Name:**David Bukedi Diela Date:2020-10-11**

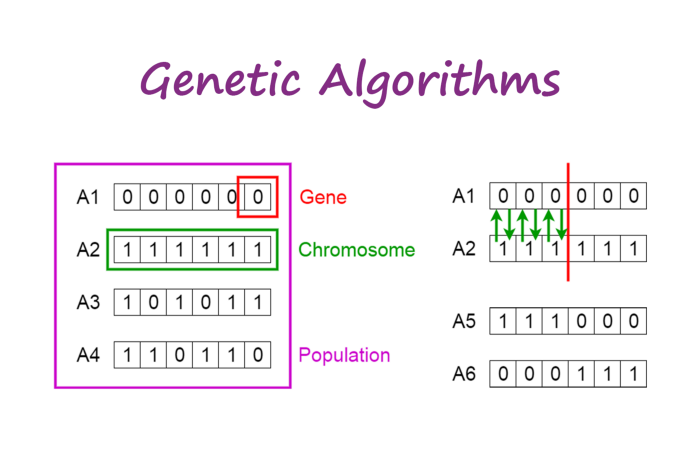
Student ID:**2018529627050**

**Genetic Algorithm-Guess Password**

1. Introduction

A genetic algorithm is a search heuristic that is inspired by **Charles Darwin**’s theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Genetic algorithms are one of the tools you can use to apply machine learning to finding good, sometimes even optimal, solutions to problems that have billions of potential solutions.

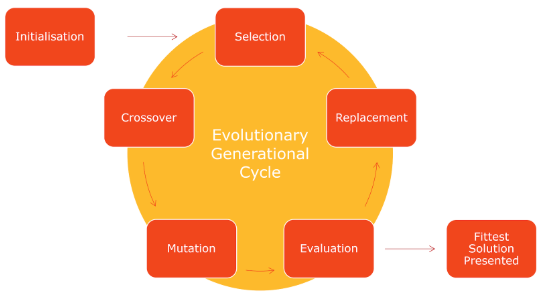


1. Outline of the algorithm

The following outline summarizes how the genetic algorithm works:

1. The algorithm begins by creating a random initial population.
2. The algorithm then creates a sequence of new populations. At each step, the algorithm uses the individuals in the current generation to create the next population. To create the new population, the algorithm performs the following steps:
3. Scores each member of the current population by computing its fitness value. These values are called the raw fitness scores.
4. Scales the raw fitness scores to convert them into a more usable range of values. These scaled values are called expectation values.
5. Selects members, called parents, based on their expectation.
6. Some of the individuals in the current population that have lower fitness are chosen as elite. These elite individuals are passed to the next population.
7. Produces children from the parents. Children are produced either by making random changes to a single parent—mutation—or by combining the vector entries of a pair of parents—crossover.
8. Replaces the current population with the children to form the next generation.
9. The algorithm stops when one of the stopping criteria is met(see below)

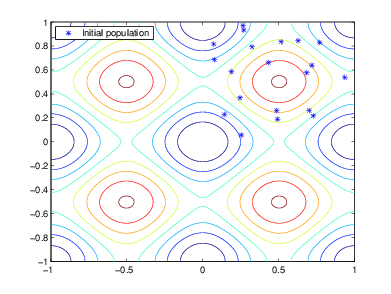
Main idea:



1. Process

### 1.Initial Population

The algorithm begins by creating a random initial population, as shown in the following figure.



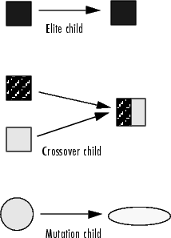
### 2.Creating the Next Generation

At each step, the genetic algorithm uses the current population to create the children that make up the next generation. The algorithm selects a group of individuals in the current population, called parents, who contribute their genes—the entries of their vectors—to their children. The algorithm usually selects individuals that have better fitness values as parents. You can specify the function that the algorithm uses to select the parents in the Selection function option.

The genetic algorithm creates three types of children for the next generation:

* Eliterae the individuals in the current generation with the best fitness values. These individuals automatically survive to the next generation.
* Crossover are created by combining the vectors of a pair of parents.
* Mutation children are created by introducing random changes, or mutations, to a single parent.

The following schematic diagram illustrates the three types of children:



#### **Crossover Children**

The algorithm creates crossover children by combining pairs of parents in the current population. At each coordinate of the child vector, the default crossover function randomly selects an entry, or gene, at the same coordinate from one of the two parents and assigns it to the child. For problems with linear constraints, the default crossover function creates the child as a random weighted average of the parents.

#### **Mutation Children**

The algorithm creates mutation children by randomly changing the genes of individual parents. By default, for unconstrained problems the algorithm adds a random vector from a Gaussian distribution to the parent. For bounded or linearly constrained problems, the child remains feasible.

3.Stopping Conditions for the Algorithm

The genetic algorithm uses the following options to determine when to stop.

* The algorithm stops when the number of generations reaches Max Generations.
* The algorithm stops after running for an amount of time in seconds equal to Max Time.
* The algorithm stops when the value of the fitness function for the best point in the current population is less than or equal to Fitness Limit.
* The algorithm stops when the average relative change in the fitness function value over Max Stall Generations is less than Function tolerance.
* The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to Max Stall Time.
* The algorithm runs until the average relative change in the fitness function value over Max Stall Generations is less than Function tolerance.

The algorithm stops as soon as any one of these conditions is met.

### 4.Selection

The selection function chooses parents for the next generation based on their scaled values from the fitness scaling function. The scaled fitness values are called the expectation values. An individual can be selected more than once as a parent, in which case it contributes its genes to more than one child. The default selection option, @selectionstochunif, lays out a line in which each parent corresponds to a section of the line of length proportional to its scaled value. The algorithm moves along the line in steps of equal size. At each step, the algorithm allocates a parent from the section it lands on.

### **5.**Reproduction Options

Reproduction options control how the genetic algorithm creates the next generation. The options are:

-The number of individuals with the best fitness values in the current generation that are guaranteed to survive to the next generation. These individuals are called elite children.

- The fraction of individuals in the next generation, other than elite children, that are created by crossover.

Because elite individuals have already been evaluated, ga does not reevaluate the fitness function of elite individuals during reproduction. This behavior assumes that the fitness function of an individual is not random, but is a deterministic function. To change this behavior, use an output function.

### 6.Mutation and Crossover

The genetic algorithm uses the individuals in the current generation to create the children that make up the next generation. Besides elite children, which correspond to the individuals in the current generation with the best fitness values, the algorithm creates

Crossover children by selecting vector entries, or genes, from a pair of individuals in the current generation and combines them to form a child

Mutation children by applying random changes to a single individual in the current generation to create a child

Both processes are essential to the genetic algorithm. Crossover enables the algorithm to extract the best genes from different individuals and recombine them into potentially superior children. Mutation adds to the diversity of a population and thereby increases the likelihood that the algorithm will generate individuals with better fitness values.

4. Pseudo code

START

Generate the initial population

Compute fitness

REPEAT

Selection

Crossover

Mutation

Compute fitness

UNTIL population has converged

STOP

1. Use and example

Here are some uses of the genetic algorithm in different sectors

## Natural Sciences, Mathematics and Computer Science[[edit](https://en.wikipedia.org/w/index.php?title=List_of_genetic_algorithm_applications&action=edit&section=1" \o "Edit section: Natural Sciences, Mathematics and Computer Science)]

* Bayesian inference links to particle methods in Bayesian statistics and hidden Markov chain models[[1]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-1)[[2]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-hal.inria.fr-2)
* [Artificial creativity](https://en.wikipedia.org/wiki/Computational_creativity" \o "Computational creativity)
* Chemical kinetics ([gas](https://archive.is/20121223015305/http://www.personal.leeds.ac.uk/~fuensm/project.html) and [solid](http://repositories.cdlib.org/postprints/1154) phases)
* Calculation of [bound states](https://en.wikipedia.org/wiki/Bound_state" \o "Bound state) and [local-density approximations](https://en.wikipedia.org/wiki/Local-density_approximation" \o "Local-density approximation)
* [Code-breaking](https://en.wikipedia.org/wiki/Code-breaking" \o "Code-breaking), using the GA to search large solution spaces of [ciphers](https://en.wikipedia.org/wiki/Cipher" \o "Cipher) for the one correct decryption.[[3]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-3)
* Computer architecture: using GA to find out weak links in [approximate computing](https://en.wikipedia.org/wiki/Approximate_computing" \o "Approximate computing) such as [lookahead](https://en.wikipedia.org/wiki/Combinatorial_search" \l "Lookahead" \o "Combinatorial search).

## Earth Sciences[[edit](https://en.wikipedia.org/w/index.php?title=List_of_genetic_algorithm_applications&action=edit&section=2" \o "Edit section: Earth Sciences)]

* [Climatology](https://en.wikipedia.org/wiki/Climatology" \o "Climatology): Estimation of [heat flux](https://en.wikipedia.org/wiki/Heat_flux" \o "Heat flux) between the atmosphere and sea ice[[18]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-18)
* [Climatology](https://en.wikipedia.org/wiki/Climatology" \o "Climatology): Modelling [global temperature](https://en.wikipedia.org/wiki/Temperature_record" \o "Temperature record) changes[[19]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-19)
* Design of [water resource](https://en.wikipedia.org/wiki/Water_resource" \o "Water resource) systems [[20]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-Zhang_&_Babovic_2012-20)
* Groundwater monitoring networks[[21]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-21)

## Finance and Economics[[edit](https://en.wikipedia.org/w/index.php?title=List_of_genetic_algorithm_applications&action=edit&section=3" \o "Edit section: Finance and Economics)]

* [Financial mathematics](https://en.wikipedia.org/wiki/Financial_mathematics" \o "Financial mathematics)[[2]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-hal.inria.fr-2)[[22]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-22)
  + Automated design of sophisticated trading systems in the financial sector; see [Automated trading system](https://en.wikipedia.org/wiki/Automated_trading_system" \o "Automated trading system)
  + [Real options valuation](https://en.wikipedia.org/wiki/Real_options_valuation" \o "Real options valuation) [[23]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-Zhang_&_Babovic-23)
  + [Portfolio optimization](https://en.wikipedia.org/wiki/Portfolio_optimization" \o "Portfolio optimization)[[24]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-24)
* [Genetic algorithm in economics](https://en.wikipedia.org/wiki/Genetic_algorithm_in_economics" \o "Genetic algorithm in economics)
  + Representing rational agents in economic models such as the [cobweb model](https://en.wikipedia.org/wiki/Cobweb_model" \o "Cobweb model)

## Social Sciences[[edit](https://en.wikipedia.org/w/index.php?title=List_of_genetic_algorithm_applications&action=edit&section=4" \o "Edit section: Social Sciences)]

* Design of [anti-terrorism](https://en.wikipedia.org/wiki/Anti-terrorism" \o "Anti-terrorism) systems [[25]](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications" \l "cite_note-Buurman,_Zhang_&_Babovic-25)
* Linguistic analysis, including [grammar induction](https://en.wikipedia.org/wiki/Grammar_induction" \o "Grammar induction) and other aspects of [Natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing" \o "Natural language processing) (NLP) such as word sense disambiguation.

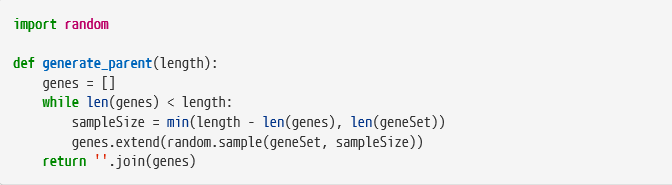
1. Guessing the password

Now let’s see how this applies to guessing a password. Start with a randomly generated initial sequence of letters, then mutate/change one random letter at a time until the sequence of letters is "Hello World!". Conceptually:

Code:

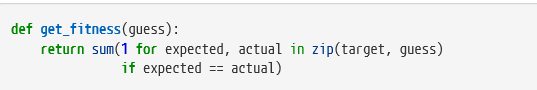
1. Generate a guess

generate a random string from the gene set.



1. Fitness

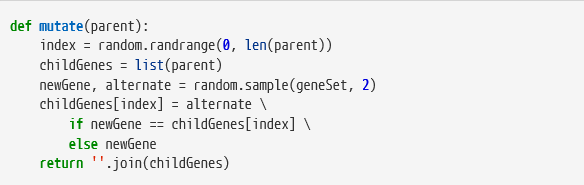
The fitness value the genetic algorithm provides is the only feedback the engine gets to guide it toward a solution. In this project the fitness value is the total number of letters in the guess that match the letter in the same position of the password.



1. Mutation

This implementation uses an alternate replacement if the randomly selected newGene is the same as the one it is supposed to replace, which can prevent a significant number of wasted guesses.

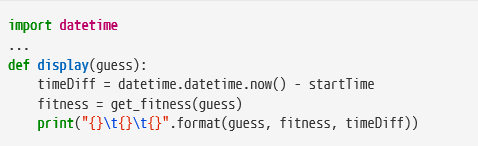
The engine needs a way to produce a new guess by mutating the current one.



1. Display

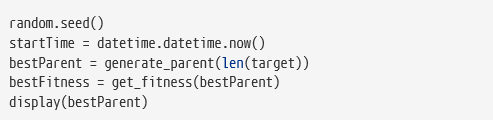
it is important to monitor what is happening so that the engine can be stopped if it gets stuck. Having a visual representation of the gene sequence, which may not be the literal gene sequence, is often critical to identifying what works and what does not so that the algorithm can be improved.

The display function also outputs the fitness value and how much time has elapsed



1. Main

The main program begins by initializing bestParent to a random sequence of letters and calling the display function.

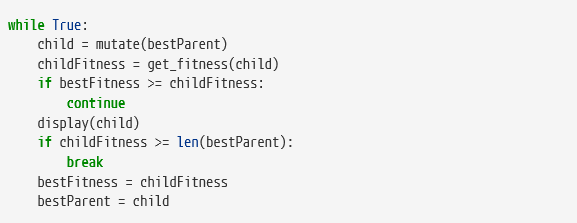


• generates a guess,

• requests the fitness for that guess, then

• compares the fitness to that of the previous best guess, and

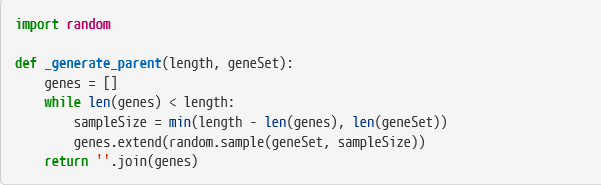
• keeps the guess with the better fitness.

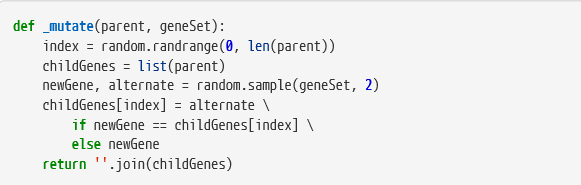


(we will then create a new file called genetic.py to create protected methods )

1. Genetic.py

Future projects will need to be able to customize the gene set, so that needs to become a parameter to \_generate\_parent and \_mutate.





1. get\_best

The next step is to move the main loop into a new public function named get\_best in the genetic module. Its parameters are:

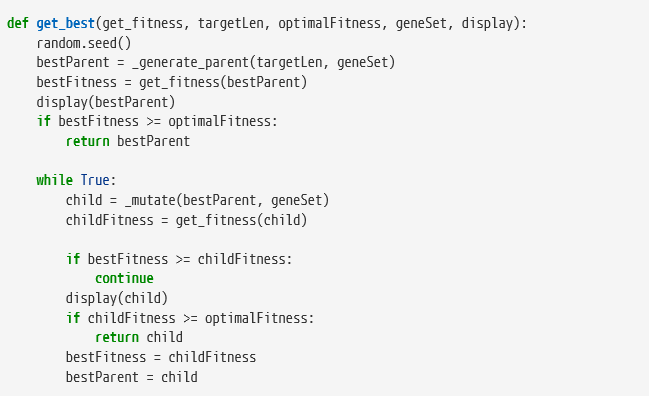
• the function it calls to request the fitness for a guess,

• the number of genes to use when creating a new gene sequence,

• the optimal fitness value,

• the set of genes to use for creating and mutating gene sequences, and

• the function it should call to display, or report, each improvement found.



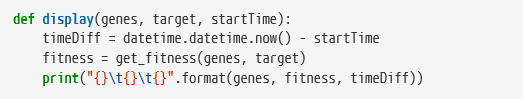
1. Helper functions

Each helper function will take the candidate gene sequence it receives and call the local functions with additional required parameters as necessary



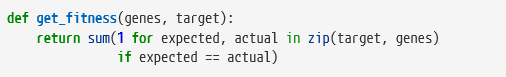
9. Display

Now change display to take the target password as a parameter.



1. Fitness

The fitness function also needs to receive the target password as a parameter

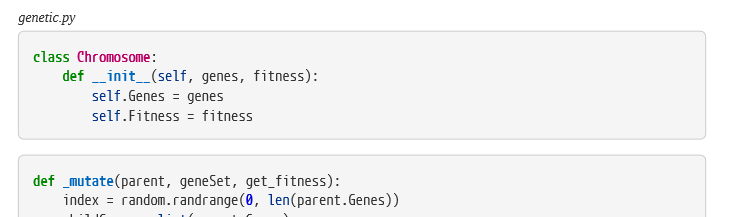


1. Use Python’s unittest framework



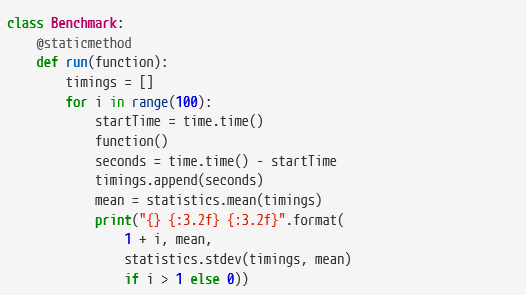
1. introduce chromosomes class

The next change is to introduce a Chromosome object that has Genes and Fitness attributes. This will make the genetic engine more flexible by making it possible to pass those values around as a unit.



1. Benchmarking

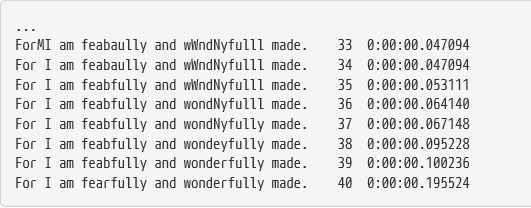
improvement is to add support for benchmarking to genetic because it is useful to know how long the engine takes to find a solution on average and the standard deviation(see the PDF for more)



13.

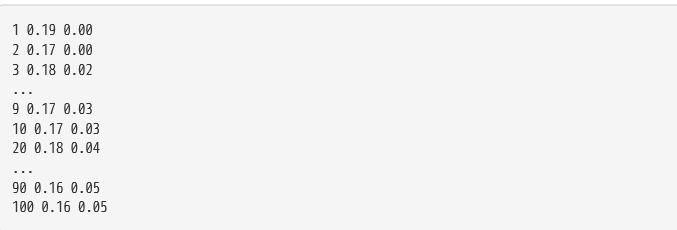
1. Results and analysis

Here is a simple demonstration of how the genetic algorithm works (the password was found at the 40th generation)



we can try any password, the algorithm will guess by starting with a randomly generated initial sequence of letters, then mutating/changing one random letter at a time until the sequence of letters is the given password

\*Benchmarking:



This means that, averaging 100 runs, it takes .16 seconds to guess the password, and 68 percent of the time (one standard deviation) it takes between .11 (.16 - .05) and .21 (.16 + .05) seconds.

1. Summary

we built a simple genetic engine that makes use of random mutation to produce better results. This engine was able to guess a secret password given only its length, a set of characters that might be in the password, and a fitness function that returns a count of the number characters in the guess that match the secret. This is a good benchmark project for the engine because as the target string gets longer the engine wastes more and more guesses trying to change positions that are already correct.

1. Sources

Github:

https://github.com/handcraftsman/GeneticAlgorithmsWithPython/tree/master/ch01

My Github repository:

https://github.com/redcartel243/project-files

Book: book Genetic Algorithms with Python by Clinton Sheppard