Python Documentation

version

abril 20, 2020

Contents

Genetic algorithm with chromosomes with variable length

Main class - Gavl

Gavl

In this file it is defined the main class to execute the GA with chromosomes with variable length.

Classes:

Gavl: Main class.

Example call:

ga = Gavl() ga.set_hyperparameter('size_population', 100) ga.set_hyperparameter('min_length_chromosome', 3) ga.set_hyperparameter('max_length_chromosome', 10) def fitness(chromosome):

return sum(chromosome)

ga.set_hyperparameter('fitness', fitness) ga.set_hyperparameter('possible_genes', list(range(20))) ga.set_hyperparameter('termination_criteria', {'max_num_generation_reached': 150}) ga.set_hyperparameter('repeated_genes_allowed', 0) ga.set_hyperparameter('minimize', 1) ga.set_hyperparameter('elitism_rate', 0.3) ga.set_hyperparameter('mutation_rate', 0.1) ga.set_hyperparameter('mutation_type', 'both') ga.set_hyperparameter('keep_diversity',5) ga.set_hyperparameter('show_progress',1) # Launch optimization: best_individual = ga.optimize() # Get the results:

class `Gavl.Gavl.`**`Gavl`**

This class is the main class for executing the GA and the only one that should be called.

`add_individual``(individual)`

Method to add a new individual to the population. This method is overriding the method with the same name of the class Population —> This is done to check that it is not added more individuals than self.size_population.

Parameters: **individual** – (Individual or list) individual of the class Individual or list representing the chromosome.

Returns:

`best_individual``()`

This method returns the best individual.

Returns: individual(Individual) Best individual.

`get_results``()`

This method is used to get the results of the optimization process, once it has finished.

Returns: best_individual(Individual) Best individual. population(list of Individuals) List with all the individuals of the last generation. historic_fitness: (list of floats) List with the best fitness value in each generation.

`historic_fitness``()`

This method returns the best fitness value in each generation.

Returns: bfv(list of floats) This method returns a list with the best fitness value in each generation.

`optimize``()`

This method is the one that is called to start the optimization (start the genetic algorithm). Before calling this method it must be defined Gavl.size_population, Gavl.min_length_chromosome, Gavl.max_length_chromosome, Gavl.fitness and Gavl.possible_genes.

Returns: (Individual) Best individual.

set_hyperparameter (id_hyperparameter, value)
Method to set the hyperparameters (attributes of the class Gavl).

Parameters:
- **id_hyperparameter** – (str) Id (name) of the hyperparameter that is going to be set.
- **value** – (…) Value of the hyperparameter that is going to be set.

Returns:

Auxiliary classes

Population

In this file it is defined the class to hold the population.

Classes:

Population: Main class.

class Gavl.tools.population.**Population**
Class of the population.

add_individual (individual)
Method to add a new individual to the population. BEWARE: The population may have a bigger size than the specified if new individuals are added.

Parameters: **individual** – (Individual or list) individual of the class Individual or list representing the chromosome.

Returns:

abstract **best_individual** ()
This method returns the best individual.

Returns: individual(Individual) Best individual.

get_individual_by_id (id_individual)
This method returns an Individual given its ID. If there is no Individual with this ID in the population, it returns None.

Parameters: **id_individual** – (str) ID of the individual.

Returns: individual(Individual) Individual whose id is id_individual.

get_population ()
Method to get the population

Returns:

Individual

In this file it is defined the class Individual.

Classes:

Individual: Main class.

class Gavl.tools.individual.**Individual** (chromosome)
Class of the individuals. It has the attributes chromosome, fitness and normalized fitness.

calculate_fitness (fitness)

Method to calculate the fitness of the individual.

Parameters: **fitness** – (function) Function to evaluate the fitness. Its ONLY argument is the individual's chromosome (fitness(chromosome)) and returns the value of the fitness.

Returns: fitness_value(float) The fitness value It is also set the value of the attribute Individual.fitness_value to the calculated value when this method is called.

kill_and_reset (chromosome**)**

This is a bit of a "tricky" method. If a new individual of a new generation is going to be created, it will be computationally much faster to get the object of an individual that is going to be killed and reset its values to the ones of the new individual. —> This will be done instead of directly creating a completely new individual with the new data and stop referencing the old individual.

Parameters: **chromosome** – (list of genes) Chromosome of the individual.

set_fitness_value (fitness_value**)**

Method to set the fitness value.

Parameters: **fitness_value** – (float) Fitness value.

Returns:

set_inverse_normalized_fitness_value (inverse_normalized_fitness_value**)**

Method to set the inverse normalized fitness value.

Parameters: **inverse_normalized_fitness_value** – (float) Fitness value.

Returns:

set_new_chromosome (chromosome**)**

Method to set a chromosome.

Parameters: **chromosome** – (list of genes) Chromosome of the individual.

Returns:

set_normalized_fitness_value (normalized_fitness_value**)**

Method to set the normalized fitness value.

Parameters: **normalized_fitness_value** – (float) Fitness value.

Returns:

Tools

Crossover

In this file it is defined the functions to perform crossover.

Functions:

cross_individuals: this function performs the crossover of two given individuals. mating: this function calculates the crossover of ALL the paired individuals with the help of the function cross_ind.

Gavl.tools.crossover.**cross_individuals** (chromosome_a, chromosome_b, min_length_chromosome, max_length_chromosome, repeated_genes_allowed, check_valid_individual)

This function calculates the crossover between two different individuals. It selects a random number of genes of the individuals a and b that are going to be interchanged (note that the number of genes from individual a to individual b may be different than the number of genes changed from individual b to individual a), and calculates if there is any possible crossover of that size (with the function check_valid_individual). If so, one of the possible crossovers (selected randomly) is performed and the resulting new chromosomes are returned. If there is no possible crossover of that size, it is tested another different combination of sizes. Notice that the function check_valid_individual is used to test the created individuals, and if it is done 2000 unsuccessful crossovers, it is taken as an impossible to couple pair of individuals and their original chromosomes are returned.

Parameters:

- **chromosome_a** – (Individual) Individual A's chromosome.
- **chromosome_b** – (Individual) Individual B's chromosome.
- **min_length_chromosome** – (int) Minimum number of genes of the chromosome.
- **max_length_chromosome** – (int) Maximum number of genes of the chromosome.
- **repeated_genes_allowed** – (int) Boolean that indicates if an individual can have repeated genes. It can take the values 1 (repeated genes allowed) or 0 (repeated genes not allowed).
- **check_valid_individual** – (function) Function that receives a chromosome and returns a boolean that indicates whether the chromosome makes a valid individual (True) or not (False) according to some criteria.

Returns: crossed_aCrossed individual a. crossed_bCrossed individual b.

`Gavl.tools.crossover.`**`mating`** (`list_of_paired_ind`, `min_length_chromosome`, `max_length_chromosome`, `repeated_genes_allowed`, `check_valid_individual`)

This function returns the mated couples of individuals when it is possible to make this mating. If it is not possible to make the mating because all the combinations give invalid individuals (if repeated_genes_allowed = 0 and the two individuals are exactly the same genes), then the two intended to be paired individuals are returned.

Parameters:

- **list_of_paired_ind** – (list of tuples of Individuals) List of tuples where each tuple represents two paired individuals. It has the form [(Individual_a, Individual_b), (Individual_c, Individual_d), …]; having paired in this example the individual a with the individual b, and the individual c with the individual d. Note that the tuples contain objects of the class Individual.
- **min_length_chromosome** – (int) Minimum number of genes of the chromosome.
- **max_length_chromosome** – (int) Maximum number of genes of the chromosome.
- **repeated_genes_allowed** – (int) Boolean that indicates if an individual can have repeated genes. It can take the values 1 (repeated genes allowed) or 0 (repeated genes not allowed).
- **check_valid_individual** – (function) Function that receives a chromosome and returns a boolean that indicates whether the chromosome makes a valid individual (True) or not (False) according to some criteria.

Returns: crossed_individuals(list of chromosomes) List with the crossed individuals' chromosomes.

Generate chromosome

In this file it is defined the functions to create a new individual (chromosome).

Functions:

generate_chromosome: Main function.

`Gavl.tools.generate_chromosome.`**`generate_chromosome`** (`min_length_chromosome`, `max_length_chromosome`, `possible_genes`, `repeated_genes_allowed`)

Function called to create a new individual (its chromosome). It randomly chooses its length (between min_length_chromosome and min_length_chromosome), and it randomly chooses genes among the list of possible_genes.

Parameters:

- **min_length_chromosome** – (int) Minimum allowed length of the chromosome.
- **max_length_chromosome** – (int) Maximum allowed length of the chromosome.
- **possible_genes** – (list of …) List with the all the possible values that the genes can take.
- **repeated_genes_allowed** – (bool) It is a boolean that indicates whether the genes can be repeated in the chromosome (repeated_genes_allowed = 1) or they cannot be repeated (repeated_genes_allowed = 0).

Returns: (list of genes) List that represents the chromosome.

Keep diversity

In this file it is defined a function to keep the diversity.

Functions:

keep_diversity: Function called to keep the diversity.

`Gavl.tools.keep_diversity.`**`keep_diversity`** (`population`, `generate_chromosome`, `min_length_chromosome`, `max_length_chromosome`, `possible_genes`, `repeated_genes_allowed`, `check_valid_chromosome`)

This function is called when it is wanted to do a great emphasis in the diversity of the population. When an individual is repeated in the population it is substituted by a completely new and randomly generated individual. As well the worst 25% of the population is substituted by completely new and random individuals. Note that before calling this function the population MUST be sorted by fitness.

Parameters:

- **population** – (list of Individuals) Sorted population by fitness (from best to worst).
- **generate_chromosome** – (function) Function to generate a new chromosome.
- **min_length_chromosome** – (int) Minimum allowed length of the chromosome.
- **max_length_chromosome** – (int) Maximum allowed length of the chromosome.
- **possible_genes** – (list of …) List with the all the possible values that the genes can take.
- **repeated_genes_allowed** – (bool) It is a boolean that indicates whether the genes can be repeated in the chromosome (repeated_genes_allowed = 1) or they cannot be repeated (repeated_genes_allowed = 0).
- **check_valid_chromosome** – (function) Function that receives the chromosome and returns True if it creates a valid individual and False otherwise.

Returns: (list of chromosomes) List of chromosomes that will represent the next generation.

Mutation

In this file it is defined the function to perform mutation.

Function:

mutation: Function that performs mutation. mutate_genes_manner: Auxiliary function to make the mutation by changing the genes. mutate_length_manner: Auxiliary function to make the mutation in length.

`Gavl.tools.mutation.`**`mutate_genes_manner`** (`chromosome`, `max_num_gen_changed_mutation`, `mutation_genes`, `check_valid_individual`)

This function performs the mutation of the genes (not the length).

Parameters:

- **chromosome** – (list of genes) Chromosome to mutate.
- **max_num_gen_changed_mutation** – (int) Maximum number of genes changed in the mutation. It is the attribute .max_num_gen_changed_mutation of the class Gavl().
- **mutation_genes** – (list of genes) List with the possible genes that can be mutated (genes that are not in the individual).
- **check_valid_individual** – (function) Function that receives a chromosome and returns a boolean that indicates whether the chromosome makes a valid individual (True) or not (False) according to some criteria.

Returns: new_chromosome(list of genes) New mutated chromosome.

`Gavl.tools.mutation.`**`mutate_length_manner`** (`chromosome`, `max_num_gen_changed_mutation`, `min_length_chromosome`, `max_length_chromosome`, `mutation_genes`, `check_valid_individual`)

This function performs the mutation of the genes (not the length).

Parameters:

- **chromosome** – (list of genes) Chromosome to mutate.
- **max_num_gen_changed_mutation** – (int) Maximum number of genes changed in the mutation. It is the attribute .max_num_gen_changed_mutation of the class Gavl().
- **min_length_chromosome** – (int) Minimum allowed number of genes of a chromosome. It is the attribute .min_length_chromosome of the class Gavl().
- **max_length_chromosome** – (int) Maximum allowed number of genes of a chromosome. It is the attribute .max_length_chromosome of the class Gavl().
- **mutation_genes** – (list of genes) List with the possible genes that can be mutated (genes that are not in the individual).
- **check_valid_individual** – (function) Function that receives a chromosome and returns a boolean that indicates whether the chromosome makes a valid individual (True) or not (False) according to some criteria.

Returns: new_chromosome(list of genes) New mutated chromosome.

`Gavl.tools.mutation.`**`mutation`** (`chromosomes_to_mutate`, `mutation_type`, `max_num_gen_changed_mutation`, `min_length_chromosome`, `max_length_chromosome`, `repeated_genes_allowed`, `check_valid_individual`, `possible_genes`)

This function receives a chromosome and performs a random mutation over one of its elements. It iterates over all the possible mutations until one is found, moment in which the execution stops. If no mutation is found, then it returns the input chromosome. Notice that the function check_valid_individual is used to test the created individuals, and if it is done 1000 unsuccessful mutations on the same individual, it is taken as an impossible to mutate individual and its original chromosome is returned.

Parameters:

- **chromosomes_to_mutate** – (list of chromosomes) List of the chromosomes that are going to be mutated.
- **mutation_type** – (str) String that represents the mutation type. It can ONLY take the values ‘mut_gene’, ‘addsub_gene’ or ‘both’. It is the attribute .mutation_type of the class Gavl().
- **max_num_gen_changed_mutation** – (int) Maximum number of genes changed in the mutation. It is the attribute .max_num_gen_changed_mutation of the class Gavl().
- **min_length_chromosome** – (int) Minimum allowed number of genes of a chromosome. It is the attribute .min_length_chromosome of the class Gavl().
- **max_length_chromosome** – (int) Maximum allowed number of genes of a chromosome. It is the attribute .max_length_chromosome of the class Gavl().
- **repeated_genes_allowed** – (int) Boolean that indicates if an individual can have repeated genes. It can take the values 1 (repeated genes allowed) or 0 (repeated genes not allowed).
- **check_valid_individual** – (function) Function that receives a chromosome and returns a boolean that indicates whether the chromosome makes a valid individual (True) or not (False) according to some criteria.
- **possible_genes** – (list of genes) List with the possible genes.

Returns: list_new_mutated_chromosomes(list of chromosomes) List with the chromosomes of the individuals that are mutated.

Pairing

In this file it is defined the function to perform random pairing given a selection (like the one given by roulette wheel).

Functions:

pairing: Given a selection (list with IDs), it performs a pairing of individuals.

`Gavl.tools.pairing.`**`pairing`** (`list_selected_ind`)

This function performs random pairing. NOTE that this function accepts repetitions and elements may be paired with themselves. However, this rarely happens and it can be used as a elite process.

Parameters: **list_selected_ind** – List with the IDs of the selected individuals. It is the output of the function roulette_selection.

Returns: paired_indList of tuples with the IDs of the paired individuals.

Selection

In this file it is defined the function to perform a roulette wheel selection.

Functions:

roulette_selection: Given a list with the tuples (id_individual, normalized_fitness), this function calculates the a roulette wheel selection based in the normalized fitness.

`Gavl.tools.selection.`**`roulette_selection`** (`population`, `minimize`, `num_selected_ind`)

This function returns a list with the ids of the individuals selected by the roulette wheel selection. Note that the normalized fitness of the population must be calculated before calling this function (call the method Gavl._Population__calculate_normalized_fitness).

Parameters:

- **population** – (list of Individuals) This is a list of individuals (see class Individual).
- **minimize** – (int) Int that represents if the goal is minimizing the fitness (minimize = 1) or maximizing it (minimize = 0).
- **num_selected_ind** – (int) number of individuals to be selected.

Returns: list_selected_individuals(list of str) List with the ids of the selected individuals. Note that there can be repeated individuals.

Termination criteria

In this file it is defined the functions to check the termination criteria.

Functions:

max_num_generation_reached: Function that checks if the max number of generations is reached. goal_fitness_reached: Function that checks if the goal fitness is reached. check_termination_criteria: This function selects the termination criteria

`Gavl.tools.termination_criteria.`**`check_termination_criteria`** (`termination_criteria_args`)

This function checks the termination criteria.

Parameters: **termination_criteria_args** – (dictionary) This is a dictionary with the needed arguments to check the termination criteria. It can take the values: * {'termination_criteria': 'goal_fitness_reached', 'goal_fitness': _ , 'generation_fitness': _ , 'minimize': _ } * {'termination_criteria': 'max_num_generation_reached', 'generation_goal': _ , 'generation_count': _ }

Returns: (bool) True if the termination criteria is met. False otherwise.

`Gavl.tools.termination_criteria.`**`goal_fitness_reached`** (`generation_best_fitness`, `goal_fitness`, `minimize`)

This function returns True if the goal fitness is reached.

Parameters:

- **generation_best_fitness** – (int) Current generation best fitness.
- **goal_fitness** – (int) Goal fitness.
- **minimize** – (bool) If 1, the goal is minimize the fitness function and viceversa.

Returns: (bool) True if the goal fitness is reached.

`Gavl.tools.termination_criteria.`**`max_num_generation_reached`** (`generation_count`, `max_generations`)

This function returns True if the maximum number of generations is reached.

Parameters:

- **generation_count** – (int) Number (count) of the current generation.
- **max_generations** – (int) Maximum number of generations.

Returns: (bool) True if the maximum number of generations is reached.

Auxiliary functions

In this file it is defined the auxiliary function to get all the combinations of some length of the elements of a list. It is exactly like the tool itertools.combinations, but, as said below, itertools goes prrr (just wanted to define this function in a recurrent and dynamic programming manner).

Function:

combinations: function that returns the combinations of some length of the elements in a list.

`Gavl.tools.aux_functions.combinations.`**`combinations`**`(list_get_comb, length_combination)`

Generator to get all the combinations of some length of the elements of a list. —> itertools.combinations goes prrrr.

Parameters:

- **list_get_comb** – (list) List from which it is wanted to get the combination of its elements.
- **length_combination** – (int) Length of the combinations of the elements of list_get_comb.

Returns: generatorGenerator with the combinations of this list.

Index

Python Module Index

g