GAFT Documentation

Release GAFT

Zhengjiang Shao

Dec 23, 2018

Contents:

CHAPTER 1

Introduction

Welcome to the *GAFT* documentation!

GAFT is a general genetic algorithm framwork written in Python with MPI parallelization under the GPLv3 license. It's an acronym for Genetic Algorithm Framework for Python.

You can always find the latest stable version of the program here: https://github.com/pytlab/gaft

This documentation describes version 0.5.6

1.1 Why Genetic Algorithm?

In contrast to those local search methods, genetic algorithms which are categorized as global search heuristics are a particular class of evolutionary algorithm (EA) that utilizes techniques inspired by evolutionary biology ideas such as inheritance, mutation, selection and crossover. Without calculating derivatives, genetic algorithms can be used in many domains to find the optimal solutions for complex problems such as biology, engineering, computational chemistry, computer science and social science. Especially in a case where the mathematical data is available and answers are available but the formula that joins the data to the answers is missing, at this time, a genetic algorithm can be used to 'evolve' an expression tree to create a very close fit to the data, for example, the complex hyper parameters optimization of a mathematical model. At the same time, genetic algorithms have relative fixed iteration process and large space for algorithm adjustment by genetic operator improvement. Therefore, genetic algorithm is one of the most appropriate methods to construct a general optimization framework for more realistic applications in different fields

1.2 Why *GAFT*?

Optimization problem that needs is a common problem researchers meet in computational physics and chemistry field. It is a great challenge to create a high-efficient program that are general and easy-to-use enough to be used directly for optimizing different target problems and are customizable enough to help researcher to develop new algorithm and run tests.

To this end, we present a Python general Genetic Algorithm framework named GAFT which provides flexible and customizable API to help researchers in various fields to apply genetic algorithm optimization flow to their own targets.

CHAPTER 2

Installation

2.1 Serial version

If you only want to use GAFT to run some simple mathmetical optimization without parallel acceleration, you can install GAFT without any prerequisite.

2.1.1 Via pip (Recommended)

```
pip install gaft
```

2.1.2 From source

```
git clone --recursive https://github.com/PytLab/gaft.git
cd gaft
python setup.py install
```

2.2 Parallel version

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.

MPI implementation for different platforms: 1. MPICH 2. OpenMPI 3. Microsoft MPI

You also install mpi4py explicitly:

```
pip install mpi4py
```

Then you can install GAFT in the same way with the serial version.

2.3 For developers

GAFT also provides unit tests for developers, if you have cloned the repository, just

```
python setup.py test
```

or

```
python -m gaft.tests.test_all
```

CHAPTER 3

Build C++ API

3.1 Create dir for build

```
cd gasol
mkdir build
cd build
```

3.2 Build

- Serial version

```
cmake ..
make
```

- MPI parallel version

```
export CXX=/<mpi_path>/mpicxx
cmake -DMPI=true ..
make
```

3.3 Run test

```
make unittest
./unittest/unittest
```

CHAPTER 4

API

4.1 gaft.engine

Genetic Algorithm engine definition

class gaft.engine.**GAEngine**(*population*, *selection*, *crossover*, *mutation*, *fitness=None*, *analysis=None*)

Bases: object

Class for representing a Genetic Algorithm engine. The class is the central object in GAFT framework for running a genetic algorithm optimization. Once the population with individuals, a set of genetic operators and fitness function are setup, the engine object unites these informations and provide means for running a genetic algorthm optimization.

Parameters

- **population** (gaft.components.Population) – The Population to be reproduced in evolution iteration.
- **selection** (gaft.plugin_interfaces.operators.Selection) – The Selection to be used for individual seleciton.
- **crossover** (gaft.plugin_interfaces.operators.Crossover) – The Crossover to be used for individual crossover.
- **mutation** (gaft.plugin_interfaces.operators.Mutation) – The Mutation to be used for individual mutation.
- **fitness** (*function*) – The fitness calculation function for an individual in population.
- **analysis** (OnTheFlyAnalysis list) – All analysis class for on-the-fly analysis.

analysis_register(*analysis_cls*)

A decorator for analysis regsiter.

Parameters analysis_cls (gaft.plugin_interfaces.OnTheFlyAnalysis) – The analysis to be registered

dynamic_linear_scaling(*target='max'*, *ksi0=2*, *r=0.9*)
A decorator constructor for fitness dynamic linear scaling.

Parameters

- **target** (*str*) – The optimization target, maximization or minimization possible value: 'min' or 'max'
- **ksi0** (*float*) – Initial selective pressure adjustment value, default value is 2
- **r** (*float in range [0.9, 0.999]*) – The reduction factor for selective pressure adjustment value, ksi^(k-1)*r is the adjustment value for generation k, default value is 0.9

Note: Dynamic Linear Scaling:

For maximizaiton, $f' = f(x) - \min f(x) + \xi^k$, k is generation number.

---

fitness_register(*fn*)
A decorator for fitness function register.

Parameters **fn** (*function*) – Fitness function to be registered

linear_scaling(*target='max'*, *ksi=0.5*)
A decorator constructor for fitness function linear scaling.

Parameters

- **target** (*str*) – The optimization target, maximization or minimization, possible value: 'max', 'min'
- **ksi** (*float*) – Selective pressure adjustment value.

Note:

Linear Scaling:

1. $arg \max f(x)$, then the scaled fitness would be $f - \min f(x) + \xi$
2. $arg \min f(x)$, then the scaled fitness would be $\max f(x) - f(x) + \xi$

---

minimize(*fn*)
A decorator for minimizing the fitness function.

Parameters **fn** (*function*) – Original fitness function

run(*ng=100*)
Run the Genetic Algorithm optimization iteration with specified parameters.

Parameters **ng** (*int*) – Evolution iteration steps (generation number)

gaft.engine.**do_profile**(*filename*, *sortby='tottime'*)
Constructor for function profiling decorator.

4.2 gaft.mpiutil

A high-level utility class for parallelizing Genetic Algorithm by using MPI interfaces in distributed MPI environment.

gaft.mpiutil.**master_only**(*func*)
Decorator to limit a function to be called only in master process in MPI env.

class gaft.mpiutil.**MPIUtil**

barrier()
Block until all processes in the communicator have reached this routine

bcast(*data*)
Broadcast data to MPI processes

Parameters **data** (*any Python object*) – Data to be broadcasted

is_master
If current process is the master process

merge_seq(*seq*)
Gather data in sub-process to root process.

Parameters **seq** (*any Python object list*) – Sub data sequence for current process

Returns Merged data sequence from all processes in a communicator

Return type any Python object list

rank
Get the rank of the calling process in the communicator

size
Get the size of the group associated with a communicator

split_seq(*sequence*)
Split the sequence according to rank and processor number.

Parameters **sequence** (*any Python object list*) – Data sequence to be splitted

Returns Sub data sequence for current process

Return type any Python object list

split_size(*size*)
Split a size number(int) to sub-size number.

Parameters **size** (*int*) – The size number to be splitted.

Returns Sub-size for current process

Return type int

4.3 gaft.components.individual

class gaft.components.individual.**DecretePrecision**
Bases: object

Descriptor for individual decrete precisions.

class gaft.components.individual.**IndividualBase**(*ranges*, *eps*)
Bases: object

Base class for individuals.

Parameters

- **ranges** (*tuple list*) – value ranges for all entries in solution.
- **eps** (*float or float list (with the same length with ranges)*) – decrete precisions for binary encoding, default is 0.001.

clone()
Clone a new individual from current one.

decode()
NEED IMPLIMENTATION

Convert chromsome sequence to solution.

Returns The solution vector

Return type list of float

encode()
NEED IMPLIMENTATION

Convert solution to chromsome sequence.

Returns The chromsome sequence

Return type list of float

init(*chromsome=None*, *solution=None*)
Initialize the individual by providing chromsome or solution.

Parameters

- **chromsome** (*list of (float / int)*) – chromesome sequence for the individual
- **solution** (*list of float*) – the variable vector of the target function.

Note: If both chromsome and solution are provided, only the chromsome would be used. If neither is provided, individual would be initialized randomly.

class gaft.components.individual.**SolutionRanges**
Bases: object

Descriptor for solution ranges.

4.4 gaft.components.binary_individual

Module for Individual with binary encoding.

class gaft.components.binary_individual.**BinaryIndividual**(*ranges*, *eps=0.001*)
Bases: *gaft.components.individual.IndividualBase*

Class for individual in population. Random solution will be initialized by default.

Parameters

- **ranges** (*tuple list*) – value ranges for all entries in solution.
- **eps** (*float or float list (with the same length with ranges)*) – decrete precisions for binary encoding, default is 0.001.

static binarize(*decimal*, *eps*, *length*)
Helper function to convert a float to binary sequence.

Parameters

- **decimal** (*float*) – the decimal number to be converted
- **eps** (*float*) – the decrete precision of binary sequence
- **length** (*int*) – the length of binary sequence.

clone()

Clone a new individual from current one.

static decimalize(*binary*, *eps*, *lower_bound*)

Helper function to convert a binary sequence back to decimal number.

Parameters

- **binary** (*list of int*) – The binary list to be converted
- **eps** (*float*) – the decrete precision of binary sequence
- **lower_bound** (*float*) – the lower bound for decimal number

decode()

Decode gene sequence to solution of target function.

encode()

Encode solution to gene sequence in individual using different encoding.

init(*chromsome=None*, *solution=None*)

Initialize the individual by providing chromsome or solution.

Parameters

- **chromsome** (*list of (float / int)*) – chromesome sequence for the individual
- **solution** (*list of float*) – the variable vector of the target function.

Note: If both chromsome and solution are provided, only the chromsome would be used. If neither is provided, individual would be initialized randomly.

4.5 gaft.components.decimal_individual

Definition of individual class with decimal encoding.

class gaft.components.decimal_individual.**DecimalIndividual**(*ranges*, *eps=0.001*)

Bases: *gaft.components.individual.IndividualBase*

Individual with decimal encoding.

Parameters

- **ranges** (*tuple list*) – value ranges for all entries in solution.
- **eps** (*float or float list (with the same length with ranges)*) – decrete precisions for binary encoding, default is 0.001.

clone()

Clone a new individual from current one.

decode()

Decode gene sequence to decimal solution

encode()

Encode solution to gene sequence

init(*chromsome=None*, *solution=None*)

Initialize the individual by providing chromsome or solution.

Parameters

- **chromsome** (*list of (float / int)*) – chromesome sequence for the individual
- **solution** (*list of float*) – the variable vector of the target function.

Note: If both chromsome and solution are provided, only the chromsome would be used. If neither is provided, individual would be initialized randomly.

4.6 gaft.components.population

class gaft.components.population.**Individuals**(*name*)

Bases: object

Descriptor for all individuals in population.

Note: Use this descriptor to ensure the individual related flags can be updated when the population indivduals are changed.

class gaft.components.population.**Memoized**(*func*)

Bases: object

Descriptor for population statistical varibles caching.

class gaft.components.population.**Population**(*indv_template*, *size=100*)

Bases: object

Class for representing population in genetic algorithm.

Parameters

- **indv_template** (gaft.components.IndividualBase) – A template individual to clone all the other individuals in current population.
- **size** (*int*) – The size of population, number of individuals in population.

best_indv(*fitness*)

The individual with the best fitness.

Parameters **fitness** (*function*) – Fitness function to calculate fitness value

Returns the best individual in current population

Return type gaft.components.IndividualBase

init(*indvs=None*)

Initialize current population with individuals.

Parameters **indvs** (*list of Individual object*) – Initial individuals in population, randomly initialized individuals are created if not provided.

max(*fitness*)

Get the maximum fitness value in population.

Parameters **fitness** (*function*) – Fitness function to calculate fitness value

Returns The maximum fitness value

Return type float

mean (*fitness*)
Get the average fitness value in population.

Parameters **fitness** (*function*) – Fitness function to calculate fitness value

Returns The average fitness value

Return type float

min (*fitness*)
Get the minimum value of fitness in population.

Parameters **fitness** (*function*) – Fitness function to calculate fitness value

Returns The minimum fitness value

Return type float

new ()
Create a new emtpy population.

update_flag ()
Interface for updating individual update flag to True.

updated
Query function for population updating flag.

worst_indv (*fitness*)
The individual with the worst fitness.

Parameters **fitness** (*function*) – Fitness function to calculate fitness value

Returns the worst individual in current population

Return type `gaft.components.IndividualBase`

4.7 gaft.operators.selection

Roulette Wheel Selection implementation.

class gaft.operators.selection.roulette_wheel_selection.**RouletteWheelSelection**
Bases: *gaft.plugin_interfaces.operators.selection.Selection*

Selection operator with fitness proportionate selection(FPS) or so-called roulette-wheel selection implementation.

select (*population, fitness*)
Select a pair of parent using FPS algorithm.

Parameters **population** (`gaft.components.Population`) – Population where the selection operation occurs.

Returns Selected parents (a father and a mother)

Return type list of `gaft.components.IndividualBase`

gaft.operators.selection.roulette_wheel_selection.**random**() → x in the interval [0, 1).

Tournament Selection implementation.

class gaft.operators.selection.tournament_selection.**TournamentSelection**(*tournament_size=2*)
Bases: *gaft.plugin_interfaces.operators.selection.Selection*

Selection operator using Tournament Strategy with tournament size equals to two by default.

Parameters **tournament_size** (*int*) – Individual number in one tournament

select(*population, fitness*)
Select a pair of parent using Tournament strategy.

Parameters **population** (gaft.components.Population) – Population where the selection operation occurs.

Returns Selected parents (a father and a mother)

Return type list of gaft.components.IndividualBase

Linear Ranking Selection implementation.

class gaft.operators.selection.linear_ranking_selection.**LinearRankingSelection**(*pmin=0.1, pmax=0.9*)
Bases: *gaft.plugin_interfaces.operators.selection.Selection*

Selection operator using Linear Ranking selection method.

Reference: Baker J E. Adaptive selection methods for genetic algorithms[C]//Proceedings of an International Conference on Genetic Algorithms and their applications. 1985: 101-111.

select(*population, fitness*)
Select a pair of parent individuals using linear ranking method.

Parameters **population** (gaft.components.Population) – Population where the selection operation occurs.

Returns Selected parents (a father and a mother)

Return type list of gaft.components.IndividualBase

gaft.operators.selection.linear_ranking_selection.**random**() → x in the interval [0, 1).

Exponential Ranking Selection implemention.

class gaft.operators.selection.exponential_ranking_selection.**ExponentialRankingSelection**(*ba*
Bases: *gaft.plugin_interfaces.operators.selection.Selection*

Selection operator using Exponential Ranking selection method.

Parameters **base** (*float in range (0.0, 1.0)*) – The base of exponent

select(*population, fitness*)
Select a pair of parent individuals using exponential ranking method.

Parameters **population** (gaft.components.Population) – Population where the selection operation occurs.

Returns Selected parents (a father and a mother)

Return type list of gaft.components.IndividualBase

gaft.operators.selection.exponential_ranking_selection.**random**() → x in the interval [0, 1).

4.8 gaft.operators.crossover

Uniform Crossover operator implementation.

class gaft.operators.crossover.uniform_crossover.**UniformCrossover**(*pc*, *pe=0.5*)
Bases: *gaft.plugin_interfaces.operators.crossover.Crossover*

Crossover operator with uniform crossover algorithm, see https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

Parameters

- **pc** (*float in (0.0, 1.0]*) – The probability of crossover (usaully between 0.25 ~ 1.0)
- **pe** (*float in range (0.0, 1.0]*) – Gene exchange probability.

cross(*father*, *mother*)
Cross chromsomes of parent using uniform crossover method.

Parameters **population** (gaft.components.Population) – Population where the selection operation occurs.

Returns Selected parents (a father and a mother)

Return type list of gaft.components.IndividualBase

gaft.operators.crossover.uniform_crossover.**random**() → x in the interval [0, 1).

4.9 gaft.operators.mutation

Flip Bit mutation implementation.

class gaft.operators.mutation.flip_bit_mutation.**FlipBitBigMutation**(*pm*, *pbm*, *alpha*)
Bases: *gaft.operators.mutation.flip_bit_mutation.FlipBitMutation*

Mutation operator using Flip Bit mutation implementation with adaptive big mutation rate to overcome premature or local-best solution.

Parameters

- **pm** (*float in (0.0, 1.0]*) – The probability of mutation (usually between 0.001 ~ 0.1)
- **pbm** (*float*) – The probability of big mutation, usually more than 5 times bigger than pm.
- **alpha** (*float, in range (0.5, 1)*) – intensive factor

mutate(*individual*, *engine*)
Mutate the individual with adaptive big mutation rate.

Parameters

- **individual** (gaft.components.IndividualBase) – The individual on which crossover operation occurs
- **engine** (*gaft.engine.GAEngine*) – Current genetic algorithm engine

Returns A mutated individual

Return type gaft.components.IndividualBase

class gaft.operators.mutation.flip_bit_mutation.**FlipBitMutation**(*pm*)
Bases: *gaft.plugin_interfaces.operators.mutation.Mutation*

Mutation operator with Flip Bit mutation implementation.

Parameters **pm** (*float in range (0.0, 1.0]*) – The probability of mutation (usually between 0.001 ~ 0.1)

mutate(*individual*, *engine*)
Mutate the individual.

Parameters

- **individual** (gaft.components.IndividualBase) – The individual on which crossover operation occurs
- **engine** (*gaft.engine.GAEngine*) – Current genetic algorithm engine

Returns A mutated individual

Return type gaft.components.IndividualBase

gaft.operators.mutation.flip_bit_mutation.**random**() → x in the interval [0, 1).

4.10 gaft.analysis.console_output

class gaft.analysis.console_output.**ConsoleOutput**
Bases: *gaft.plugin_interfaces.analysis.OnTheFlyAnalysis*

Built-in on-the-fly analysis plugin class for outputing log on console.

Attribute:

interval(int): The analysis interval in evolution iteration, default value is 1 meaning analyze every step.

master_only(bool): Flag for if the analysis plugin is only effective in master process. Default is True.

finalize(*population*, *engine*)
Called after the iteration to allow for custom finalization and post-processing of the collected data.

Parameters

- **population** (Population) – The up to date population of the iteration.
- **engine** (gaft.engine.GAEngine) – The current GAEngine where the analysis is running.

register_step(*g*, *population*, *engine*)
Function called in each iteration step.

Parameters

- **g** (*int*) – Current generation number.
- **population** (Population) – The up to date population of the iteration.
- **engine** (gaft.engine.GAEngine) – The current GAEngine where the analysis is running.

setup(*ng*, *engine*)
Function called right before the start of genetic algorithm main iteration to allow for custom setup of the analysis object.

Parameters

- **ng** (*int*) – The number of generation.
- **engine** (gaft.engine.GAEngine) – The current GAEngine where the analysis is running.

4.11 gaft.analysis.fitness_store

class gaft.analysis.fitness_store.**FitnessStore**

Bases: *gaft.plugin_interfaces.analysis.OnTheFlyAnalysis*

Built-in on-the-fly analysis plugin class for storing fitness related data during iteration.

Attribute:

interval(int): The analysis interval in evolution iteration, default value is 1 meaning analyze every step.

master_only(bool): Flag for if the analysis plugin is only effective in master process. Default is True.

finalize (*population*, *engine*)

Called after the iteration to allow for custom finalization and post-processing of the collected data.

Parameters

- **population** (Population) – The up to date population of the iteration.
- **engine** (gaft.engine.GAEngine) – The current GAEngine where the analysis is running.

register_step (*g*, *population*, *engine*)

Function called in each iteration step.

Parameters

- **g** (*int*) – Current generation number.
- **population** (Population) – The up to date population of the iteration.
- **engine** (gaft.engine.GAEngine) – The current GAEngine where the analysis is running.

setup (*ng*, *engine*)

Function called right before the start of genetic algorithm main iteration to allow for custom setup of the analysis object.

Parameters

- **ng** (*int*) – The number of generation.
- **engine** (gaft.engine.GAEngine) – The current GAEngine where the analysis is running.

4.12 gaft.plugin_interfaces.analysis

class gaft.plugin_interfaces.analysis.**OnTheFlyAnalysis**

Bases: object

Class for providing an interface to easily extend and customize the behavior of the on-the-fly analysis functionality of gaft.

Attribute:

interval(`int`): The analysis interval in evolution iteration, default value is 1 meaning analyze every step.

master_only(`bool`): Flag for if the analysis plugin is only effective in master process. Default is True.

`finalize`(*population*, *engine*)
Called after the iteration to allow for custom finalization and post-processing of the collected data.

Parameters

- **`population`** (`Population`) – The up to date population of the iteration.
- **`engine`** (`gaft.engine.GAEngine`) – The current GAEngine where the analysis is running.

`register_step`(*g*, *population*, *engine*)
Function called in each iteration step.

Parameters

- **`g`** (*`int`*) – Current generation number.
- **`population`** (`Population`) – The up to date population of the iteration.
- **`engine`** (`gaft.engine.GAEngine`) – The current GAEngine where the analysis is running.

`setup`(*ng*, *engine*)
Function called right before the start of genetic algorithm main iteration to allow for custom setup of the analysis object.

Parameters

- **`ng`** (*`int`*) – The number of generation.
- **`engine`** (`gaft.engine.GAEngine`) – The current GAEngine where the analysis is running.

4.13 gaft.plugin_interfaces.operators

Module for Genetic Algorithm selection operator class

class `gaft.plugin_interfaces.operators.selection.`**`Selection`**
Bases: `object`

Class for providing an interface to easily extend the behavior of selection operation.

`select`(*population*, *fitness*)
Called when we need to select parents from a population to later breeding.

Parameters **`population`** (*`gaft.compoenents.Population`*) – The current population

Return parents Two selected individuals for crossover

Module for Genetic Algorithm crossover operator class

class gaft.plugin_interfaces.operators.crossover.**Crossover**

Bases: object

Class for providing an interface to easily extend the behavior of crossover operation between two individuals for children breeding.

Attributes:

pc(float): The probability of crossover (usaully between 0.25 ~ 1.0)

cross (*father*, *mother*)

Called when we need to cross parents to generate children.

Parameters

- **father** (*gaft.components.IndividualBase*) – The parent individual to be crossed
- **mother** (*gaft.components.IndividualBase*) – The parent individual to be crossed

Return children Two new children individuals

Module for Genetic Algorithm mutation operator class

class gaft.plugin_interfaces.operators.mutation.**Mutation**

Bases: object

Class for providing an interface to easily extend the behavior of selection operation.

Attributes:

pm(float): Default mutation probability, default is 0.1

mutate (*individual*, *engine*)

Called when an individual to be mutated.

Parameters

- **individual** (*gaft.components.IndividualBase*) – The individual to be mutated
- **engine** (gaft.engine.GAEngine) – The GA engine where the mutation operator belongs.

Python Module Index

g

Index

I

L

M

N

O

P

R

S

T

U

W