design Automation Conference (DAC 2013), 2013.
- V. Akbarzadeh, C. Gagné, M. Parizeau, M. Argany, M. A Mostafavi, ”Probabilistic Sensing Model for Sensor Placement Optimization Based on Line-of-Sight Coverage”, Accepted in IEEE Transactions on Instrumentation and Measurement, 2012.
- M. Reif, F. Shafait, and A. Dengel. ”Dataset Generation for Meta-Learning”. Proceedings of the German Conference on Artificial Intelligence (KI’12). 2012.
- M. T. Ribeiro, A. Lacerda, A. Veloso, and N. Ziviani. ”Pareto-Efficient Hybridization for Multi-Objective Recommender Systems”. Proceedings of the Conference on Recommanders Systems (!RecSys’12). 2012.
- M. Pérez-Ortiz, A. Arauzo-Azofra, C. Hervás-Martínez, L. García-Hernández and L. Salas-Morera. ”A system learning user preferences for multiobjective optimization of facility layouts”. Pr,oceedings on the Int. Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO’12). 2012.
- Lévesque, J.C., Durand, A., Gagné, C., and Sabourin, R., Multi-Objective Evolutionary Optimization for Generating Ensembles of Classifiers in the ROC Space, Genetic and Evolutionary Computation Conference (GECCO 2012), 2012.
- Marc-André Gardner, Christian Gagné, and Marc Parizeau, ”Bloat Control in Genetic Programming with Histogram-based Accept-Reject Method”, in Proc. Genetic and Evolutionary Computation Conference (GECCO 2011), 2011.
- Vahab Akbarzadeh, Albert Ko, Christian Gagné, and Marc Parizeau, ”Topography-Aware Sensor Deployment Optimization with CMA-ES”, in Proc. of Parallel Problem Solving from Nature (PPSN 2010), Springer, 2010.
- DEAP is used in TPOT, an open source tool that uses genetic programming to optimize machine learning pipelines.
- DEAP is also used in ROS as an optimization package.
- DEAP is an optional dependency for PyXRD, a Python implementation of the matrix algorithm developed for the X-ray diffraction analysis of disordered lamellar structures.
- DEAP is used in glyph, a library for symbolic regression with applications to MLC.

2.14 Sources

All information regarding this library comes from:

- GitHub
- Read the docs.io

3 Pyvolution

3.1 Description of the library

Evolutionary Algorithms Framework, written purely in python 2.7, developed by Ashwin Panchapakesan.

3.2 License of use

Pyvolution runs under the Apache License, Version 2.0.

3.3 Features

This library features the most basic functions that are required to implement a GA. In the next paragraphs, there will be a more detailed explanation of all of the features available.

3.4 Python compatibility (2.x and 3.x)

Pyvolution requires python 2.7 and will not work with python 2.6. It can easily be transformed into python 2.6 compliant code. Most of the incompatibilities come from list/dict comprehension expressions, however, it is not compatible with python 3.x

3.5 Dependencies

The requirements for this library are:

- PyGame (and all its dependencies)

  Used in visualization of the included TSP solver

- pycontract (contract)

  Used for contract checking (only when testmode is True)

3.6 Genetic operators available

The only genetic operators available on this library, by default, are :

- Tournament Select
- Roulette Wheel Select

  if you are using the Roulette wheel, you have access into another function that returns a list of 3-tuples (Individual, lowerBound, UpperBound) for each individual

- crossover and mutation are not implemented since they change based on the GA

3.7 How genotypes are represented

- The genotypes are represented using a class. The original implementation treats each chromosome differently. Therefore, all the chromosomes of an individual are maintained in a list as opposed to a set.
- Also there are implemented methods to the individual class that help identify an individual, define its hash, test its equality to another individual instance, etc
- the population is implemented using two functions:

  chromGenfuncs: a list of functions. This function is based on the idea here is that each individual in the population is made up of C chromosomes. These C chromosomes are generated independently of each other for each individual in the initial population. Therefore, there must be exactly C functions listed in chromGenfuncs. The i-th function in chromGenfuncs will be used to generate the i-th chromosome of each individual

  chromGenParams: a list of tuples. There should be exactly as many tuples in this list, as there are functions in chromGenfuncs.

3.8 exploitation of parallelism

Pyvolution has no internal method to exploit parallelism, however, since it is compatible with Python 2.7+, it can use SCOOP to work in parallel.

3.9 Guides and tutorials

The official docs of Pyvolution show only an example of genetic algorithm, the TSP (traveling salesman problem), the name comes from its most typical representation: given a set of cities, and known the distances between each pair of them, find the minimum distance journey that a salesman must follow to visit all the cities once and only once. When writing this document, the official documentation does not contain other tutorials.

3.10 learning curve in using the library

Since this library is poorly documented, its use will be difficult for anyone not familiar with the python language, and for anyone looking to implement any algorithm using this library. However, that does not take away from the fact of its stability and practicality of the founding ideals, which this library is based on

3.11 Conda environment

The Conda environment created to test this library is named *CONDA-PYVOLUTION* and implements the TSP.

3.12 Publications

- Hydrodynamic modelling of coastal inundation Link
- The Python-OpenDSS co-simulation for the evolutionary multiobjective optimal allocation of the single tuned passive power filters Link
- Investigating the effectiveness and optimal spatial arrangement of low-impact development facilities Link

3.13 Projects using this library

- Hydrodynamic modelling of coastal inundation
- Investigating the effectiveness and optimal spatial arrangement of low-impact development facilities

3.14 Sources

- GitHub
- Google Scholar

4 Pyeasyga

4.1 Description of the library

Developed by Ayodeji Remi-Omosowon, Pyeasyga is a simple and easy-to-use implementation of a Genetic Algorithm library in Python, that provides a simple interface to the power of Genetic Algorithms (GAs), so that you don't have to have expert GA knowledge to use it.

4.2 License of use

Pyeasyga is under the BSD License (BSD)

4.3 Features

Implementation of all basic ga functionality

4.4 Python compatibility (2.x and 3.x)

Pyeasyga uses Python 3.4 without breaking Python 2 compatibility, however, in the test was used python 3.7

4.5 Dependencies

- Sphinx → 1.2.2
- sphinx-rtd-theme → 0.1.6
- coverage → 3.7.1
- flake8 → 2.2.0
- tox → 1.7.1
- wheel → 0.23.0
- six → 1.9.0

4.6 Genetic operators available

- crossover
- mutation
- random selection
- tournament selection
- rank population
- elitism
- best individual

4.7 How genotypes are represented

Pyeasyga documentation does not directly tell you how you can represent the genotypes, however, the examples provided they use lists, matrix, sets to work with, whereby, as long you can tweak the default functions provided by this library, you can represent the genotypes any way you want.

4.8 exploitation of parallelism

pyvolution has no internal method to exploit parallelism, however, since it is compatible with Python 2.7+, it can use SCOOP to work in parallel.

4.9 Guides and tutorials

Pyeasyga provides some practical examples of how to use this library, instead of giving you the specification of all the features it provides.

4.10 learning curve in using the library

Overall, the setup and basic testing of this library was rather easy, because it is created with the idea of giving a more practical approach into the world of gas, instead of giving a theoretical one. It is recommended for anyone entering the world of genetic algorithms, even without much knowledge of the domain.

4.11 Publications

- Applying computational intelligence to a real-world container loading problem in a warehouse environment Link
- Considerations on using genetic algorithms for the 2D bin packing problem: A general model and detected difficulties Link
- SOPHIA: Online Reconfiguration of Clustered NoSQL Databases for Time-Varying Workloads Link

4.12 Projects using this library

- Applying computational intelligence to a real-world container loading problem in a warehouse environment
- Considerations on using genetic algorithms for the 2D bin packing problem: A general model and detected difficulties
- SOPHIA: Online Reconfiguration of Clustered NoSQL Databases for Time-Varying Workloads

4.13 Conda environment

The environment created for the user of this library is called *CONDA-EASY* and implements the 1-dimensional knapsack algorithm

4.14 Source of origin

- GitHub
- Read the docs
- Google Scholar

5 Inspyred

5.1 Description of the library

Developed by Aaron Garrett, the inspyred library grew out of insights from Ken de Jong's book "Evolutionary Computation: A Unified Approach." The goal of the library is to separate problem-specific computation from algorithm-specific computation.

5.2 License of use

Inspyred is distributed under the MIT License.

5.3 Features

- Evolutionary Computation

  ec – Evolutionary computation framework

  emo – Evolutionary multiobjective optimization

  analysis – Optimization result analysis

  utilities – Optimization utility functions

- Operators

  archivers – Solution archival methods

  evaluators – Fitness evaluation methods

  generators – Solution generation methods

  migrators – Solution migration methods

  observers – Algorithm monitoring methods

  replacers – Survivor replacement methods

  selectors – Parent selection methods

  terminators – Algorithm termination methods

  variators – Solution variation methods

- Swarm Intelligence

  swarm – Swarm intelligence

  topologies – Swarm topologies

- Benchmark Problems

  benchmarks – Benchmark optimization functions

  Single-Objective Benchmarks

  Multi-Objective Benchmarks

  Discrete Optimization Benchmarks

5.4 Python compatibility (2.x and 3.x)

Requires at least Python 2.6+ or 3+.

5.5 Dependencies

- Numpy and Pylab are required for several functions in the observers for the evolutionary computation framework.
- Pylab and Matplotlib are required for several functions in the analysis of the evolutionary computation framework.
- Parallel Python (pp) is required if you need to work in parallel

5.6 Genetic operators available

- Selection

  default (all individuals)

  truncation (only the best individuals)

  uniform (random individuals)

  fitness proportionate

  rank

  tournament

- re-placer

  default (no replacement)

  truncation (the entire population become the individual, between population and offspring, keeping the existing population size fixed)

  steady-state replacement (the offspring replace the least fit individuals in the existing population, even if those offspring are less fit than the individuals that they replace.)

  generational (Performs generational replacement with optional weak elitism)

  random (random replacement with optional weak elitism)

  plus (the entire existing population is replaced by the best population-many elements from the combined set of parents and offspring)

  comma (the entire existing population is replaced by the best population-many elements from the offspring. This function assumes that the size of the offspring is at least as large as the original population. Otherwise, the population size will not be constant)

  crowding (the members of the population are replaced one-at-a-time with each of the offspring, based on the distance between a random sample of the population, and the closest individual to the current offspring, if the offspring is better)

simulated annealing schedule replacement

NGSA-II sorting

Pareto Archived Evolution Strategy replacement

- Cross-over

  custom

  n-point

  uniform (coin flip to choose if the offspring gets the ”mom” or ”dad” element, with optional bias)

  partially matched (cross-over more likely to happen if the individual are from the same generation)

  arithmetic (is similar to a generalized weighted averaging of the candidate elements, with the option of shifting the weight)

  blend (similar to arithmetic, with a bit of mutation)

  heuristic (similar to the update rule used in particle swarm optimization)

  simulated binary (following the implementation in NSGA-II)

  Laplace (following the implementation specified in (Deep and Thakur, ”A new crossover operator for real coded genetic algorithms,” Applied Mathematics and Computation, Volume 188, Issue 1, May 2007, pp. 895–911))

- Variation

  default (no variation)

- Mutation

  Custom

  Bit flip

  random reset (if the solutions are composed o discrete values)

  scramble

  inversion

  gaussian

  non-uniform (performs nonuniform mutation as specified in (Michalewicz, ”Genetic Algorithms + Data Structures = Evolution Programs,” Springer, 1996))

5.7 How genotypes are represented

Inspyred re-present individuals as an object, which is given to a function that generates the population, since all choices of which individual should survive is based on the fitness function. However, giving the nature of the possible combination of values that an individual can represent, you cannot apply every cross-over/mutation algorithm, because of the nature of values that these operators work with.

5.8 exploitation of parallelism

Inspyred fully support parallelism when use the parallel python (*pp*) library

5.9 Guides and tutorials

The documentation provides a list of the most common examples, such as :

- Schwefel benchmark (GA);
- Rosenbrock benchmark (ES);
- Sphere benchmark (SA) ;
- Griewank benchmark (DEA);
- Rastrigin benchmark (EDA);
- Kursawe multiobjective benchmark (PAES);
- Kursawe multiobjective benchmark (NGSA-II);
- Ackley benchmark (PSO);
- TSP benchmark (ACS);
- knapsack problem ;

Each python file is documented thoroughly, specifying how each function works and how it relates to the overall algorithm, making rather easy the choice of which operator/s should be used each generation.

5.10 learning curve in using the library

The overall usage of this library, is kinda simple, given the fact that you already are enough knowledgeable on how python work with classes and math libraries.

5.11 Conda environment

The environment created to test this library is called "CONDA-INSP", which implements the Schwefel benchmark algorithm, solved using GA, and, an optimization on a Gravity Slingshot for a space probe designed to travel around the moon and return to earth

5.12 Publications

- Population Generation from Statistics Using Genetic Algorithms with MIST+ INSPYRED Link

5.13 Projects using this library

- Population Generation from Statistics Using Genetic Algorithms with MIST+ INSPYRED

5.14 Source of origin

- Documentation
- Google Scholar
- GitHub

6 GAFT

6.1 Description of the library

Developed by Shao Zhengjiang, GAFT (**Genetic Algorithm Framework in pyThon**) is a general Python Framework for genetic algorithm computation. It provides built-in genetic operators for target optimization and plugin interfaces for users to define your genetic operators and on-the-fly analysis for algorithm testing.

6.2 License of use

GAFT is published under the GPLv3 license.

6.3 Features

- MPI (Message Passage Interface, for parallelism exploitation)
- Initialization
- Selection
- Crossover
- Mutation
- Storing and analysis information on the fly

6.4 Python compatibility (2.x and 3.x)

GAFT requires Python version 3.x (Python 2.x is not supported).

6.5 Dependencies

If you want to run your optimization flow in parallel for acceleration, you need to install an implementation of MPI on your machine and then mpi4py package.

6.6 Genetic operators available

- MPI (Message Passage Interface, for parallelism exploitation)
- Initialization
  - Binary individual
  - Decimal encoding
  - Custom
- Selection
  - Roulette wheel
  - Tournament
  - Rank
  - Exponential ranking
- Crossover
  - Uniform
- Mutation
  - Flip bit
- Storing information on the fly
  - Fitness
  - Population finished
  - Population during execution
  - Custom

6.7 How genotypes are represented

The genotypes are represented using a class, that, when it passed into a function, it creates an individual and then a population. The default implementations of individuals consist of binary and decimal genotypes, although, GAFT only needs an object to create an individual, so it supports custom types, but it is necessary to create every function needed by the class too.

6.8 exploitation of parallelism

GAFT uses a utility class named *mpiutil*, for parallelizing the Genetic Algorithm by using MPI interfaces in a distributed MPI environment.

6.9 Guides and tutorials

The GAFT documentation it's well-stocked in describing its functioning, however, there are only two examples given to test the framework, which is a 1D global max, and one 1D optimization (global min)

6.10 learning curve in using the library

GAFT has a limited number of built-in functions, making it rather easy to remember their uses, but, on the other hand, other than some basic functions, it requires an implementation starting from of every other common function for each phase of the GA

6.11 Publications

No publications found on google scholar and Research gate

6.12 Projects using this library

No publications found on google scholar and Research gate

6.13 Conda environment

The environment used to test the Framework is called *CONDA-GAFT*, implementing the example given inside the docs.

6.14 Source of origin

- Read the docs
- GitHub

7 Pyevolve

7.1 Description of the library

Developed by Christian Perone, Pyevolve is an open-source framework for genetic algorithms. The initial long-term goal of the project was to create a complete and multi-platform framework for genetic algorithms in pure python.

7.2 License of use

The framework is entirely open-source and it is licensed upon a very permissive PSF2-like license.

7.3 Features

- Evolutionary algorithms (genetic programming and gas)
- Graphical plotting
- Database and visualization adapters
- Interaction support
- Basic standard features for genetic operators, scaling, selection, etc

7.4 Python compatibility (2.x and 3.x)

the framework was written in pure Python, so it can run on Mac, Windows, Linux platforms, and on any portable devices where Python is available (e.g., the Sony PlayStation Portable Symbian OS based cellphones)

7.5 Dependencies

- Biopython
- Scipy
- Numpy ≥ 1.7

7.6 Genetic operators available

- Initialization

  1D binary string

  2D binary string

  1D list

  2D list

  Tree

  Custom

- Crossover
  - 1D binary string
    single point
    2 point
    uniform
  - 1D list
    single point
    2 point
    uniform
    order

- 2D list
  - single vertical point
  - single horizontal point
  - uniform
- Tree
  - sub-tree
  - strict sub-tree

- Mutation
  - 1D binary string
    - swap
    - flip
  - 1D list
    - swap
    - integer-Gaussian
    - real-Gaussian
    - Custom
  - 2D list
    - swap
    - integer-Gaussian
    - real-Gaussian
    - Custom
  - Tree
    - swap
    - integer-Gaussian
    - real-Gaussian
    - integer-range
    - real-range
- Selection
  - Linear
  - Sigma truncation
  - Power-law
  - Boltzmann
  - Raw
- Scaling
  - Rank
  - Uniform
  - Tournament
  - Roulette wheel

- Pausing

7.7 How genotypes are represented

The framework includes the classical representations used in genetic algorithms (1D and 2D Binar Strings) and several other representations (1D and 2D Lists, and Trees). What defines the data type of the representation is the genetic algorithm initialization function. Pyevolve provides ready-to-use built-in initialization routines for integer numbers, real numbers, and user-defined alleles. Besides the few representations available in the distribution, the flexibility of Python allows the creation of new representations by extending the existing ones.

7.8 exploitation of parallelism

Pyevolve has built-in support for the usage of multiple cores, alternating the CPU cores during the fitness evaluation, for each individual in the population.

7.9 Guides and tutorials

Pyevolve documentation is very well made, explaining every single function built-in, as well as giving a great amount of examples, varying from the simple algorithms into the benchmark problems.

7.10 learning curve in using the library

This framework is one of the most documented on the market, making it one of the simplest to use, and, because of that, you can find a great amount of research papers and support during each phase of learning/using the framework, to help you master all the features it has to offer.

7.11 Publications

- Pyevolve: a Python open-source framework for genetic algorithms Link
- Effects of Optical Parameters in a Free-Electron Laser Oscillator Link
- Using a Meta-GA for parametric optimization of simple gas in the computational chemistry domain Link
- PyEvolve: a toolkit for statistical modelling of molecular evolution Link

7.12 Projects using this library

- Effects of Optical Parameters in a Free-Electron Laser Oscillator
- Benchmark-guided HDF5 Application Tuning

7.13 Conda environment

The Conda environment used to test this framework is called *CONDA-PYEVOLVE*, implementing the tsp.

7.14 Source of origin

- Research gate
- Git Hub
- Sourceforge
- Project blog

8 Summary table

8.1 Criteria

- Population sizes: (50, 100, 150, 200, 250, 300, 350, 400).
- Population replacement models: a) Generational Replacement Model
- Crossover probability: [0.6–0.95] with step 0.05.
- Mutation probability: 1=2l where l the individual's alphanumerical string.
- Crossover mechanism: uniform crossover.
- Algorithm ending criteria: the executions stop only 100 generations
- Pseudorandom generators: random and numpy libraries
- Mutation mechanism: Gaussian.
- Selection: tournament selection.
- Fitness function: linear ranking.

8.2 Functions

Id	Function Name	Function	Limits
F1	Sphere	$f(\mathbf{x}) = \sum_{i=1}^{2} x_i^2$	[-5.12,5.12]
F2	Rosenbrok	$f(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	[-2.048, 2.048]
F3	Step (flat surfaces)	$f(\mathbf{x}) = \sum_{i-1}^{5}(Int.(x_i))$	[-5.12,5.12]
F4	Quartic	$f(\mathbf{x}) = \sum_{i=1}^{30}((ix_i^4) + Gauss(0,1))$	[-1.28,1.28]
F5	Shekel's Foxholes	$f(\mathbf{x}) = \sum_{i=1}^{25} \frac{1}{c_i + \sum_{j=1}^{2}(x_j - a_{ij})^2}$	[-6.5336,6.5336]
F6	Schwefel	$f(\mathbf{x}) = 418.9828872724339 \cdot N - \sum_{i=1}^{10} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	[-500,500]
F7	Rastrigin	$f(\mathbf{x}) = 10N + \sum_{i=1}^{2} 0x_i^2 - 10\cos(2\pi x_i)$	[-5.12,5.12]
F8	Griekwank	$f(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{10} x_i^2 - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	[-600,600]