PES
UNIVERSITY

Project Work :
UE15CS492(Minor)

Final ISA(Review 5) / ESA 2019

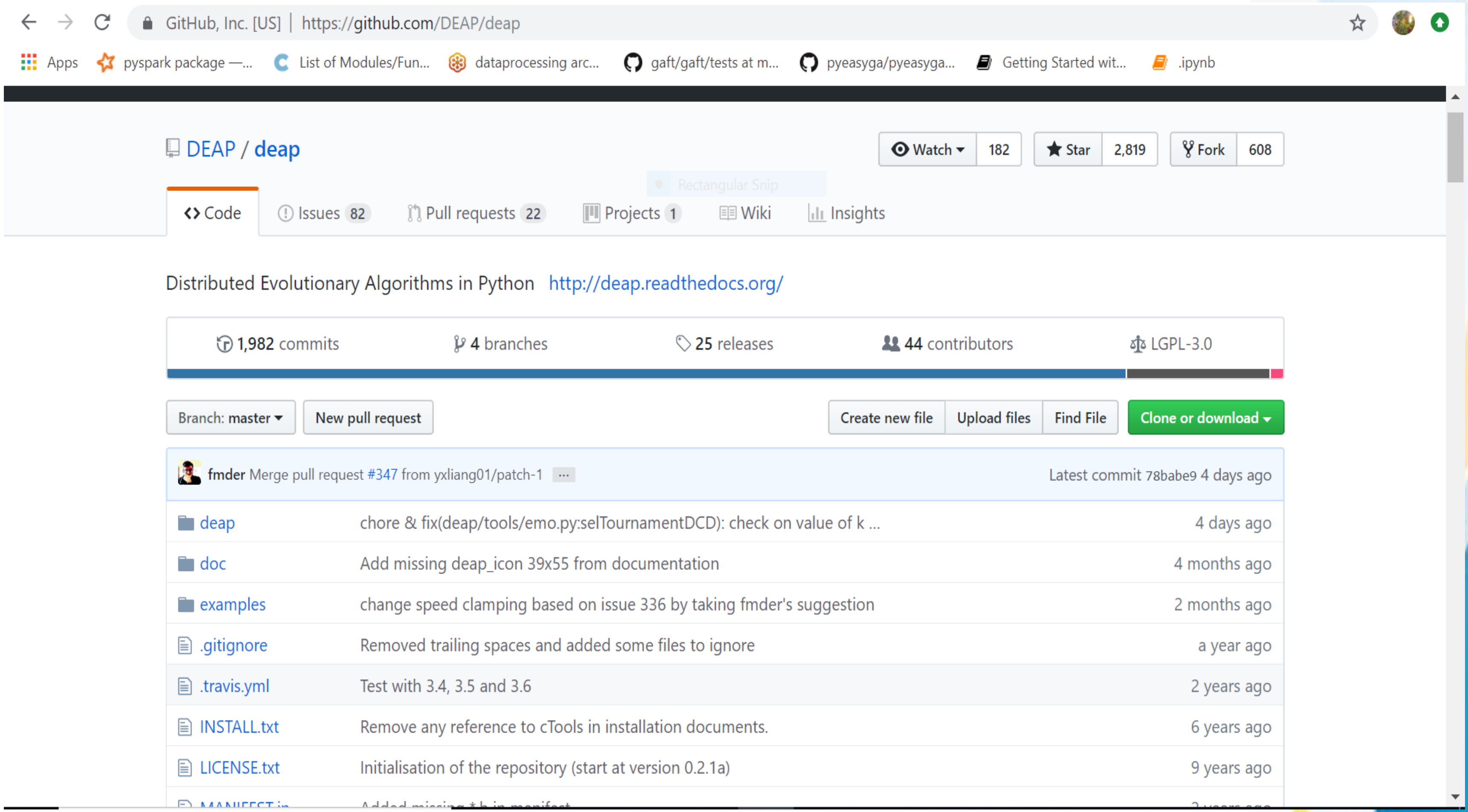
Project Title	:	Efficient Python Genetic Algorithm Framework	
Project ID	:	PW19CGM01	
Project Guide	:	Ms. Chitra G.M.	
Project Team	:	Bharatraj S Telkar	(01FB15ECS066)
		Daniel I	(01FB15ECS086)
		Shreyas Vivek Patil	(01FB15ECS286)

Genetic Algorithms: An Overview

White board discussion

So what’s this project about.. ?

Currently Available Python GA APIs...



GitHub, Inc. [US] | https://github.com/DEAP/deap

DEAP / deap

Watch 182 | Star 2,819 | Fork 608

Code | Issues 82 | Pull requests 22 | Projects 1 | Wiki | Insights

Distributed Evolutionary Algorithms in Python http://deap.readthedocs.org/

1,982 commits | 4 branches | 25 releases | 44 contributors | LGPL-3.0

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

fmder Merge pull request #347 from yxliang01/patch-1 ... Latest commit 78babe9 4 days ago

deap	chore & fix(deap/tools/emo.py:selTournamentDCD): check on value of k ...	4 days ago
doc	Add missing deap_icon 39x55 from documentation	4 months ago
examples	change speed clamping based on issue 336 by taking fmder's suggestion	2 months ago
.gitignore	Removed trailing spaces and added some files to ignore	a year ago
.travis.yml	Test with 3.4, 3.5 and 3.6	2 years ago
INSTALL.txt	Remove any reference to cTools in installation documents.	6 years ago
LICENSE.txt	Initialisation of the repository (start at version 0.2.1a)	9 years ago

DEAP

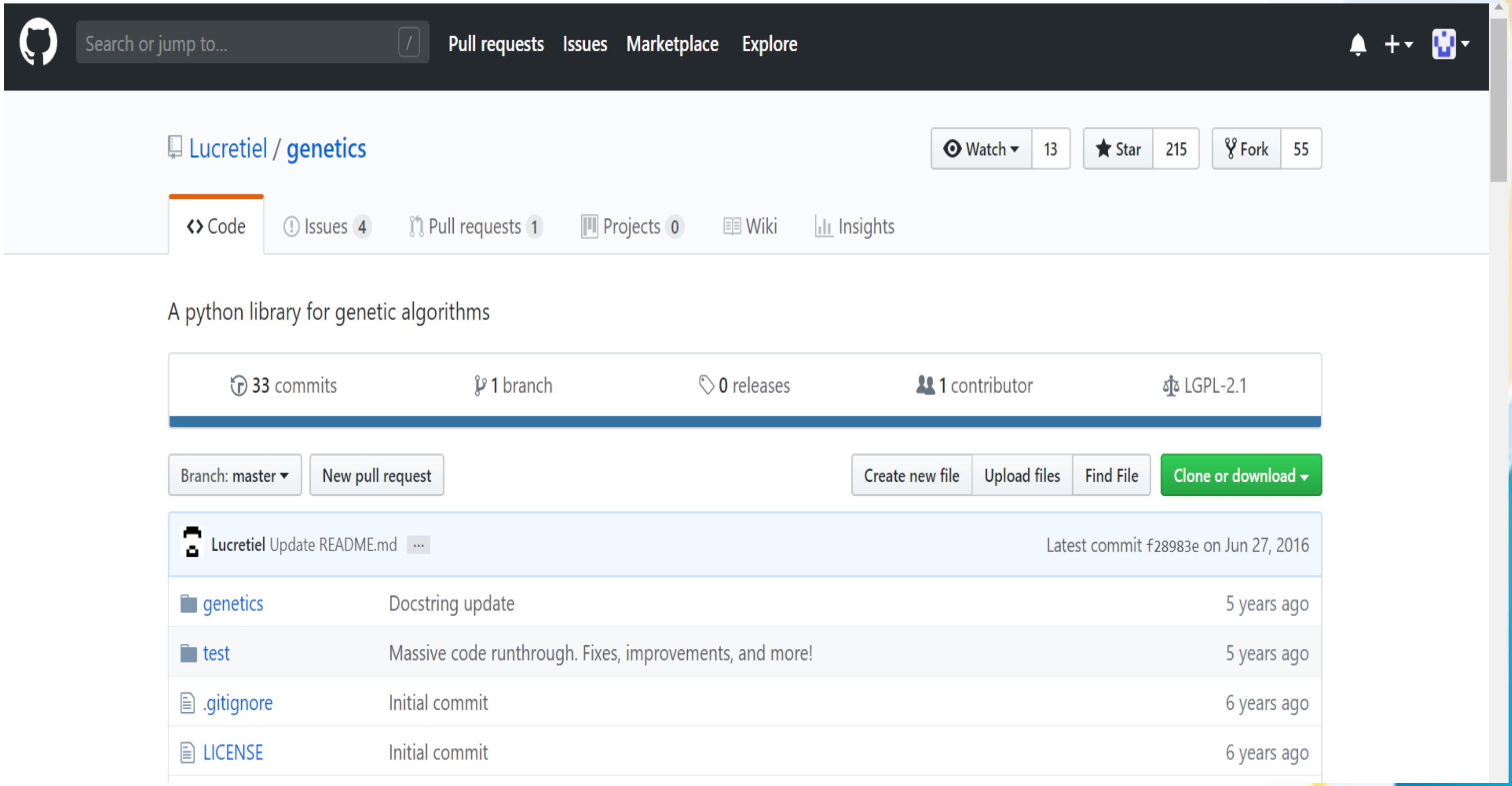
Advantages:

- Highly time efficient due to the use of parallelization mechanisms such as multiprocessing and SCOOP
- Highly generic

Disadvantages

- Although generic, bit poor in terms of user friendliness
- No famous optimizations
- Lack of high level API

Currently Available Python GA APIs…



Lucretiel / genetics

Watch 13 | Star 215 | Fork 55

Code | Issues 4 | Pull requests 1 | Projects 0 | Wiki | Insights

A python library for genetic algorithms

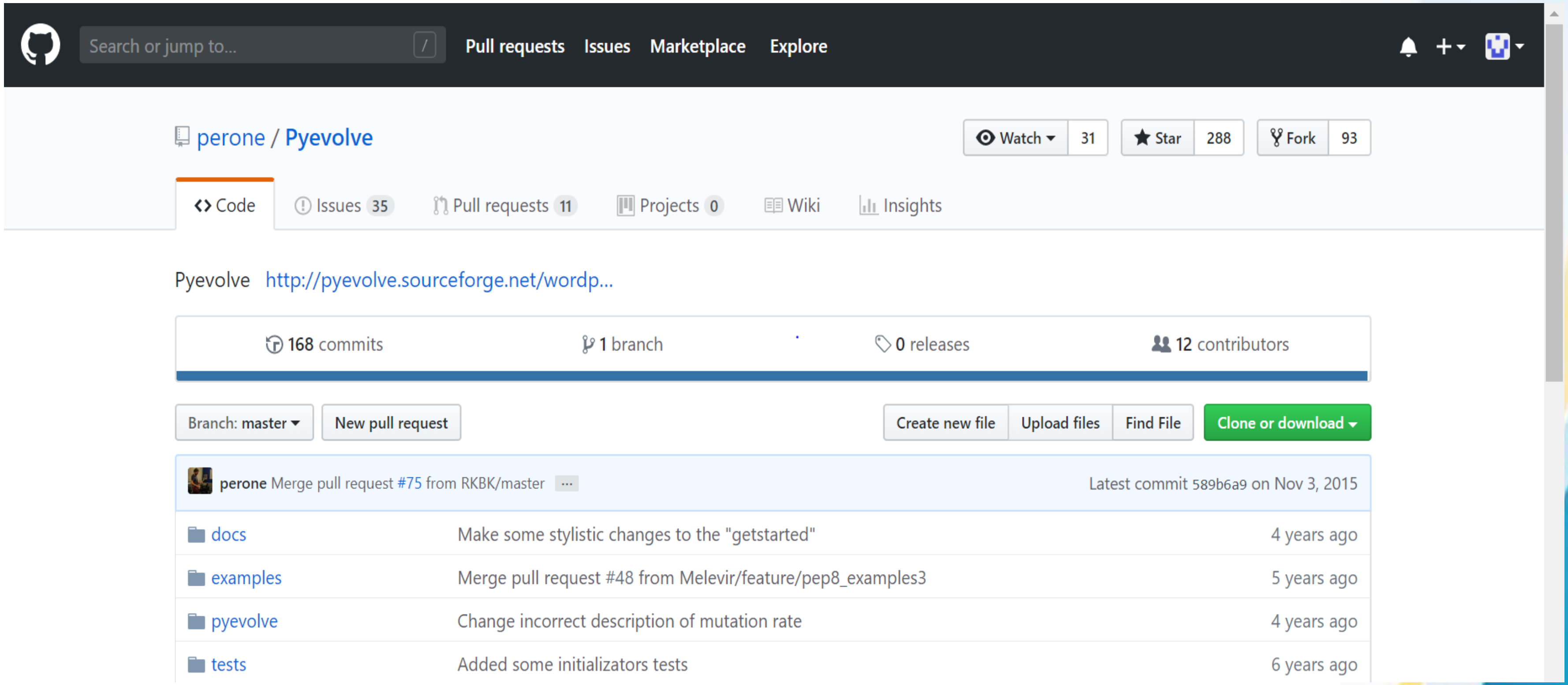
33 commits | 1 branch | 0 releases | 1 contributor | LGPL-2.1

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

Lucretiel Update README.md … Latest commit f28983e on Jun 27, 2016

genetics	Docstring update	5 years ago
test	Massive code runthrough. Fixes, improvements, and more!	5 years ago
.gitignore	Initial commit	6 years ago
LICENSE	Initial commit	6 years ago

Currently Available Python GA APIs…



perone / Pyevolve

Watch 31 | Star 288 | Fork 93

Code | Issues 35 | Pull requests 11 | Projects 0 | Wiki | Insights

Pyevolve http://pyevolve.sourceforge.net/wordp...

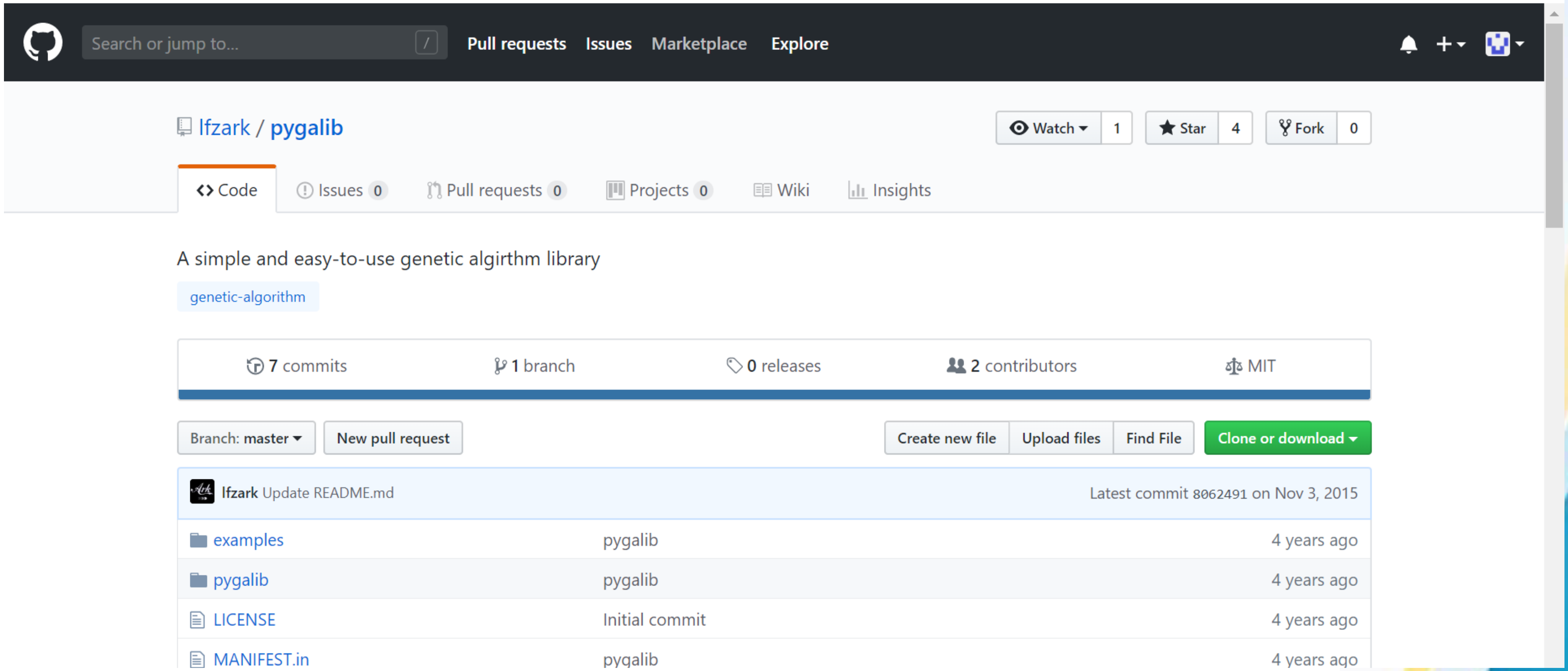
168 commits | 1 branch | 0 releases | 12 contributors

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

perone Merge pull request #75 from RKBK/master … Latest commit 589b6a9 on Nov 3, 2015

docs	Make some stylistic changes to the "getstarted"	4 years ago
examples	Merge pull request #48 from Melevir/feature/pep8_examples3	5 years ago
pyevolve	Change incorrect description of mutation rate	4 years ago
tests	Added some initializators tests	6 years ago

Currently Available Python GA APIs…



lfzark / pygalib

Watch 1 | Star 4 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights

A simple and easy-to-use genetic algirthm library

genetic-algorithm

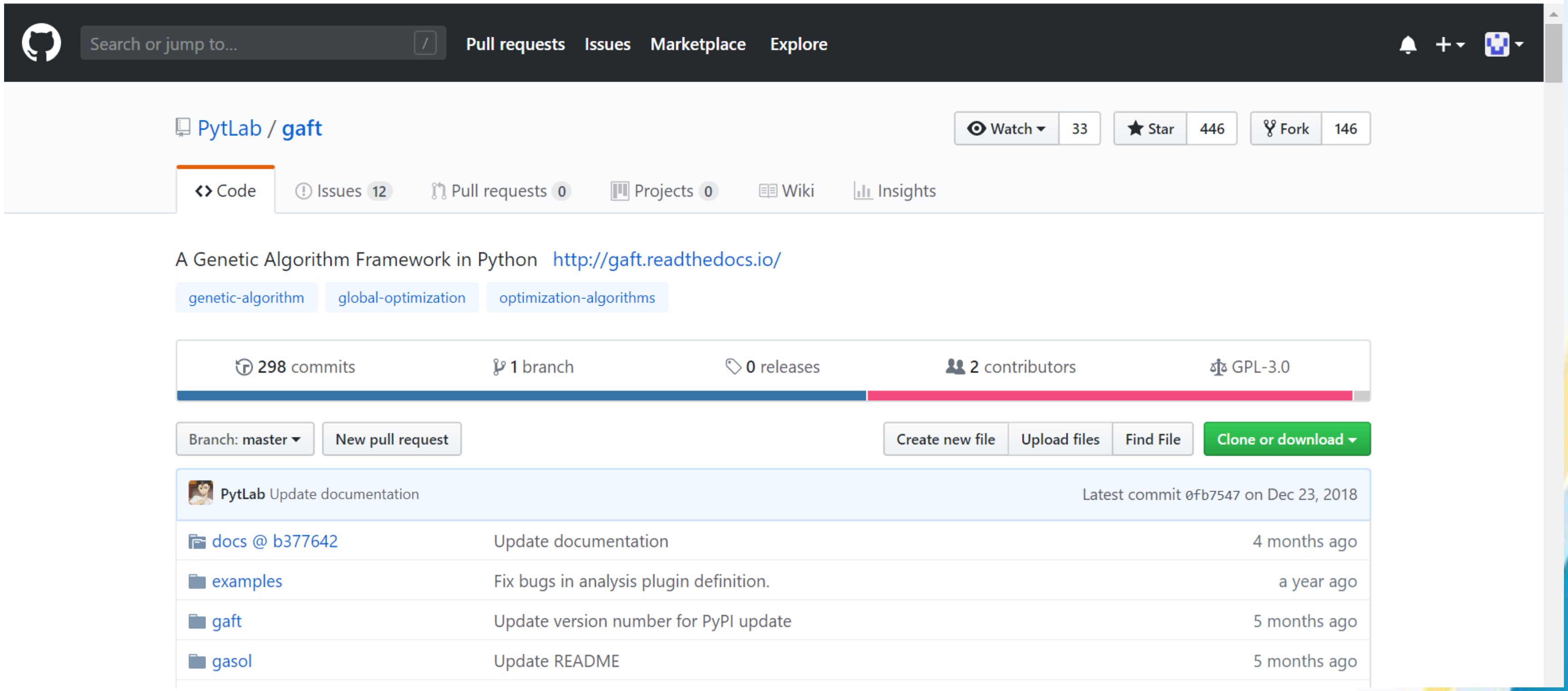
7 commits | 1 branch | 0 releases | 2 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

lfzark Update README.md — Latest commit 8062491 on Nov 3, 2015

examples	pygalib	4 years ago
pygalib	pygalib	4 years ago
LICENSE	Initial commit	4 years ago
MANIFEST.in	pygalib	4 years ago

Currently Available Python GA APIs…



Search or jump to... | Pull requests | Issues | Marketplace | Explore

PytLab / gaft

Watch 33 | Star 446 | Fork 146

Code | Issues 12 | Pull requests 0 | Projects 0 | Wiki | Insights

A Genetic Algorithm Framework in Python http://gaft.readthedocs.io/

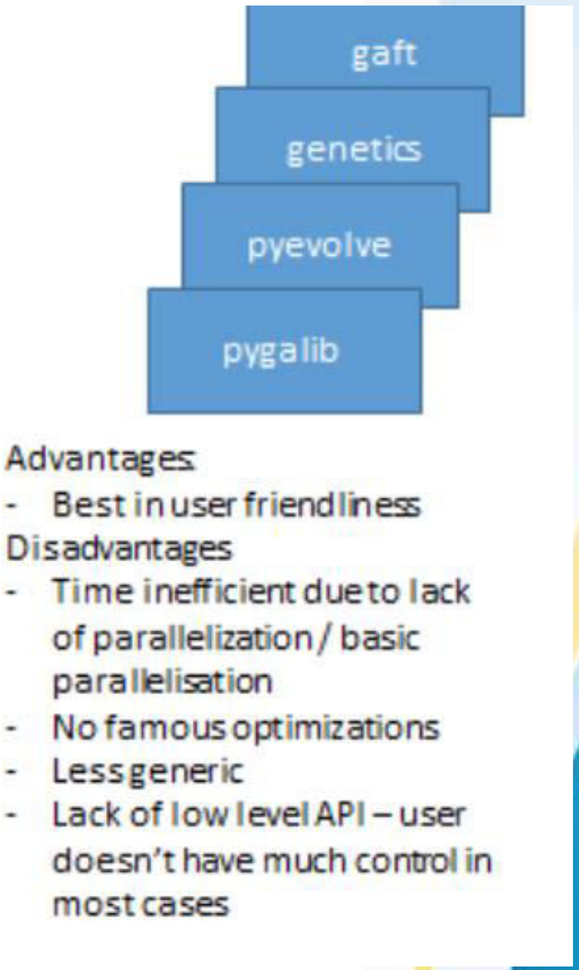
genetic-algorithm | global-optimization | optimization-algorithms

298 commits | 1 branch | 0 releases | 2 contributors | GPL-3.0

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

PytLab Update documentation — Latest commit 0fb7547 on Dec 23, 2018

docs @ b377642	Update documentation	4 months ago
examples	Fix bugs in analysis plugin definition.	a year ago
gaft	Update version number for PyPI update	5 months ago
gasol	Update README	5 months ago



gaft

genetics

pyevolve

pygalib

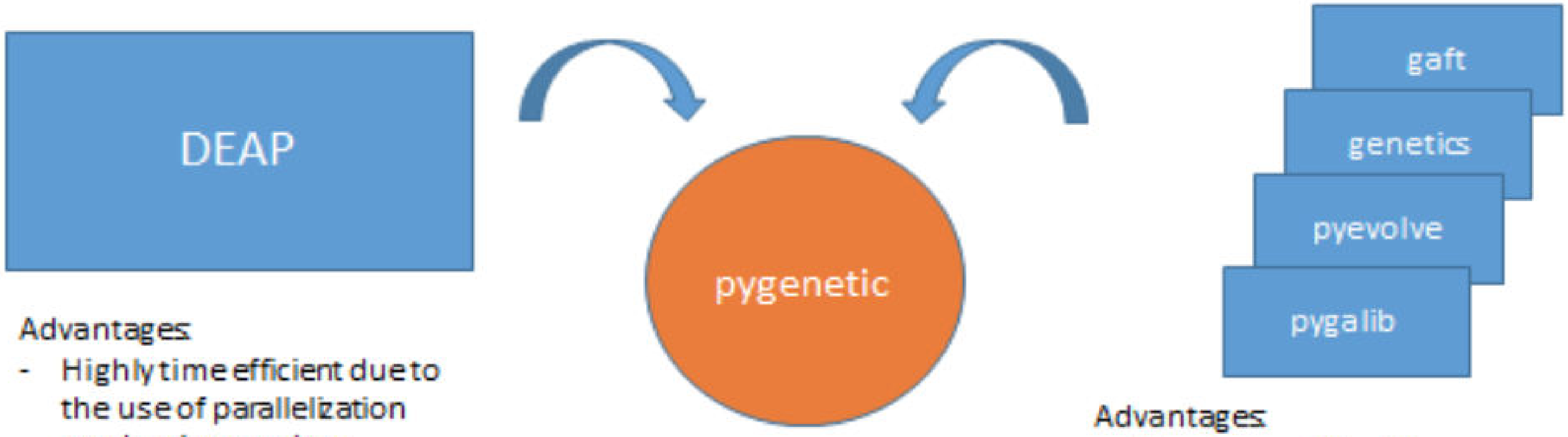
Advantages:

- Best in user friendliness

Disadvantages

- Time inefficient due to lack of parallelization / basic parallelisation
- No famous optimizations
- Less generic
- Lack of low level API – user doesn't have much control in most cases

Two worlds of Python Genetic Algorithm APIs



Advantages:
- Highly time efficient due to the use of parallelization mechanisms such as multiprocessing and SCOOP
- Highly generic

Disadvantages
- Although generic, bit poor in terms of user friendliness
- No famous optimizations
- Lack of high level API

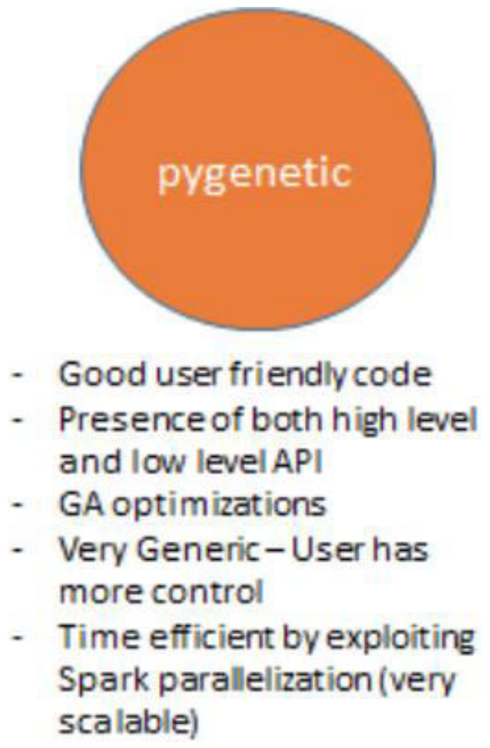
- Good user friendly code
- Presence of both high level and low level API
- GA optimizations
- Very Generic – User has more control
- Time efficient by exploiting Spark parallelization (very scalable)

Advantages:
- Best in user friendliness

Disadvantages
- Time inefficient due to lack of parallelization / basic parallelisation
- No famous optimizations
- Less generic
- Lack of low level API – user doesn't have much control in most cases

Problem Statement

- To develop a highly efficient, usable and generic genetic algorithm python framework “pygenetic”



- Good user friendly code
- Presence of both high level and low level API
- GA optimizations
- Very Generic – User has more control
- Time efficient by exploiting Spark parallelization (very scalable)

API Overview

- Presence of both high level and low level API
- GA Parallelization support using Apache Spark
- GA Optimisations
- ML using GA – ANN Topology Finder using GA

PES UNIVERSITY

User Profile

Students, teachers, researchers, company employees / entrepreneurs can all use our genetic algorithm framework while experimenting with different Machine Learning Algorithms and observing performance. They can also play around and simulate different Genetic Algorithms online on our website

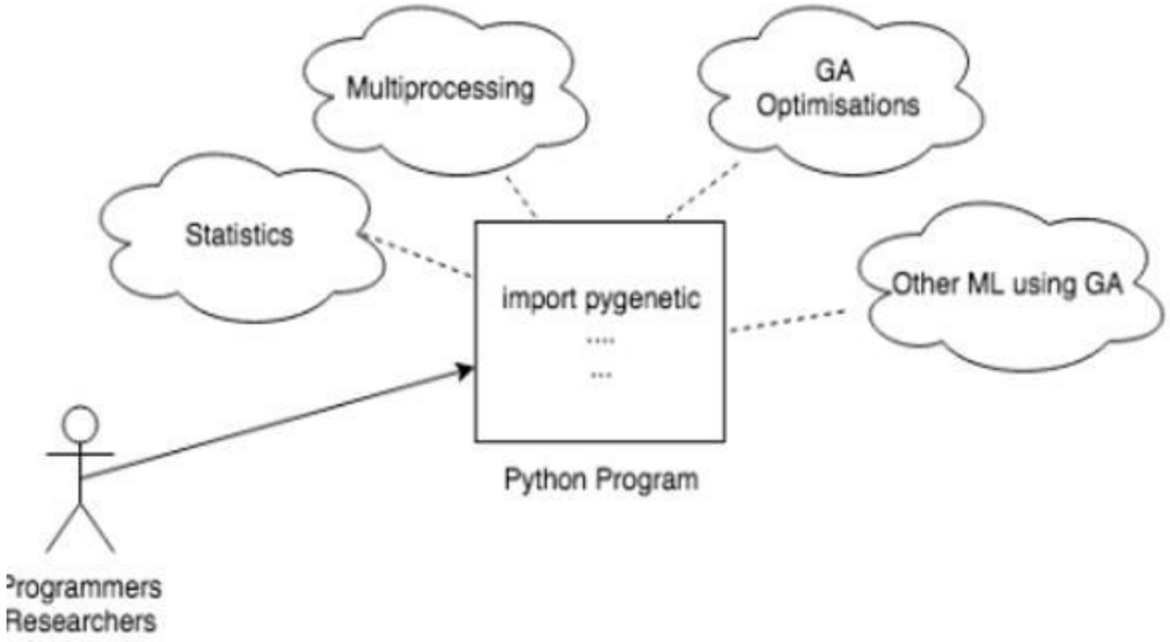
Literature Survey

S.No	Paper	Author	Inference
1.	*Scaling genetic algorithms using map reduce*	Verma, Abhishek, Xavier	Basic GA using Map Reduce
2.	*Evolve a neural network with a genetic algorithm*	Matt Harvey	Choosing efficient ANN hyperparameters

Literature Survey

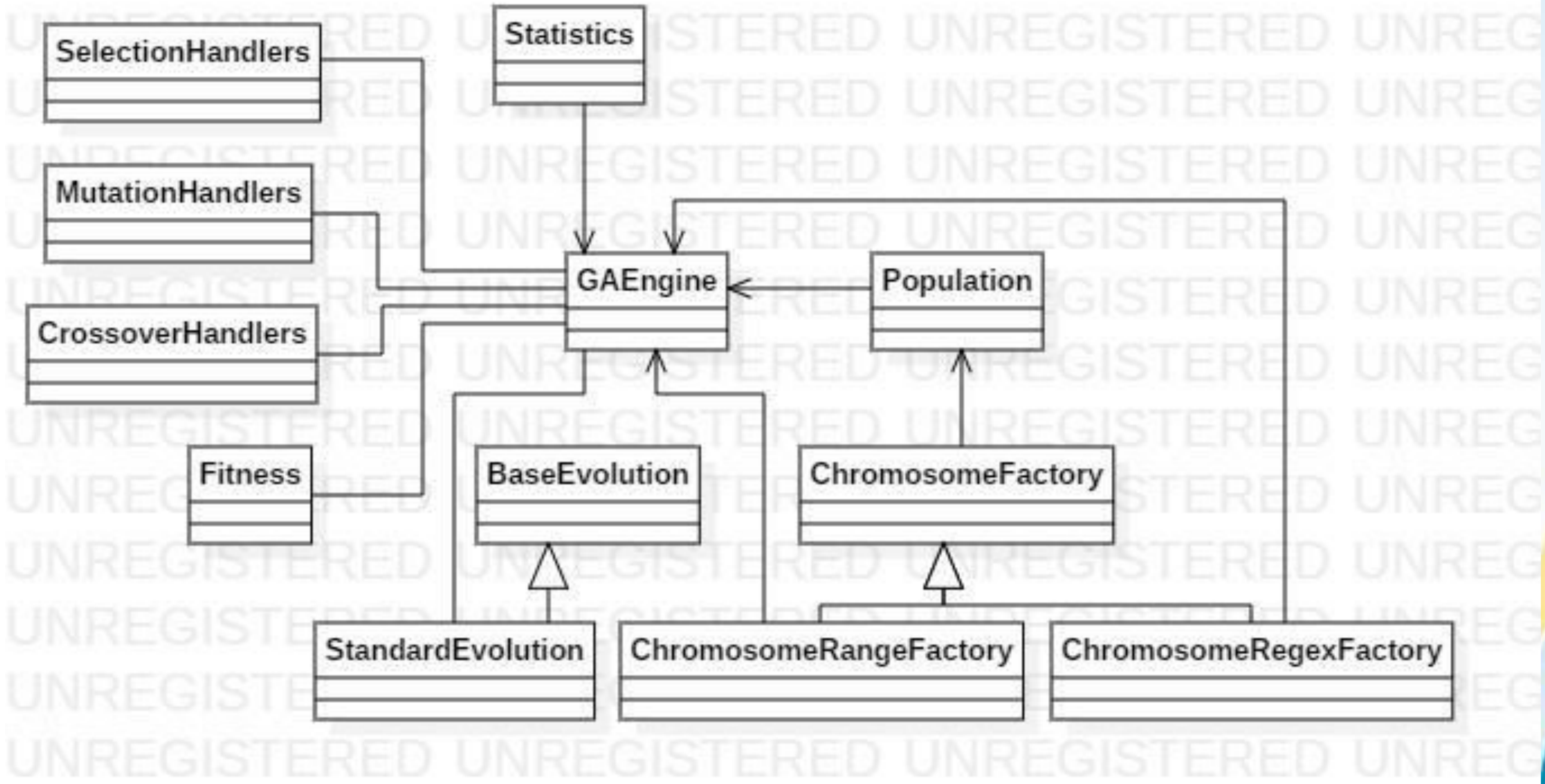
S.No	Paper	Author	Inference
3.	*Improving genetic algorithms' efficiency using intelligent fitness functions*	Jason Cooper, Chris Hinde	Improve efficiency using storage
4.	*An Adaptive Genetic Algorithm based on Population Diversity strategy*	Chen Lin	Adaptive mutation rate
5.	*A Parallel Genetic Algorithm Based on Hadoop MapReduce for the Automatic Generation of JUnit Test Suites*	Linda, Ferrucci, Alflonso, Saro	Basic Mapper and Reducer

Modules


Multiprocessing
GA Optimisations
Statistics
import pygenetic
....
...
Other ML using GA
Python Program
Programmers
Researchers

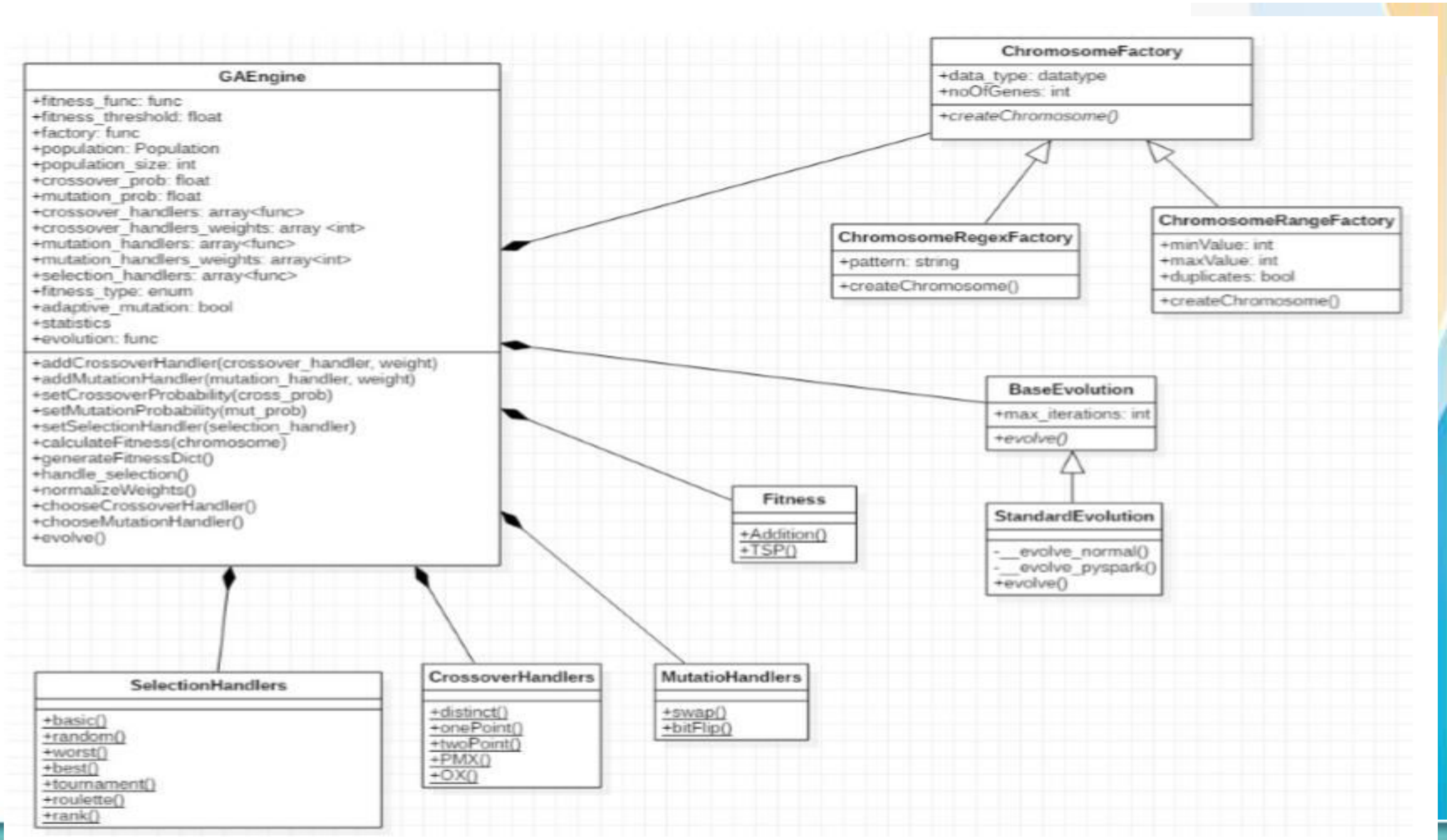
Pygenetic: Making the Design

Design Description



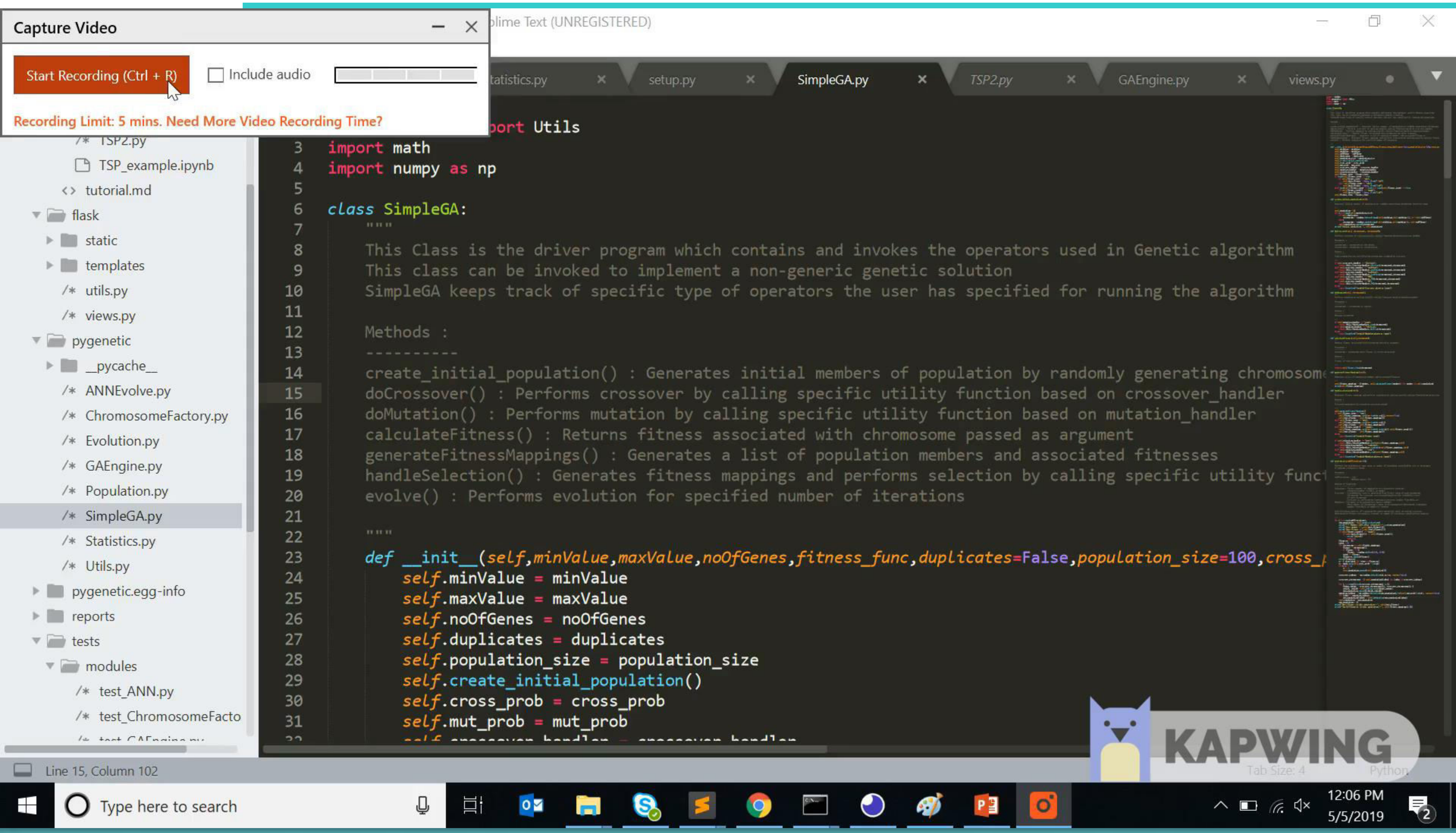
PES UNIVERSITY

Design Description



Pygenetic features

1. Presence of both High and Low Level APIs for GA Execution



Recording Limit: 5 mins. Need More Video Recording Time?

```python
port Utils
import math
import numpy as np

class SimpleGA:
    """
    This Class is the driver program which contains and invokes the operators used in Genetic algorithm
    This class can be invoked to implement a non-generic genetic solution
    SimpleGA keeps track of specific type of operators the user has specified for running the algorithm

    Methods :
    ----------
    create_initial_population() : Generates initial members of population by randomly generating chromosome
    doCrossover() : Performs crossover by calling specific utility function based on crossover_handler
    doMutation() : Performs mutation by calling specific utility function based on mutation_handler
    calculateFitness() : Returns fitness associated with chromosome passed as argument
    generateFitnessMappings() : Generates a list of population members and associated fitnesses
    handleSelection() : Generates fitness mappings and performs selection by calling specific utility funct
    evolve() : Performs evolution for specified number of iterations

    """

    def __init__(self,minValue,maxValue,noOfGenes,fitness_func,duplicates=False,population_size=100,cross_
        self.minValue = minValue
        self.maxValue = maxValue
        self.noOfGenes = noOfGenes
        self.duplicates = duplicates
        self.population_size = population_size
        self.create_initial_population()
        self.cross_prob = cross_prob
        self.mut_prob = mut_prob
        self.crossover_handler = crossover_handler
```

1. Presence of both High and Low Level APIs for GA Execution

Users can use the very easy to use SimpleGA for execution of Simple Genetic Algorithms

- Only write One line of code !!
- Lacks most of the advanced features of GAEngine

Users can implement all possible GAs using the generic GAEngine module

- Very generic + User friendly
- Implements GA Optimisations and other advanced features

2. Very Generic API

- The GAEngine Module is very generic in nature.

- Users can easily define
  - Custom Chromosome Factories
  - Custom Selection, Crossover, Mutation, Fitness handlers.
  - Even, Custom Evolutions !!

2. Very Generic API

- Custom Chromosome Factories

pygenetic supports two types of ChromosomeFactories

- ChromosomeRegexFactory: for creating chromosomes whose genes follow a particular regex
- ChromosomeRangeFactory: for creating chromosomes whose genes are between some numeric interval

Users can easily define custom factories by subclassing ChromosomeFactory.

2. Very Generic API

```
>>> class CustomFactory(ChromosomeFactory.ChromosomeFactory):
...     def __init__(self,noOfGenes,input_list):
...             self.noOfGenes = noOfGenes
...             self.input_list = input_list
...     def createChromosome(self):
...             return
random.sample(self.input_list,self.noOfGenes)
...
>>> factory =
CustomFactory(noOfGenes=5,input_list=['duck','cow',

'monkey','giraffe','dog','cat','peacock','mice','sun'])
>>> factory.createChromosome()
['mice', 'giraffe', 'cow', 'dog', 'cat']
```

2. Very Generic API

- Custom Selection, Crossover, Mutation, Fitness handlers.

`custom_function(fitness_mappings, ga)` is provided as a function signature for end users to implement custom selection handlers

`custom_function(chromosome1, chromosome2)` is provided as a function signature for end users to implement custom crossover and mutation handlers

2. Very Generic API

- Custom Selection, Crossover, Mutation, Fitness handlers.

`custom_function(chromosome)` is provided as a function signature for end users to implement custom fitness handlers

Note:

1. Very generic function signatures.
2. Support for extra parameters

2. Very Generic API

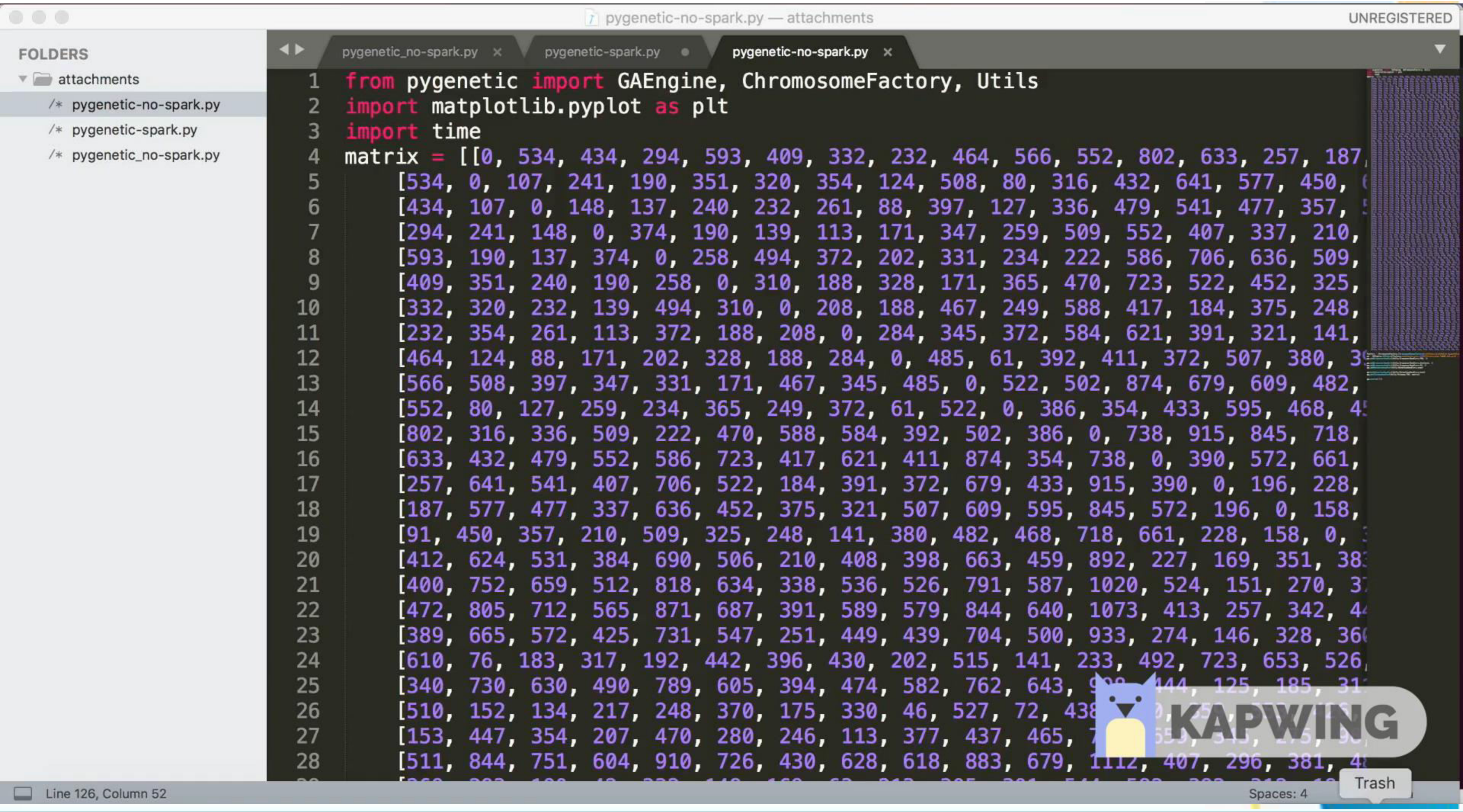
- Even Custom Evolutions !!

- Users can define a custom evolution by subclassing BaseEvolution and filling ga.population.new_members with the new members from the evolution in the def evolve(self,ga) function.
- Return 1 from this function if the required fitness value is found else no need to return anything

2. Very Generic API

```
➢ from pygenetic import Evolution
➢ class CustomEvolution(Evolution.BaseEvolution):
➢   def __init__(self,...):
➢     ....
➢   def evolve(self,ga):
➢     # Carry out custom evolution
➢     # Current population is at ga.population.members
➢     ### ga.handle_selection() does the selection using the given
  selection handler
➢     ### Fitness mappings are present at ga.fitness_mappings
➢     ### ga.chooseCrossoverHandler() chooses
➢     ### ga.doCrossover(crossoverHandler,father,mother) executes
  crossover
➢     ### ga.chooseMutationHandler() chooses
➢     ### ga.doMutation(mutationHandler,chromosome) does mutation
➢     # Fill ga.population.new_members with the new population from
  evolution
➢     # Return 1 if the required fitness value is found
```

3. Supports parallelization using Apache Spark



pygenetic-no-spark.py — attachments

UNREGISTERED

FOLDERS
- attachments
  - pygenetic-no-spark.py
  - pygenetic-spark.py
  - pygenetic_no-spark.py

pygenetic_no-spark.py | pygenetic-spark.py | pygenetic-no-spark.py

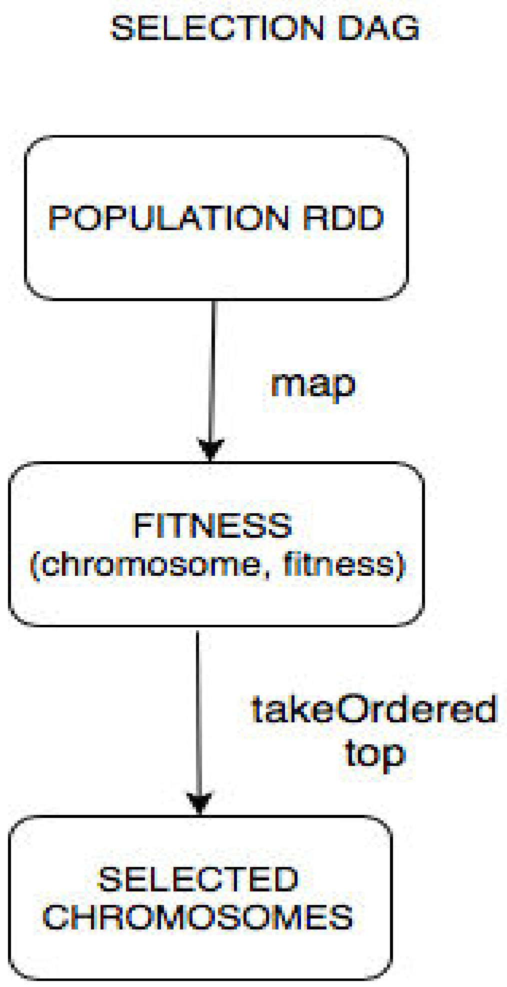
```
from pygenetic import GAEngine, ChromosomeFactory, Utils
import matplotlib.pyplot as plt
import time
matrix = [[0, 534, 434, 294, 593, 409, 332, 232, 464, 566, 552, 802, 633, 257, 187,
          [534, 0, 107, 241, 190, 351, 320, 354, 124, 508, 80, 316, 432, 641, 577, 450,
          [434, 107, 0, 148, 137, 240, 232, 261, 88, 397, 127, 336, 479, 541, 477, 357,
          [294, 241, 148, 0, 374, 190, 139, 113, 171, 347, 259, 509, 552, 407, 337, 210,
          [593, 190, 137, 374, 0, 258, 494, 372, 202, 331, 234, 222, 586, 706, 636, 509,
          [409, 351, 240, 190, 258, 0, 310, 188, 328, 171, 365, 470, 723, 522, 452, 325,
          [332, 320, 232, 139, 494, 310, 0, 208, 188, 467, 249, 588, 417, 184, 375, 248,
          [232, 354, 261, 113, 372, 188, 208, 0, 284, 345, 372, 584, 621, 391, 321, 141,
          [464, 124, 88, 171, 202, 328, 188, 284, 0, 485, 61, 392, 411, 372, 507, 380, 3
          [566, 508, 397, 347, 331, 171, 467, 345, 485, 0, 522, 502, 874, 679, 609, 482,
          [552, 80, 127, 259, 234, 365, 249, 372, 61, 522, 0, 386, 354, 433, 595, 468, 4
          [802, 316, 336, 509, 222, 470, 588, 584, 392, 502, 386, 0, 738, 915, 845, 718,
          [633, 432, 479, 552, 586, 723, 417, 621, 411, 874, 354, 738, 0, 390, 572, 661,
          [257, 641, 541, 407, 706, 522, 184, 391, 372, 679, 433, 915, 390, 0, 196, 228,
          [187, 577, 477, 337, 636, 452, 375, 321, 507, 609, 595, 845, 572, 196, 0, 158,
          [91, 450, 357, 210, 509, 325, 248, 141, 380, 482, 468, 718, 661, 228, 158, 0,
          [412, 624, 531, 384, 690, 506, 210, 408, 398, 663, 459, 892, 227, 169, 351, 38
          [400, 752, 659, 512, 818, 634, 338, 536, 526, 791, 587, 1020, 524, 151, 270, 3
          [472, 805, 712, 565, 871, 687, 391, 589, 579, 844, 640, 1073, 413, 257, 342, 4
          [389, 665, 572, 425, 731, 547, 251, 449, 439, 704, 500, 933, 274, 146, 328, 36
          [610, 76, 183, 317, 192, 442, 396, 430, 202, 515, 141, 233, 492, 723, 653, 526,
          [340, 730, 630, 490, 789, 605, 394, 474, 582, 762, 643, [illegible] 444, 125, 185, 31
          [510, 152, 134, 217, 248, 370, 175, 330, 46, 527, 72, 438 [illegible]
          [153, 447, 354, 207, 470, 280, 246, 113, 377, 437, 465, [illegible]
          [511, 844, 751, 604, 910, 726, 430, 628, 618, 883, 679, 1112, 407, 296, 381, 4
```

Line 126, Column 52 | Spaces: 4

How does it Work So Fast??

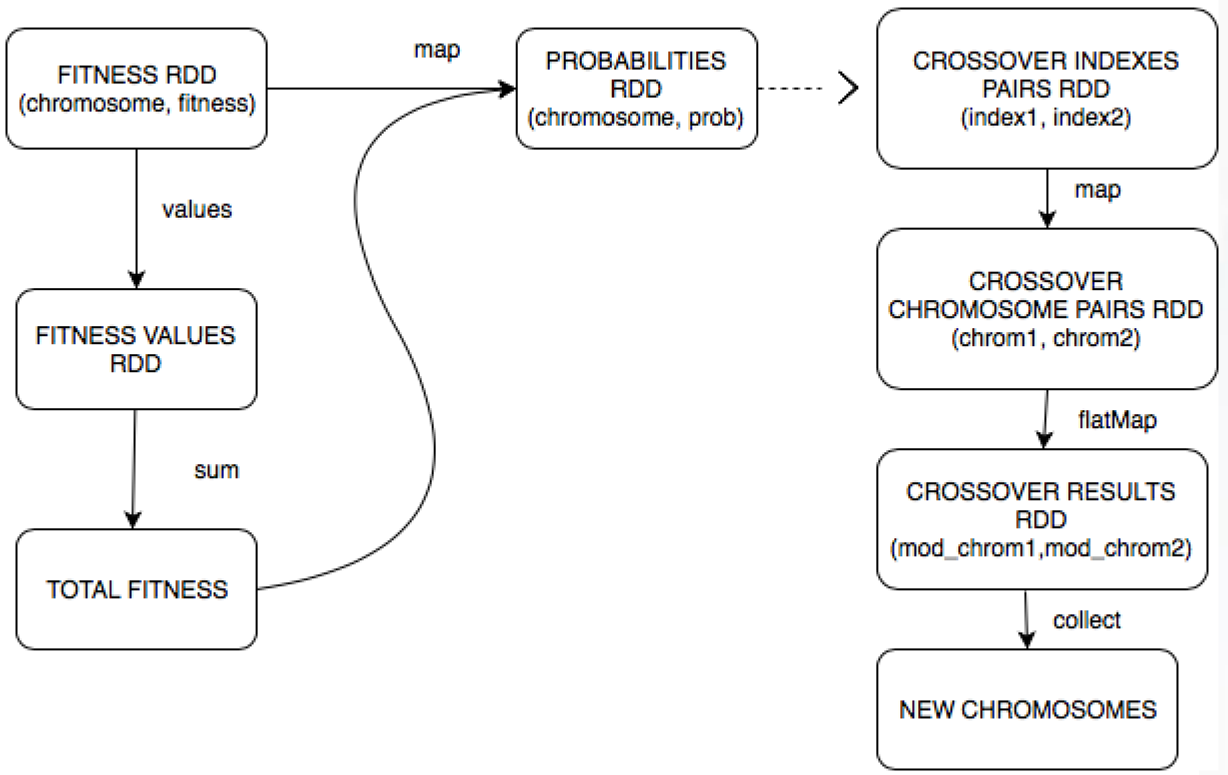
- We leverage Apache Spark to parallelize all the fitness calculation, selection, crossover and mutation operations based on the architecture we created.

Spark DAG Design Description: Selection

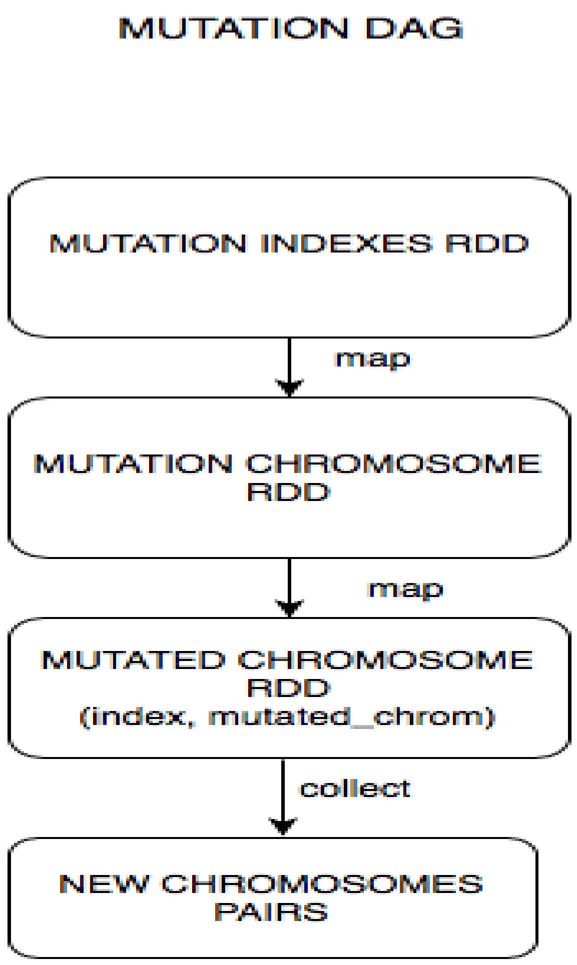


Spark DAG Design Description: Crossover

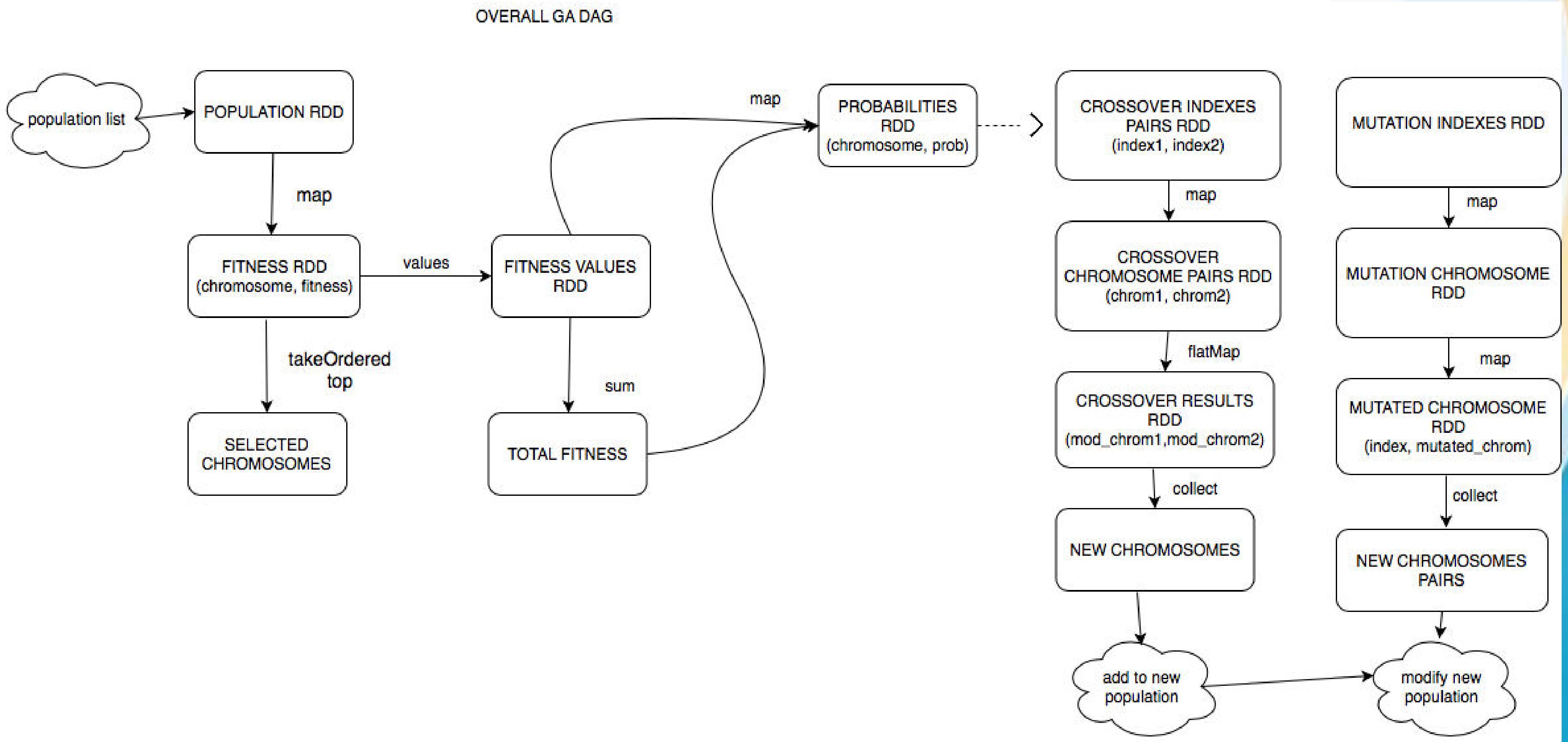
CROSSOVER DAG


FITNESS RDD (chromosome, fitness)
map
PROBABILITIES RDD (chromosome, prob)
CROSSOVER INDEXES PAIRS RDD (index1, index2)
values
map
FITNESS VALUES RDD
CROSSOVER CHROMOSOME PAIRS RDD (chrom1, chrom2)
flatMap
sum
CROSSOVER RESULTS RDD (mod_chrom1,mod_chrom2)
TOTAL FITNESS
collect
NEW CHROMOSOMES

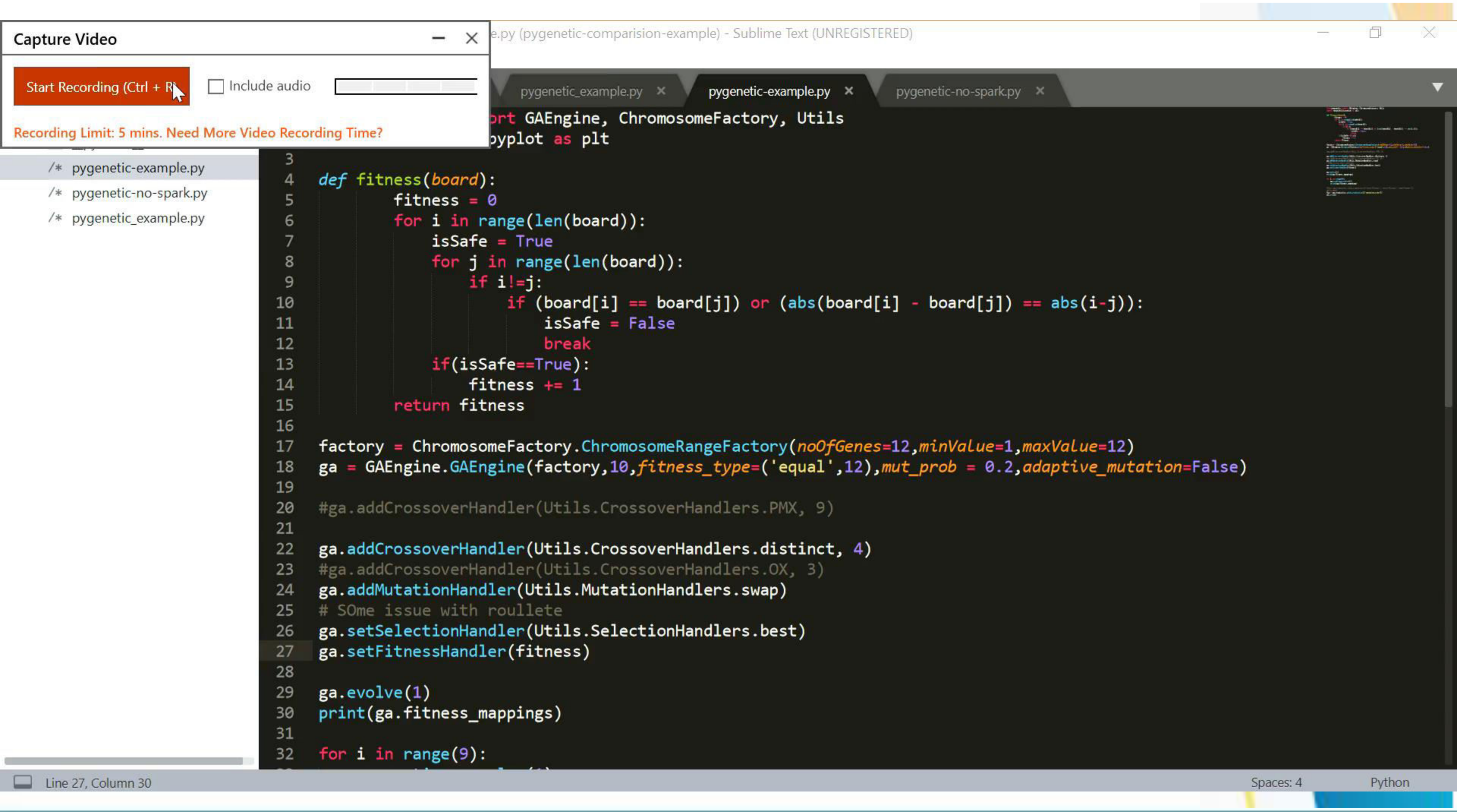
Spark DAG Design Description: Mutation



Spark DAG Design Description: Overall



4. Supports Adaptive Mutation Rates


Capture Video
Start Recording (Ctrl + R)
Include audio
Recording Limit: 5 mins. Need More Video Recording Time?
e.py (pygenetic-comparision-example) - Sublime Text (UNREGISTERED)
pygenetic_example.py
pygenetic-example.py
pygenetic-no-spark.py
Line 27, Column 30
Spaces: 4
Python

```python
ort GAEngine, ChromosomeFactory, Utils
pyplot as plt

def fitness(board):
        fitness = 0
        for i in range(len(board)):
            isSafe = True
            for j in range(len(board)):
                if i!=j:
                    if (board[i] == board[j]) or (abs(board[i] - board[j]) == abs(i-j)):
                        isSafe = False
                        break
            if(isSafe==True):
                fitness += 1
        return fitness

factory = ChromosomeFactory.ChromosomeRangeFactory(noOfGenes=12,minValue=1,maxValue=12)
ga = GAEngine.GAEngine(factory,10,fitness_type=('equal',12),mut_prob = 0.2,adaptive_mutation=False)

#ga.addCrossoverHandler(Utils.CrossoverHandlers.PMX, 9)

ga.addCrossoverHandler(Utils.CrossoverHandlers.distinct, 4)
#ga.addCrossoverHandler(Utils.CrossoverHandlers.OX, 3)
ga.addMutationHandler(Utils.MutationHandlers.swap)
# SOme issue with roullete
ga.setSelectionHandler(Utils.SelectionHandlers.best)
ga.setFitnessHandler(fitness)

ga.evolve(1)
print(ga.fitness_mappings)

for i in range(9):
```

4. Supports Adaptive Mutation Rates

- Having Adaptive Mutation Rates are more advantageous:

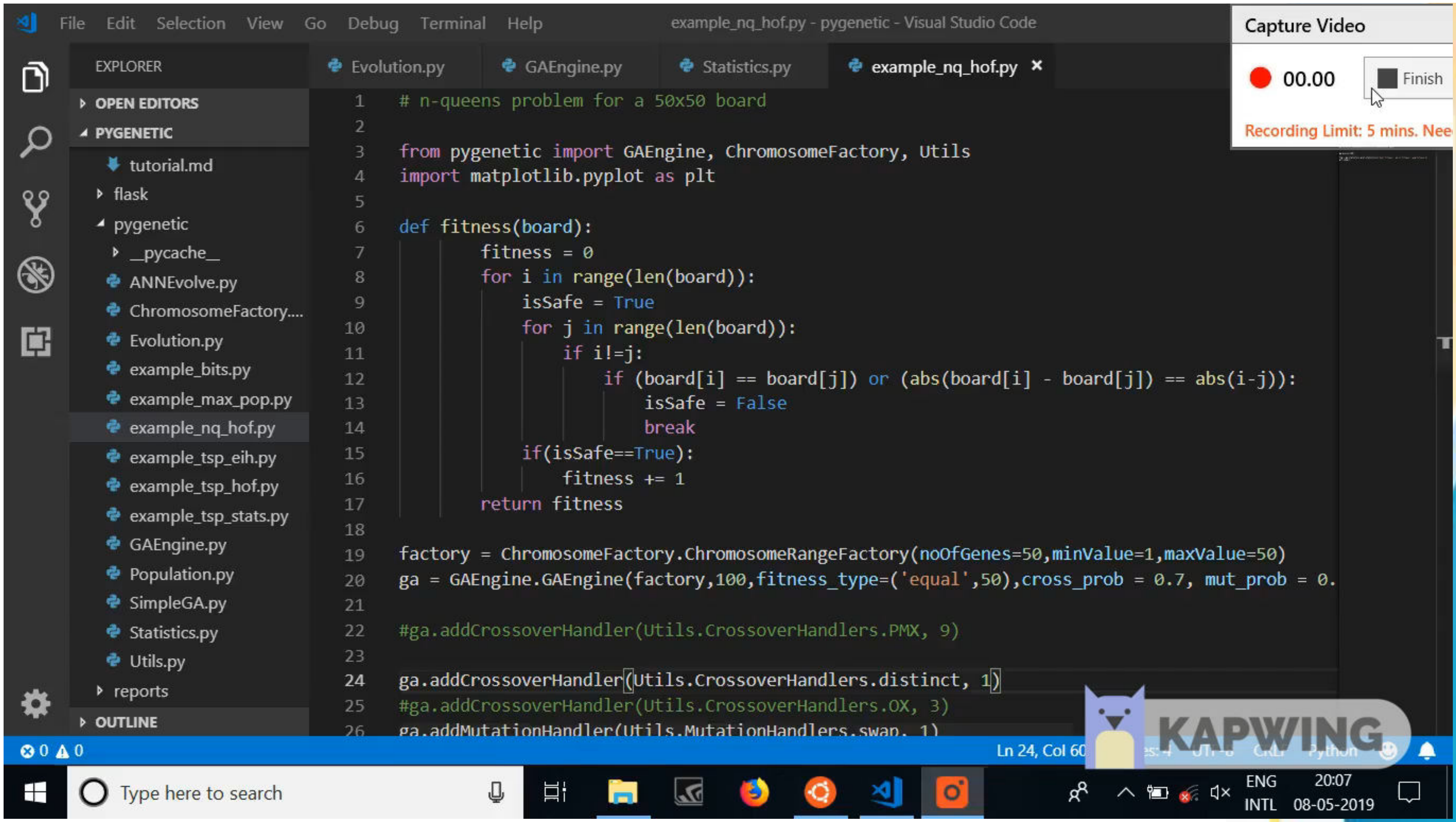
a. Low Diversity -> Increase the mutation to get more diversity (else, too much similar population)

b. High Diversity -> Decrease the mutation to get lesser diversity (else, too random)

c. Prevents stagnating at local minima

$$\boxed{p_m = M_a * \left(1 + \frac{f_{\max} - ASD_t}{f_{\max} + ASD_t}\right)}$$

5. Supports Hall of Fame Injection



```python
# n-queens problem for a 50x50 board

from pygenetic import GAEngine, ChromosomeFactory, Utils
import matplotlib.pyplot as plt

def fitness(board):
        fitness = 0
        for i in range(len(board)):
            isSafe = True
            for j in range(len(board)):
                if i!=j:
                    if (board[i] == board[j]) or (abs(board[i] - board[j]) == abs(i-j)):
                        isSafe = False
                        break
            if(isSafe==True):
                fitness += 1
        return fitness

factory = ChromosomeFactory.ChromosomeRangeFactory(noOfGenes=50,minValue=1,maxValue=50)
ga = GAEngine.GAEngine(factory,100,fitness_type=('equal',50),cross_prob = 0.7, mut_prob = 0.

#ga.addCrossoverHandler(Utils.CrossoverHandlers.PMX, 9)

ga.addCrossoverHandler(Utils.CrossoverHandlers.distinct, 1)
#ga.addCrossoverHandler(Utils.CrossoverHandlers.OX, 3)
ga.addMutationHandler(Utils.MutationHandlers.swap, 1)
```

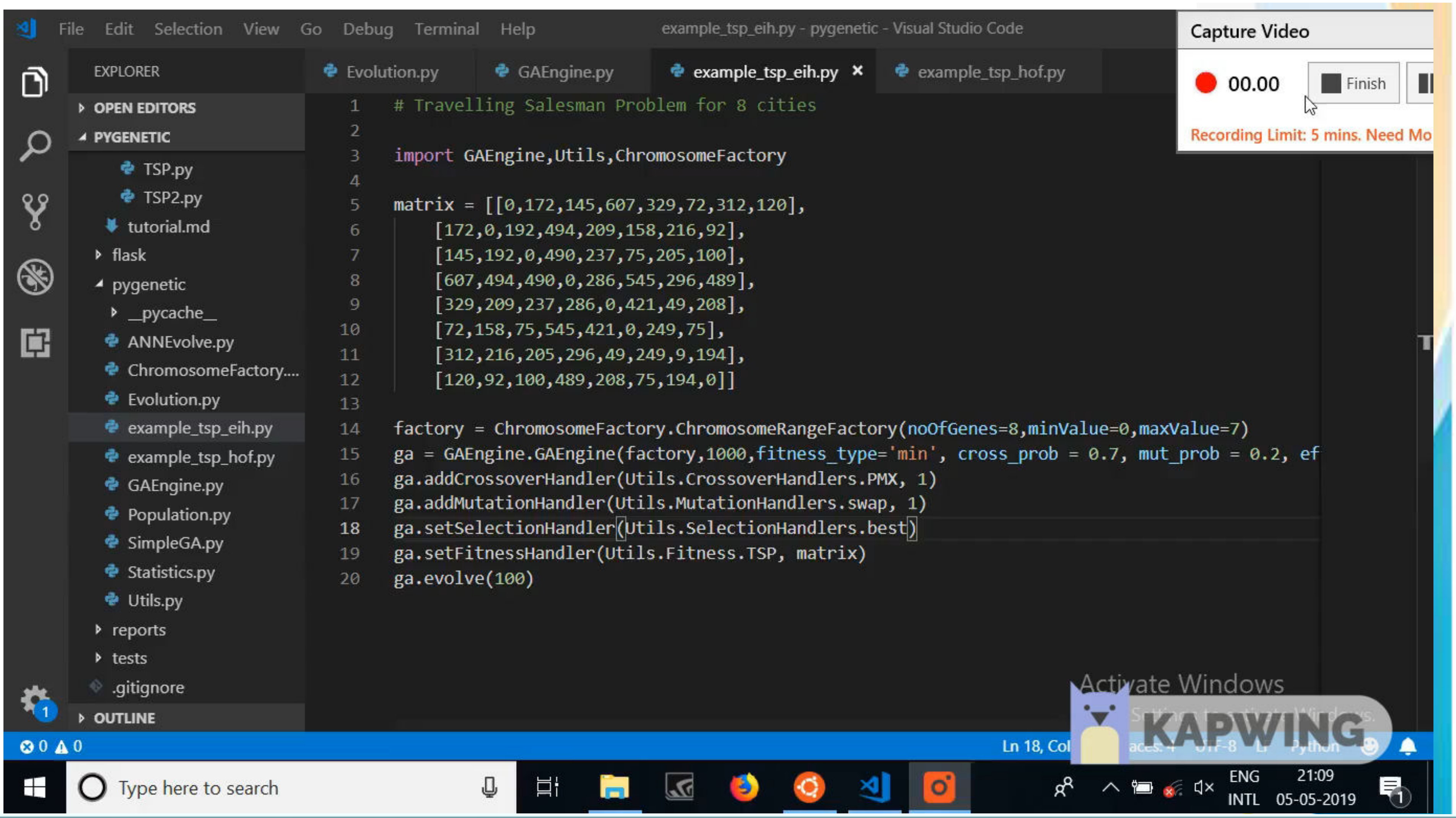
5. Supports Hall of Fame Injection

- The best encountered chromosome(Hall of fame chromosome) is injected back into the population after every 20 generations.

- Philosophy: The GA shouldn’t lose the best chromosomes while doing different selection methods or having high mut_prob over multiple generations

- This optimization improves the chances of HOF chromosome being re-added to population to produce better off springs

6. Supports Efficient Iteration Halt



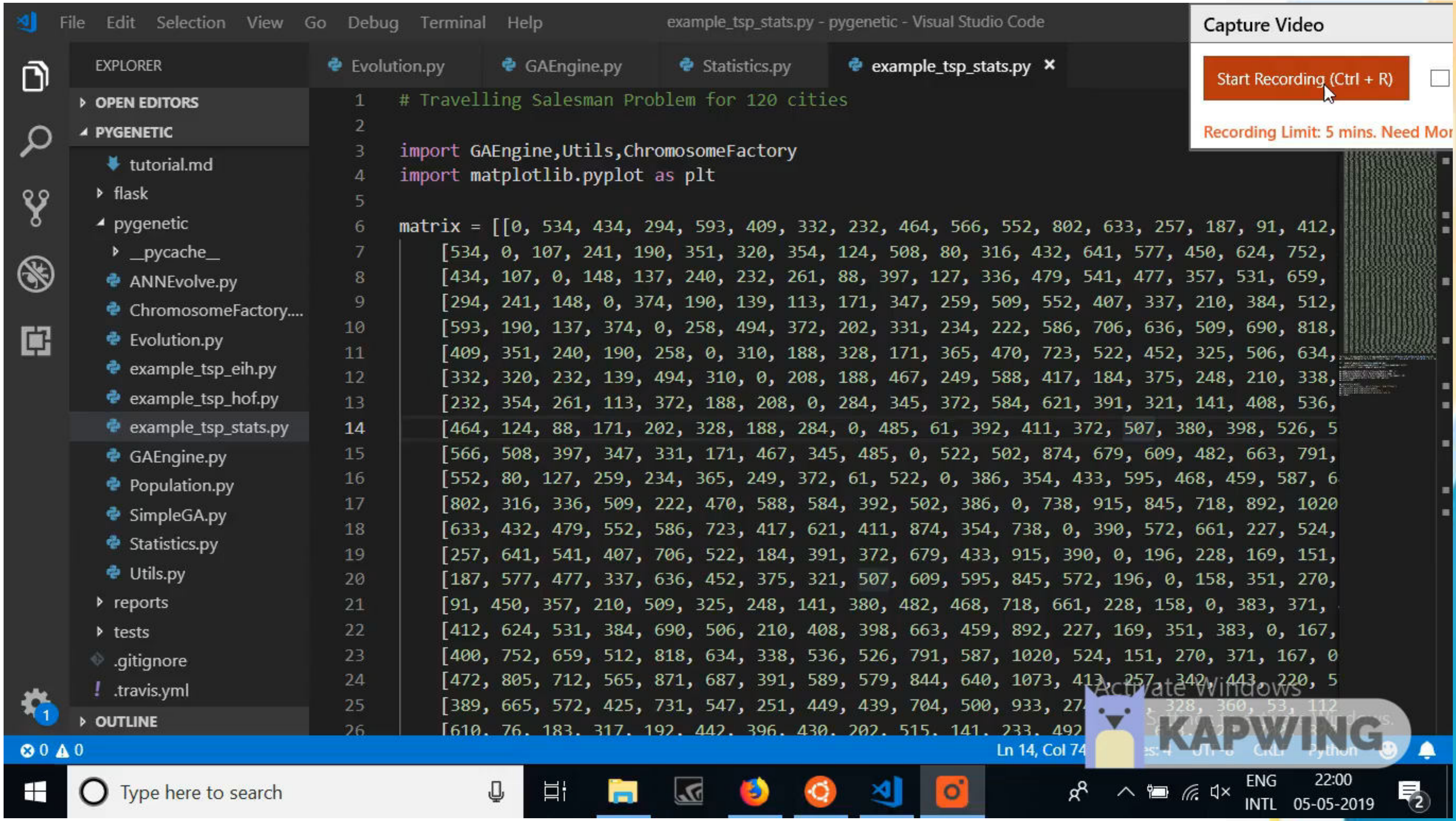
```python
# Travelling Salesman Problem for 8 cities

import GAEngine,Utils,ChromosomeFactory

matrix = [[0,172,145,607,329,72,312,120],
    [172,0,192,494,209,158,216,92],
    [145,192,0,490,237,75,205,100],
    [607,494,490,0,286,545,296,489],
    [329,209,237,286,0,421,49,208],
    [72,158,75,545,421,0,249,75],
    [312,216,205,296,49,249,9,194],
    [120,92,100,489,208,75,194,0]]

factory = ChromosomeFactory.ChromosomeRangeFactory(noOfGenes=8,minValue=0,maxValue=7)
ga = GAEngine.GAEngine(factory,1000,fitness_type='min', cross_prob = 0.7, mut_prob = 0.2, ef
ga.addCrossoverHandler(Utils.CrossoverHandlers.PMX, 1)
ga.addMutationHandler(Utils.MutationHandlers.swap, 1)
ga.setSelectionHandler(Utils.SelectionHandlers.best)
ga.setFitnessHandler(Utils.Fitness.TSP, matrix)
ga.evolve(100)
```

6. Supports Efficient Iteration Halt

- Iterations can be made to stop if best fitness repeats continuously for many generations
- Hence, processing power can be saved

PES UNIVERSITY

7. Supports Visualization of Statistics / Custom Statistics



```
# Travelling Salesman Problem for 120 cities

import GAEngine,Utils,ChromosomeFactory
import matplotlib.pyplot as plt

matrix = [[0, 534, 434, 294, 593, 409, 332, 232, 464, 566, 552, 802, 633, 257, 187, 91, 412,
    [534, 0, 107, 241, 190, 351, 320, 354, 124, 508, 80, 316, 432, 641, 577, 450, 624, 752,
    [434, 107, 0, 148, 137, 240, 232, 261, 88, 397, 127, 336, 479, 541, 477, 357, 531, 659,
    [294, 241, 148, 0, 374, 190, 139, 113, 171, 347, 259, 509, 552, 407, 337, 210, 384, 512,
    [593, 190, 137, 374, 0, 258, 494, 372, 202, 331, 234, 222, 586, 706, 636, 509, 690, 818,
    [409, 351, 240, 190, 258, 0, 310, 188, 328, 171, 365, 470, 723, 522, 452, 325, 506, 634,
    [332, 320, 232, 139, 494, 310, 0, 208, 188, 467, 249, 588, 417, 184, 375, 248, 210, 338,
    [232, 354, 261, 113, 372, 188, 208, 0, 284, 345, 372, 584, 621, 391, 321, 141, 408, 536,
    [464, 124, 88, 171, 202, 328, 188, 284, 0, 485, 61, 392, 411, 372, 507, 380, 398, 526, 5
    [566, 508, 397, 347, 331, 171, 467, 345, 485, 0, 522, 502, 874, 679, 609, 482, 663, 791,
    [552, 80, 127, 259, 234, 365, 249, 372, 61, 522, 0, 386, 354, 433, 595, 468, 459, 587, 6
    [802, 316, 336, 509, 222, 470, 588, 584, 392, 502, 386, 0, 738, 915, 845, 718, 892, 1020
    [633, 432, 479, 552, 586, 723, 417, 621, 411, 874, 354, 738, 0, 390, 572, 661, 227, 524,
    [257, 641, 541, 407, 706, 522, 184, 391, 372, 679, 433, 915, 390, 0, 196, 228, 169, 151,
    [187, 577, 477, 337, 636, 452, 375, 321, 507, 609, 595, 845, 572, 196, 0, 158, 351, 270,
    [91, 450, 357, 210, 509, 325, 248, 141, 380, 482, 468, 718, 661, 228, 158, 0, 383, 371,
    [412, 624, 531, 384, 690, 506, 210, 408, 398, 663, 459, 892, 227, 169, 351, 383, 0, 167,
    [400, 752, 659, 512, 818, 634, 338, 536, 526, 791, 587, 1020, 524, 151, 270, 371, 167, 0
    [472, 805, 712, 565, 871, 687, 391, 589, 579, 844, 640, 1073, 413, 257, 342, 443, 220, 5
    [389, 665, 572, 425, 731, 547, 251, 449, 439, 704, 500, 933, 27
    [610, 76, 183, 317, 192, 442, 396, 430, 202, 515, 141, 233, 492
```

7. Supports Visualization of Statistics / Custom Statistics

- Default Statistics: best-fitness,worst-fitness,avg-fitness,diversity, mutation_rate

- Custom Statistics:

```
ga = ...
...
def range_of_generation(fitness_mappings,ga):
        return abs(fitness_mappings[0][1] - fitness_mappings[-1][1])

ga.addStatistic('range',range_of_generation)
```

- Note: Very generic function. Various types of statistics can be defined.

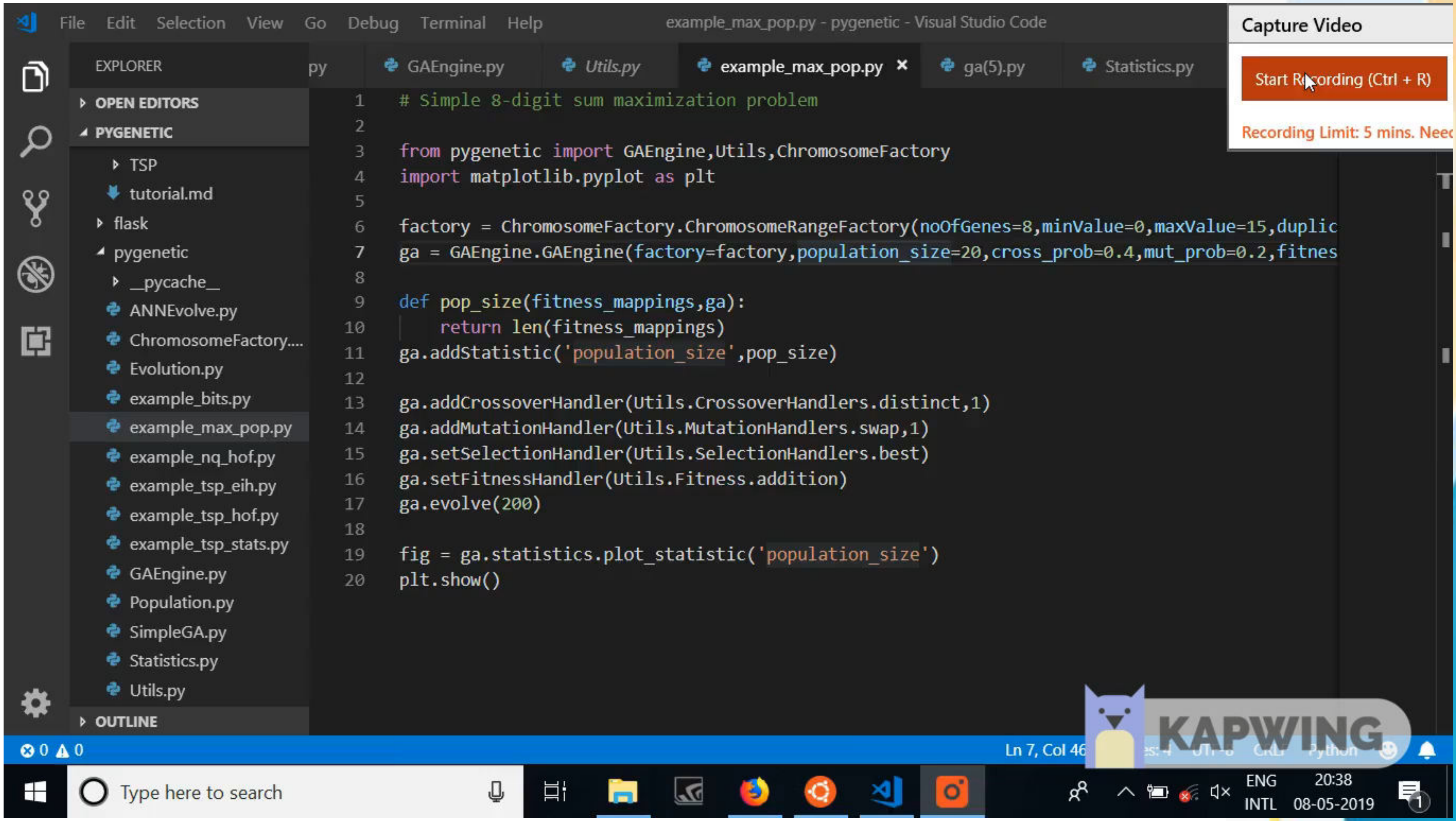
8. Supports multiple crossovers/mutations in one GA execution

- `pygenetic` supports more than one crossovers and mutations in one GA execution.

```
from pygenetic import Utils
ga.addCrossoverHandler(Utils.CrossoverHandlers.distinct, 4)
ga.addCrossoverHandler(Utils.CrossoverHandlers.OX, 3)
ga.addMutationHandler(Utils.MutationHandlers.swap,2)
ga.addMutationHandler(Utils.MutationHandlers.bitFlip,2)
```

- Having more than one crossover/mutation improves the diversity of the population.

9. Supports Population Control



```python
# Simple 8-digit sum maximization problem

from pygenetic import GAEngine,Utils,ChromosomeFactory
import matplotlib.pyplot as plt

factory = ChromosomeFactory.ChromosomeRangeFactory(noOfGenes=8,minValue=0,maxValue=15,duplic
ga = GAEngine.GAEngine(factory=factory,population_size=20,cross_prob=0.4,mut_prob=0.2,fitnes

def pop_size(fitness_mappings,ga):
    return len(fitness_mappings)
ga.addStatistic('population_size',pop_size)

ga.addCrossoverHandler(Utils.CrossoverHandlers.distinct,1)
ga.addMutationHandler(Utils.MutationHandlers.swap,1)
ga.setSelectionHandler(Utils.SelectionHandlers.best)
ga.setFitnessHandler(Utils.Fitness.addition)
ga.evolve(200)

fig = ga.statistics.plot_statistic('population_size')
plt.show()
```

9. Supports Population Control

- Since end users may define custom handlers, custom evolutions, etc, the number of chromosomes in a population can be made to go beyond/below the population size.
- Enabling this option ensures that same population size is maintained.
- Disabling it can be done to allow population size to vary. (eg: for some research purpose applications)

10. Supports bunch of standard Crossover, Mutation, Selection handlers

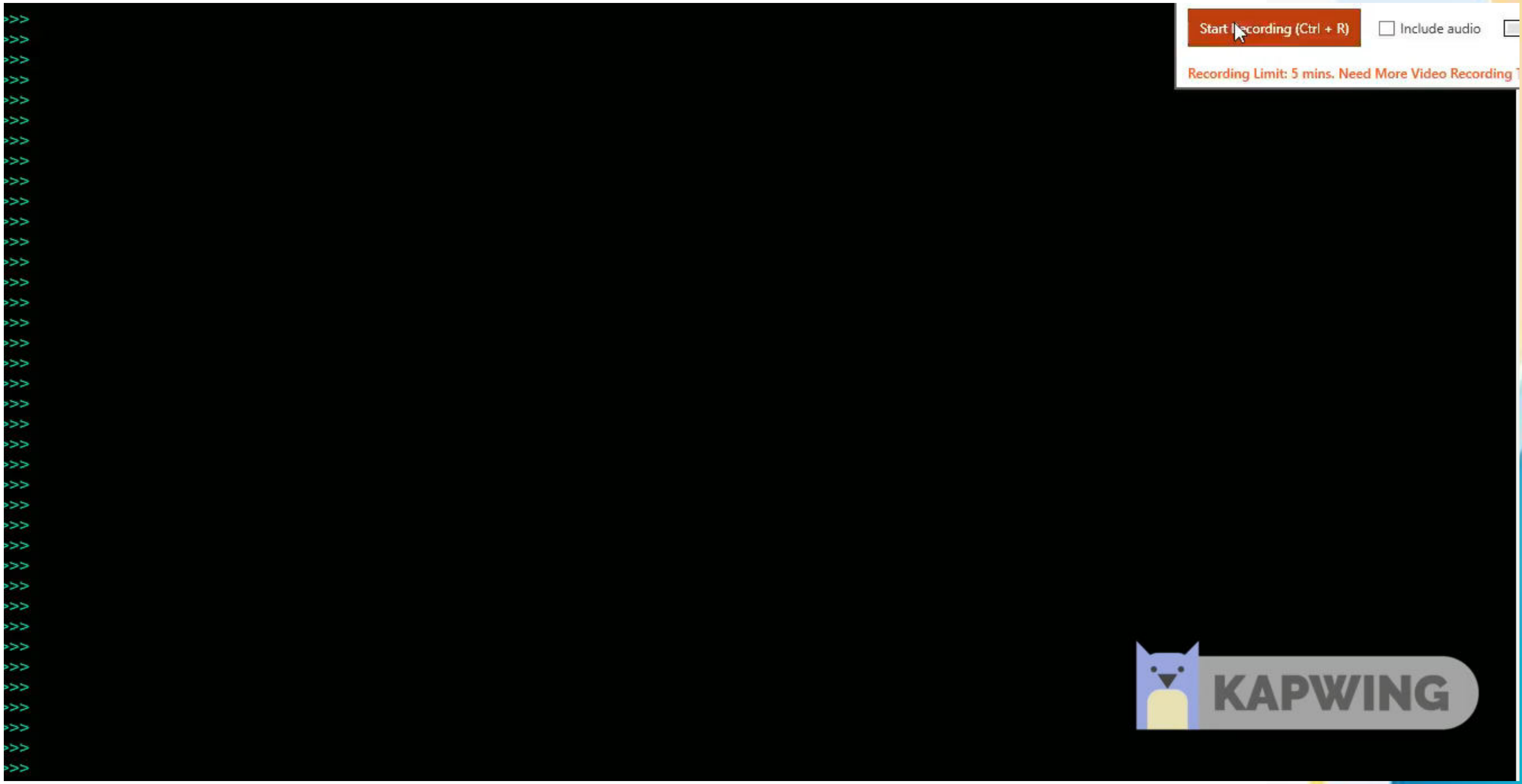
- Many standard Crossovers, Mutations and Selection Functions are already available in Utils of pygenetic module.

➢ Selection - random, best, tournament, roulette, rank and SUS

➢ Crossover - distinct, onePoint, twoPoint, PMX and OX

➢ Mutation - swap and bitFlip

11. Provides continue evolution feature



```
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

Start Recording (Ctrl + R)
Include audio
Recording Limit: 5 mins. Need More Video Recording
KAPWING

11. Provides continue evolution feature

- Often users may evolve a GA for some generations and then realise if they had continued for a few more generations, the problem would have been solved

- Pygenetic supports continuing from previous evolutions (No need to start evolutions again!)

```
ga = ...
...
ga.evolve(100)
ga.continue_evolve(20)
```

12. Provides ANN Best Topology Finder Feature

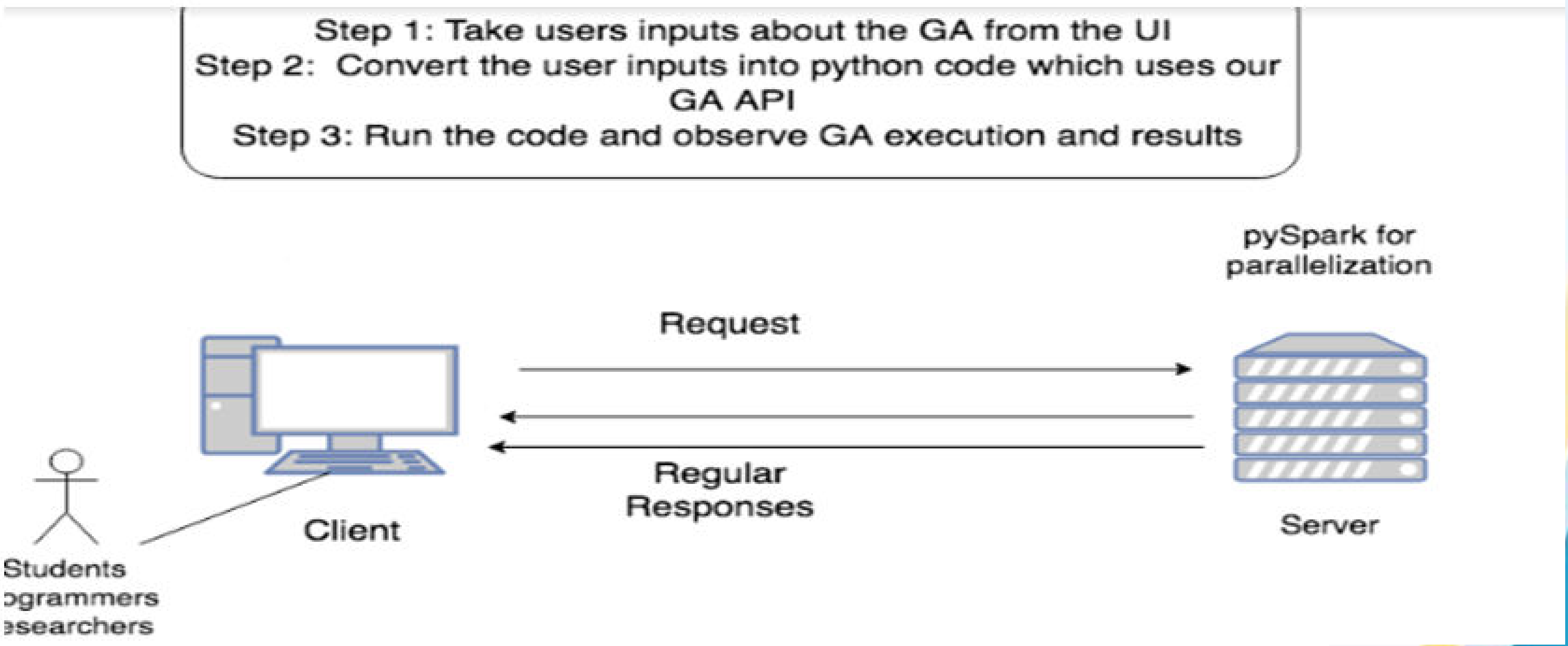
- Users can find best ANN Topology to use for training to solve a classification problem. Done using GA.

```
from pygenetic import ANNEvolve
...
X = dataset[:,0:8]
Y = dataset[:,8]
a = ANNEvolve.ANNTopologyEvolve(X,Y,hiddenLayers=2,population_size=100,
                                neuronsPerLayer=[2,5,10,12],
                                activations=['relu','sigmoid'],
                                optimizers=['adam'],
                                loss='binary_crossentropy',
                                metrics='accuracy',
                                epochs=30,batch_size=10)
a.evolve(100)
print(a.best_fitness)
```

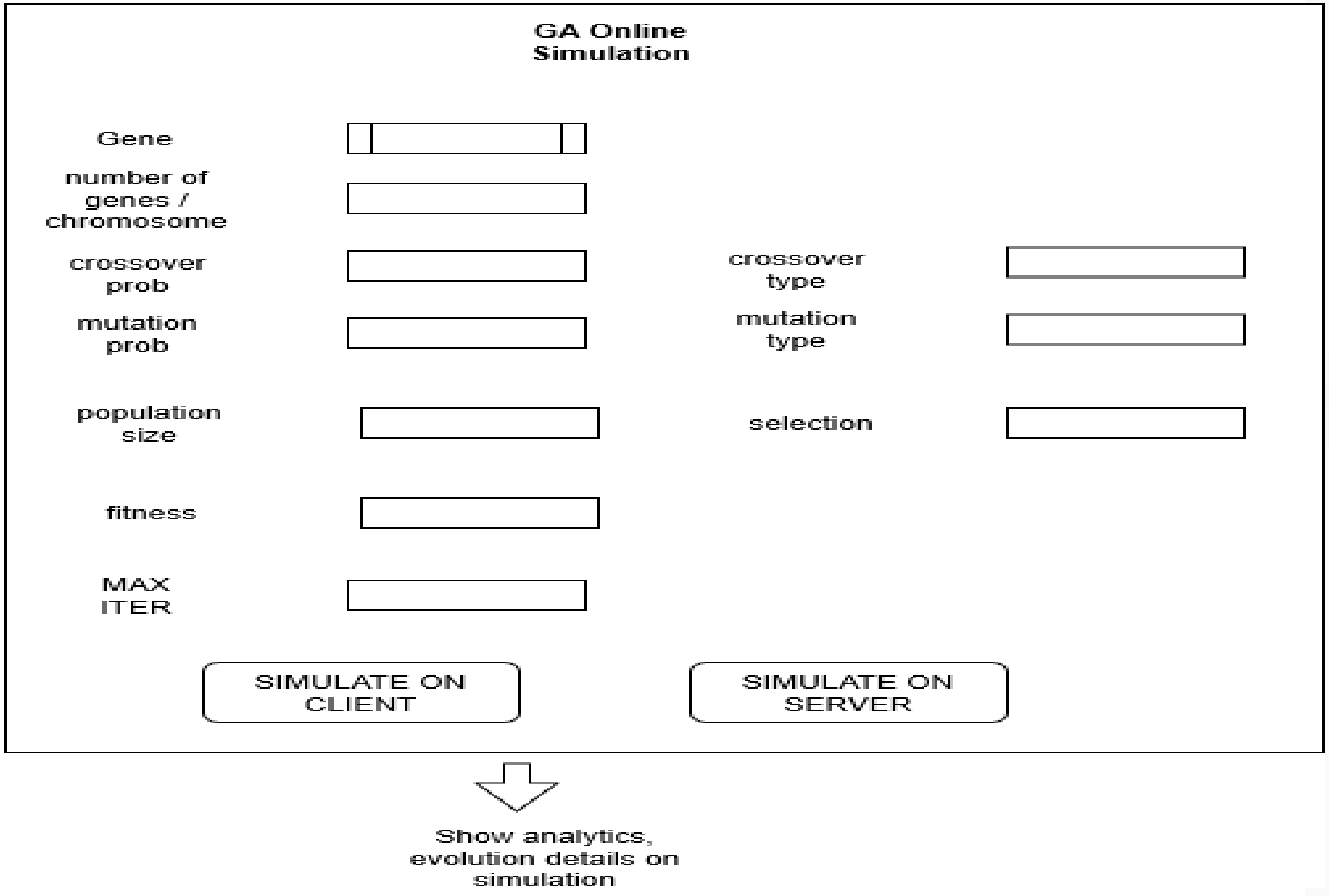
Solving GAs using pygenetic Code Walkthrough

GA Online Execution Website

System Architecture


Step 1: Take users inputs about the GA from the UI
Step 2: Convert the user inputs into python code which uses our GA API
Step 3: Run the code and observe GA execution and results
pySpark for parallelization
Request
Regular Responses
Client
Server
Students
ogrammers
esearchers

UI/ Use Case


GA Online Simulation
Gene
number of genes / chromosome
crossover prob
mutation prob
population size
fitness
MAX ITER
crossover type
mutation type
selection
SIMULATE ON CLIENT
SIMULATE ON SERVER
Show analytics, evolution details on simulation

GA Online Execution Website: DEMO

Testing

Emphasis was laid on testing the functionalities of various classes of the GA.

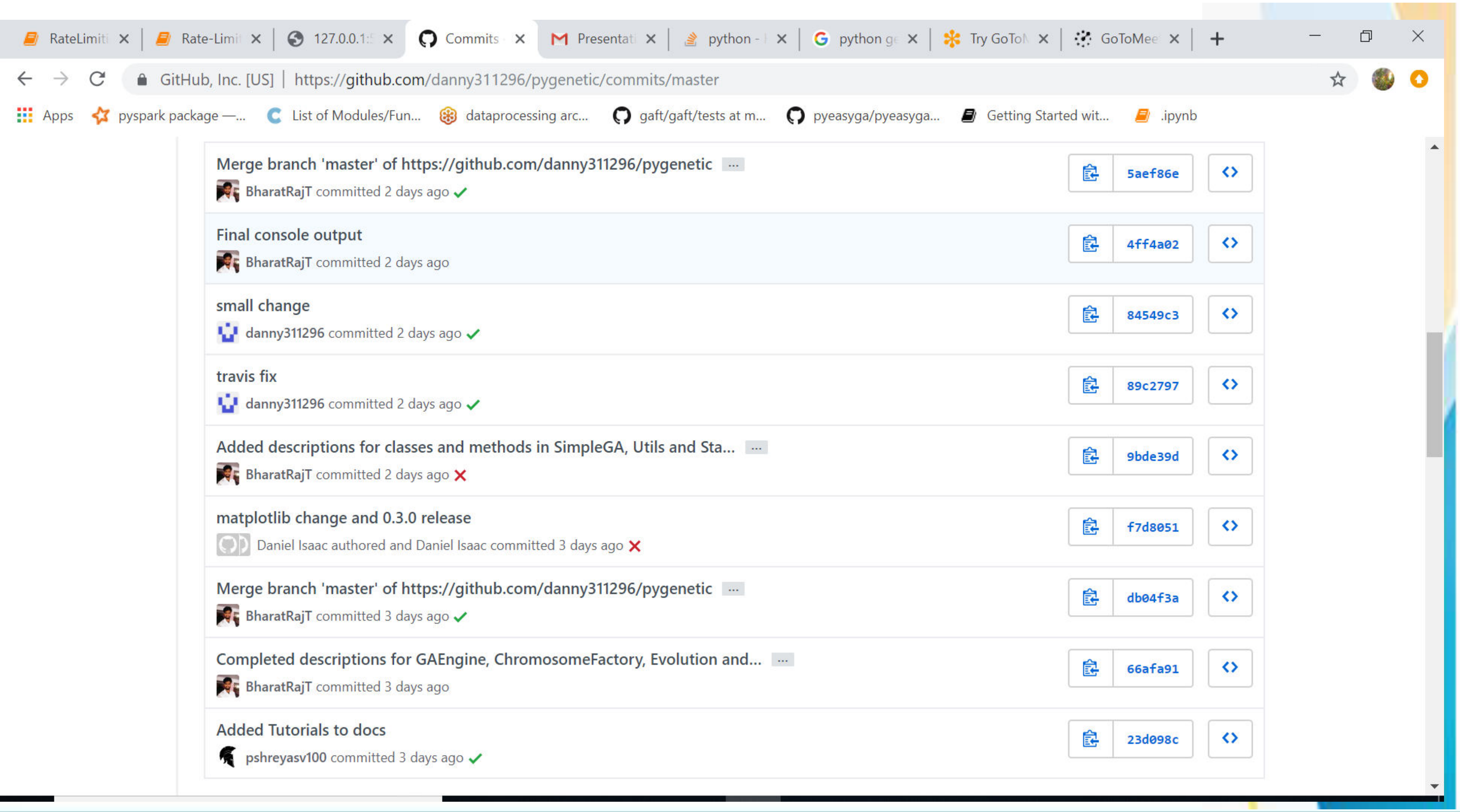
Main API Testing Approaches

1. White Box Testing
2. Black Box Testing
3. Error Handling Testing

Tools

1. Pytest
2. Pylint
3. Unitest.mock

Testing: TravisCI



https://github.com/danny311296/pygenetic/commits/master

Merge branch 'master' of https://github.com/danny311296/pygenetic ...
BharatRajT committed 2 days ago ✓ — 5aef86e

Final console output
BharatRajT committed 2 days ago — 4ff4a02

small change
danny311296 committed 2 days ago ✓ — 84549c3

travis fix
danny311296 committed 2 days ago ✓ — 89c2797

Added descriptions for classes and methods in SimpleGA, Utils and Sta... ...
BharatRajT committed 2 days ago ✗ — 9bde39d

matplotlib change and 0.3.0 release
Daniel Isaac authored and Daniel Isaac committed 3 days ago ✗ — f7d8051

Merge branch 'master' of https://github.com/danny311296/pygenetic ...
BharatRajT committed 3 days ago ✓ — db04f3a

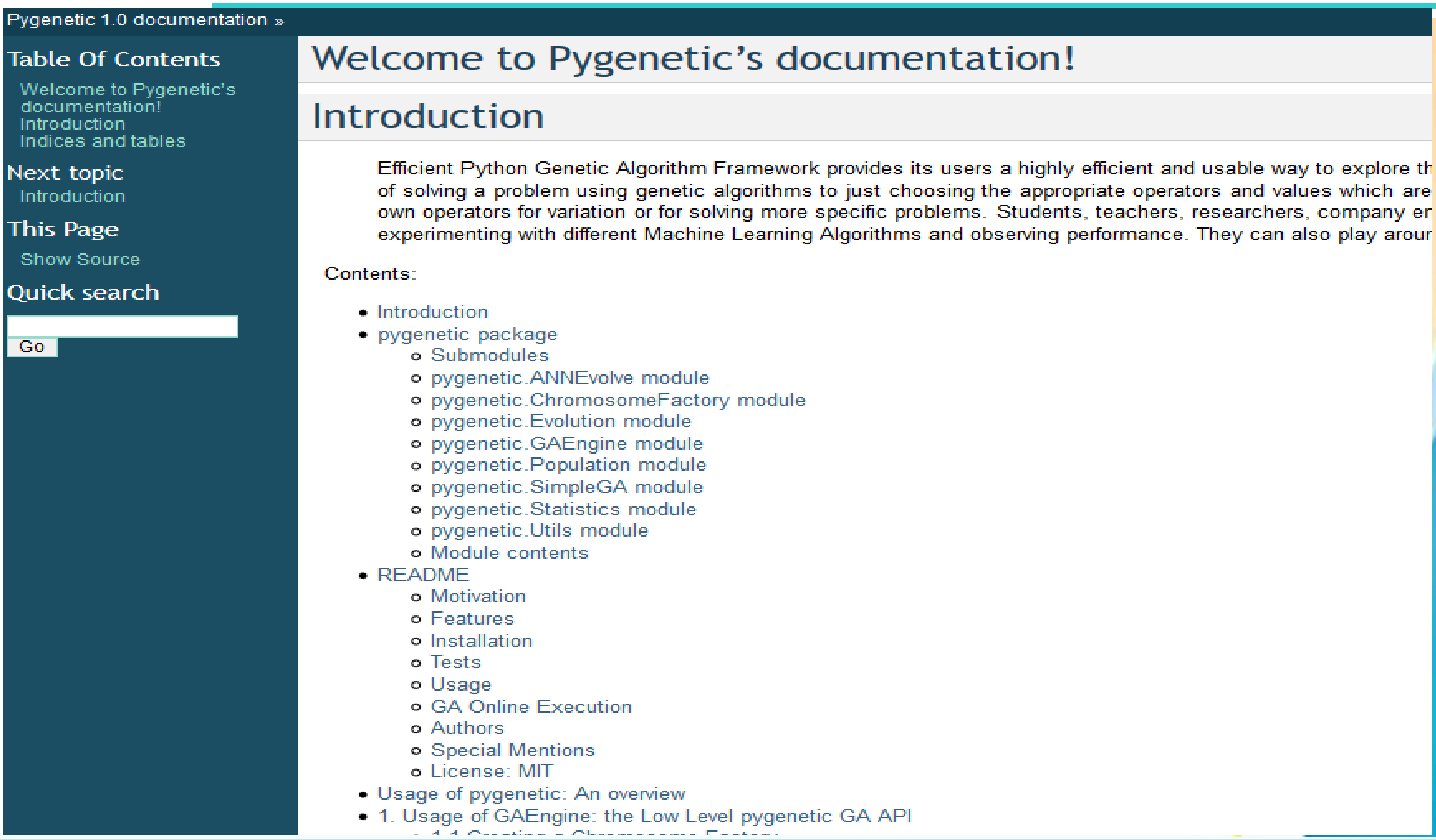
Completed descriptions for GAEngine, ChromosomeFactory, Evolution and... ...
BharatRajT committed 3 days ago — 66afa91

Added Tutorials to docs
pshreyasv100 committed 3 days ago ✓ — 23d098c

Documentation

- No API is complete without proper code documentation and tutorials for API end users
- Our API has been intensively documented and hosted on ReadTheDocs
- It has various tutorials to help new users to get a quick grasp about the usage of the pygenetic.

Documentation on ReadTheDocs



Pygenetic 1.0 documentation »

Table Of Contents

Next topic

Introduction

This Page

Show Source

Quick search

Go

Welcome to Pygenetic's documentation!

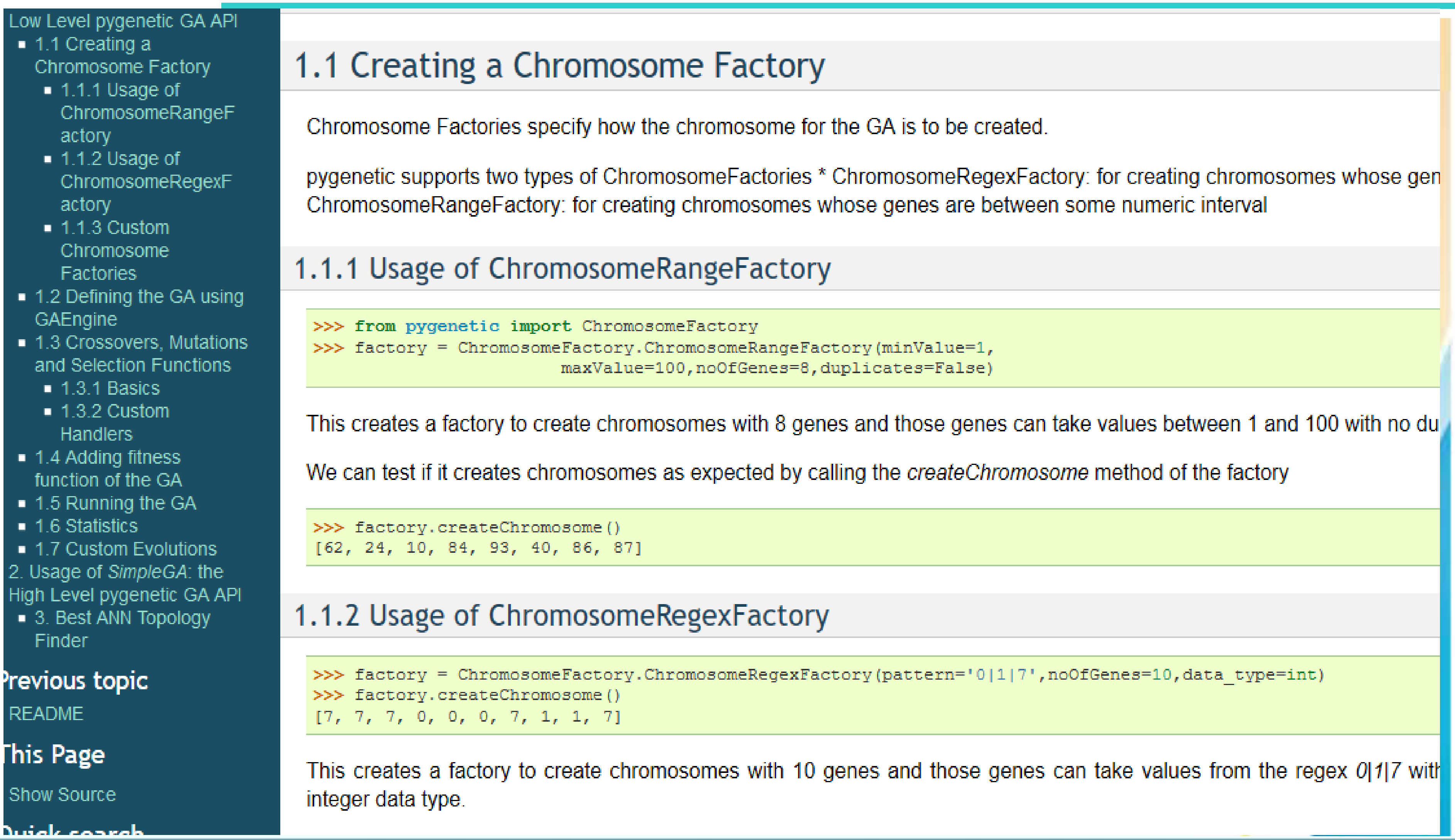
Introduction

Efficient Python Genetic Algorithm Framework provides its users a highly efficient and usable way to explore th
of solving a problem using genetic algorithms to just choosing the appropriate operators and values which are
own operators for variation or for solving more specific problems. Students, teachers, researchers, company er
experimenting with different Machine Learning Algorithms and observing performance. They can also play arour

Contents:

PES UNIVERSITY

Documentation on ReadTheDocs



revious topic

README

his Page

Show Source

1.1 Creating a Chromosome Factory

Chromosome Factories specify how the chromosome for the GA is to be created.

pygenetic supports two types of ChromosomeFactories * ChromosomeRegexFactory: for creating chromosomes whose gen
ChromosomeRangeFactory: for creating chromosomes whose genes are between some numeric interval

1.1.1 Usage of ChromosomeRangeFactory

```
>>> from pygenetic import ChromosomeFactory
>>> factory = ChromosomeFactory.ChromosomeRangeFactory(minValue=1,
                    maxValue=100,noOfGenes=8,duplicates=False)
```

This creates a factory to create chromosomes with 8 genes and those genes can take values between 1 and 100 with no du

We can test if it creates chromosomes as expected by calling the *createChromosome* method of the factory

```
>>> factory.createChromosome()
[62, 24, 10, 84, 93, 40, 86, 87]
```

1.1.2 Usage of ChromosomeRegexFactory

```
>>> factory = ChromosomeFactory.ChromosomeRegexFactory(pattern='0|1|7',noOfGenes=10,data_type=int)
>>> factory.createChromosome()
[7, 7, 7, 0, 0, 0, 7, 1, 1, 7]
```

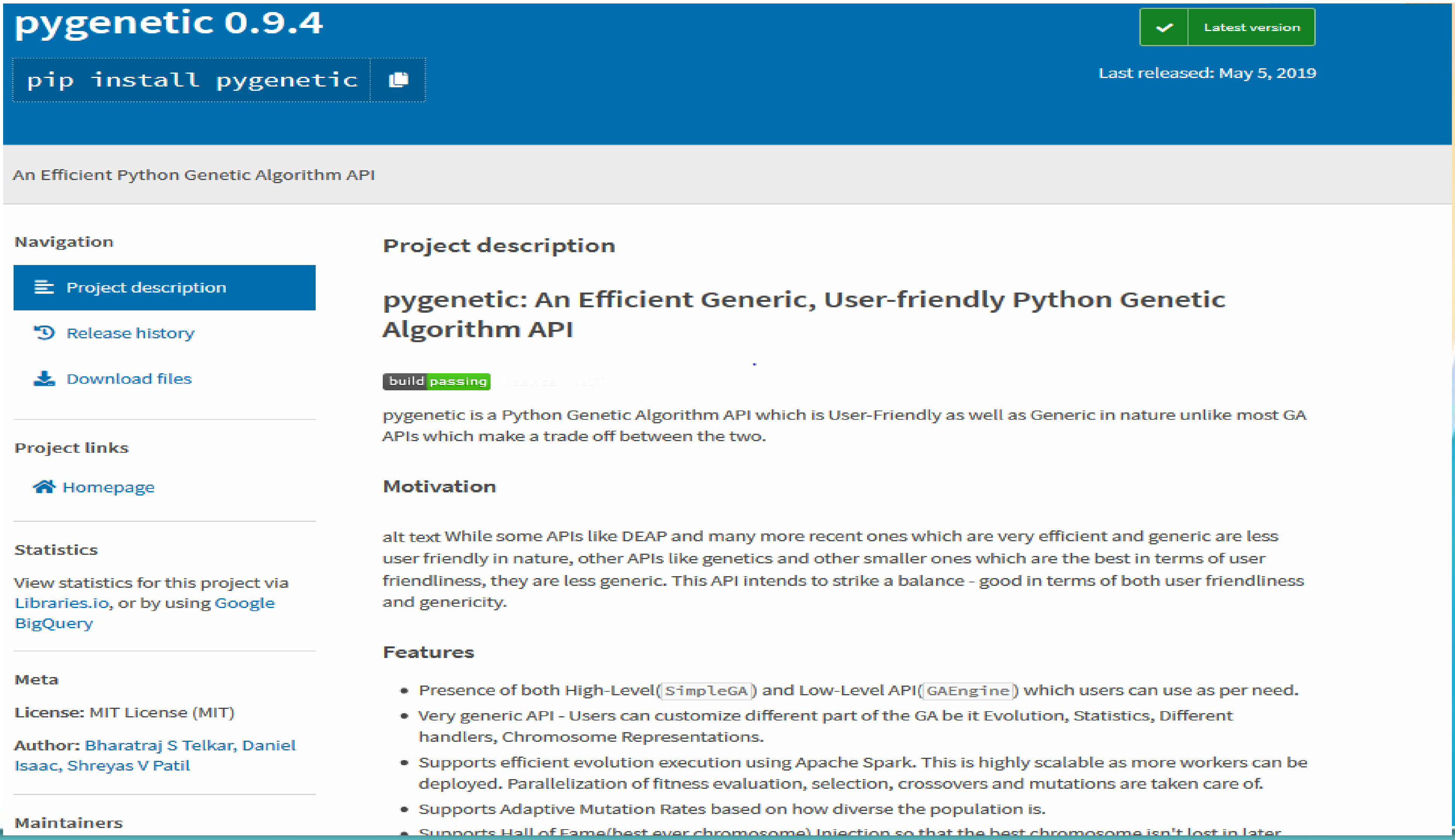
This creates a factory to create chromosomes with 10 genes and those genes can take values from the regex *0|1|7* wit
integer data type.

Publishing On PyPI

- Python Packaging Index is a place where most of the standard library projects as well as open source python projects are hosted.
- Packages published on PyPI as distributed as distribution packages, source archives and python wheels are popular formats.
- PyPI allows users to search for packages by keywords or by filters against their metadata.
- Packages can then be installed by other users using pip install

```
$pip install pygenetic
```

Publishing on PyPI



pygenetic 0.9.4

```
pip install pygenetic
```

Latest version

Last released: May 5, 2019

An Efficient Python Genetic Algorithm API

Navigation

- Project description
- Release history
- Download files

Project links

- Homepage

Statistics

View statistics for this project via Libraries.io, or by using Google BigQuery

Meta

License: MIT License (MIT)

Author: Bharatraj S Telkar, Daniel Isaac, Shreyas V Patil

Maintainers

Project description

pygenetic: An Efficient Generic, User-friendly Python Genetic Algorithm API

build passing

pygenetic is a Python Genetic Algorithm API which is User-Friendly as well as Generic in nature unlike most GA APIs which make a trade off between the two.

Motivation

alt text While some APIs like DEAP and many more recent ones which are very efficient and generic are less user friendly in nature, other APIs like genetics and other smaller ones which are the best in terms of user friendliness, they are less generic. This API intends to strike a balance - good in terms of both user friendliness and genericity.

Features

- Presence of both High-Level(`SimpleGA`) and Low-Level API(`GAEngine`) which users can use as per need.
- Very generic API - Users can customize different part of the GA be it Evolution, Statistics, Different handlers, Chromosome Representations.
- Supports efficient evolution execution using Apache Spark. This is highly scalable as more workers can be deployed. Parallelization of fitness evaluation, selection, crossovers and mutations are taken care of.
- Supports Adaptive Mutation Rates based on how diverse the population is.
- Supports Hall of Fame(best ever chromosome) Injection so that the best chromosome isn't lost in later

Project Results: Performance

Performance comparison of Pygenetic

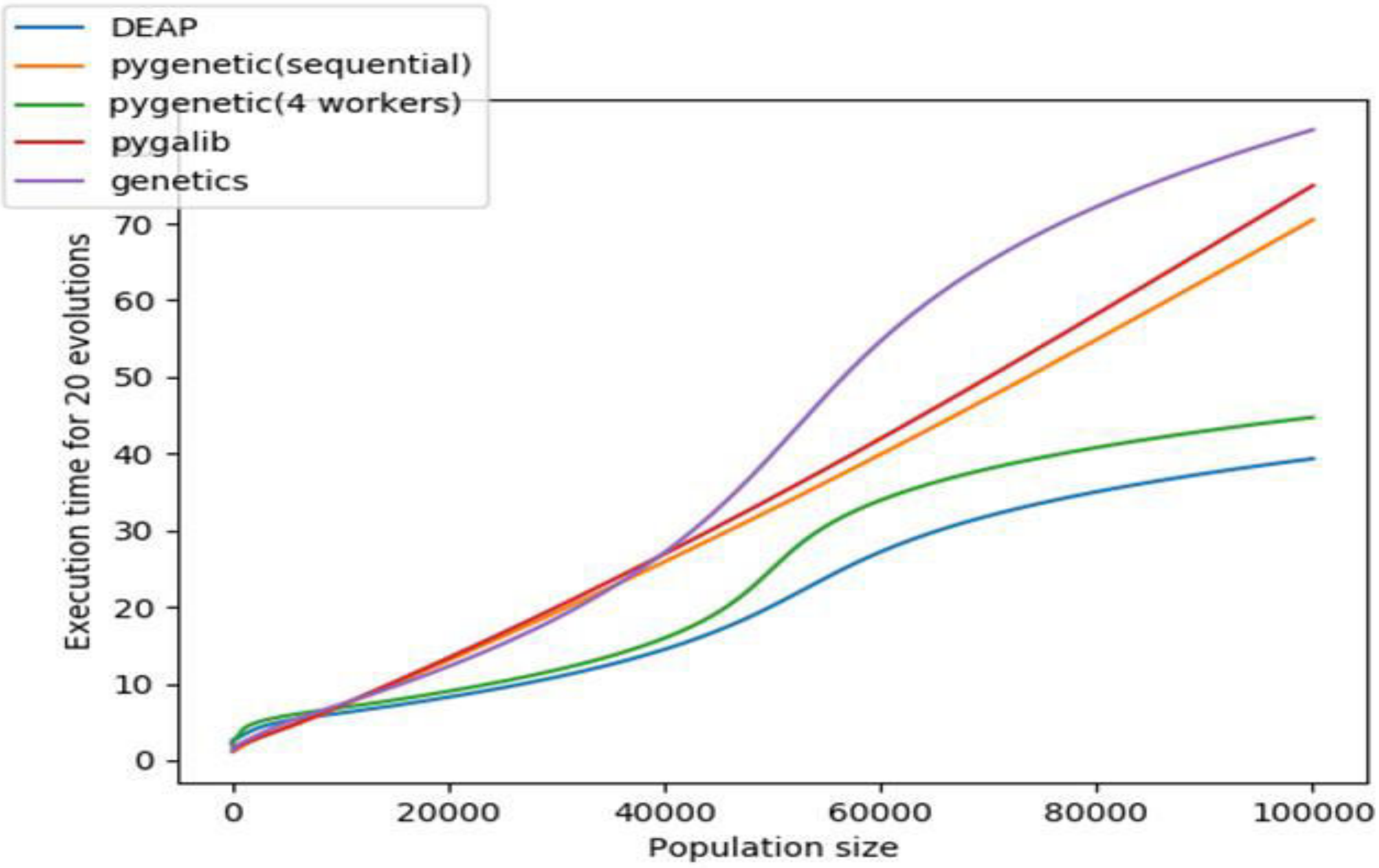
The time comparisons of pygenetic vs DEAP vs other python GA APIs were monitored for execution of a TSP GA of different population sizes and their execution times were examined. It was run for 20 evolutions.

Project Results

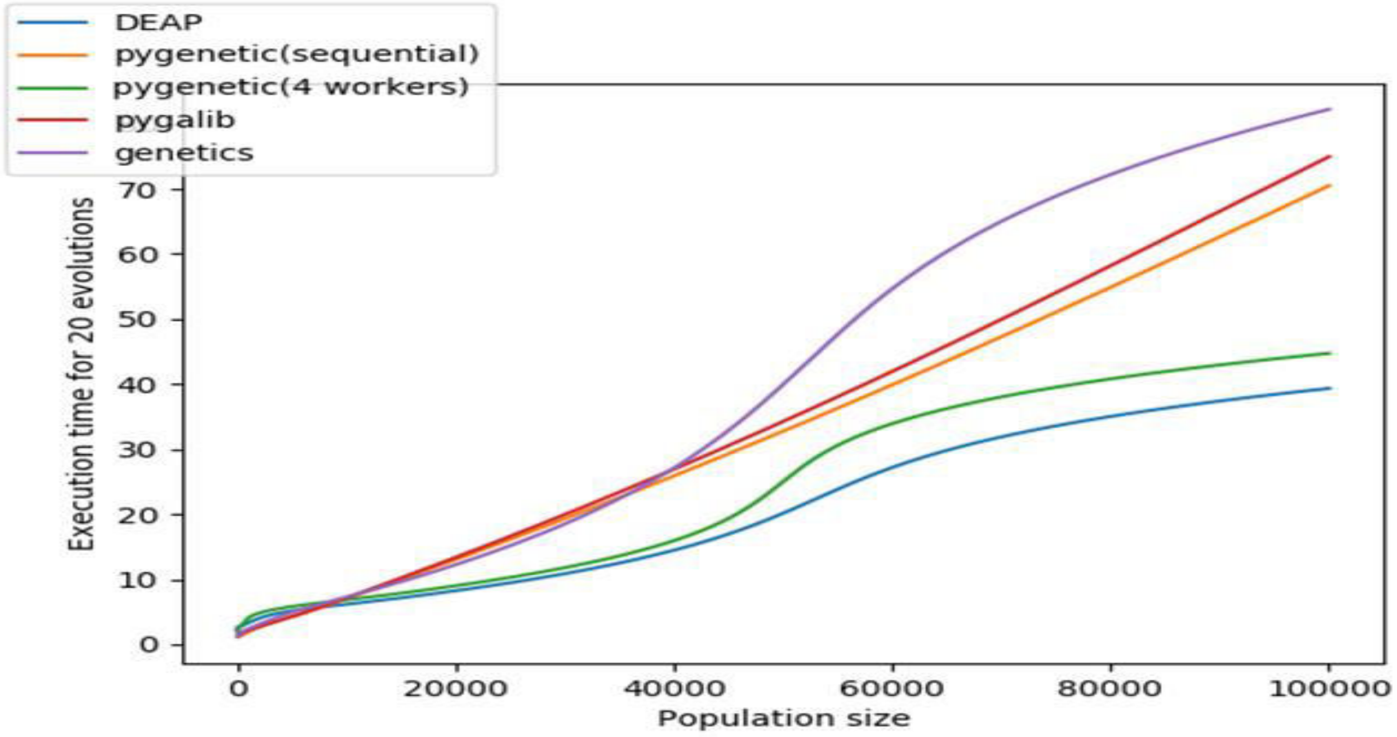
Population size	DEAP	pygenetic	pygenetic(2workers)	genetics	pygalib
100	1.923	1.093	2.620	1.289	1.278
500	2.879	1.437	3.027	1.983	1.762
1000	3.259	1.922	3.996	2.345	2.093
5000	5.129	4.201	5.798	4.892	4.231
10000	6.142	6.987	6.823	7.234	7.102
50000	20.123	32.738	24.982	39.902	34.234
100000	39.332	70.487	44.729	82.234	74.956

PES UNIVERSITY

Project Results



Project Results



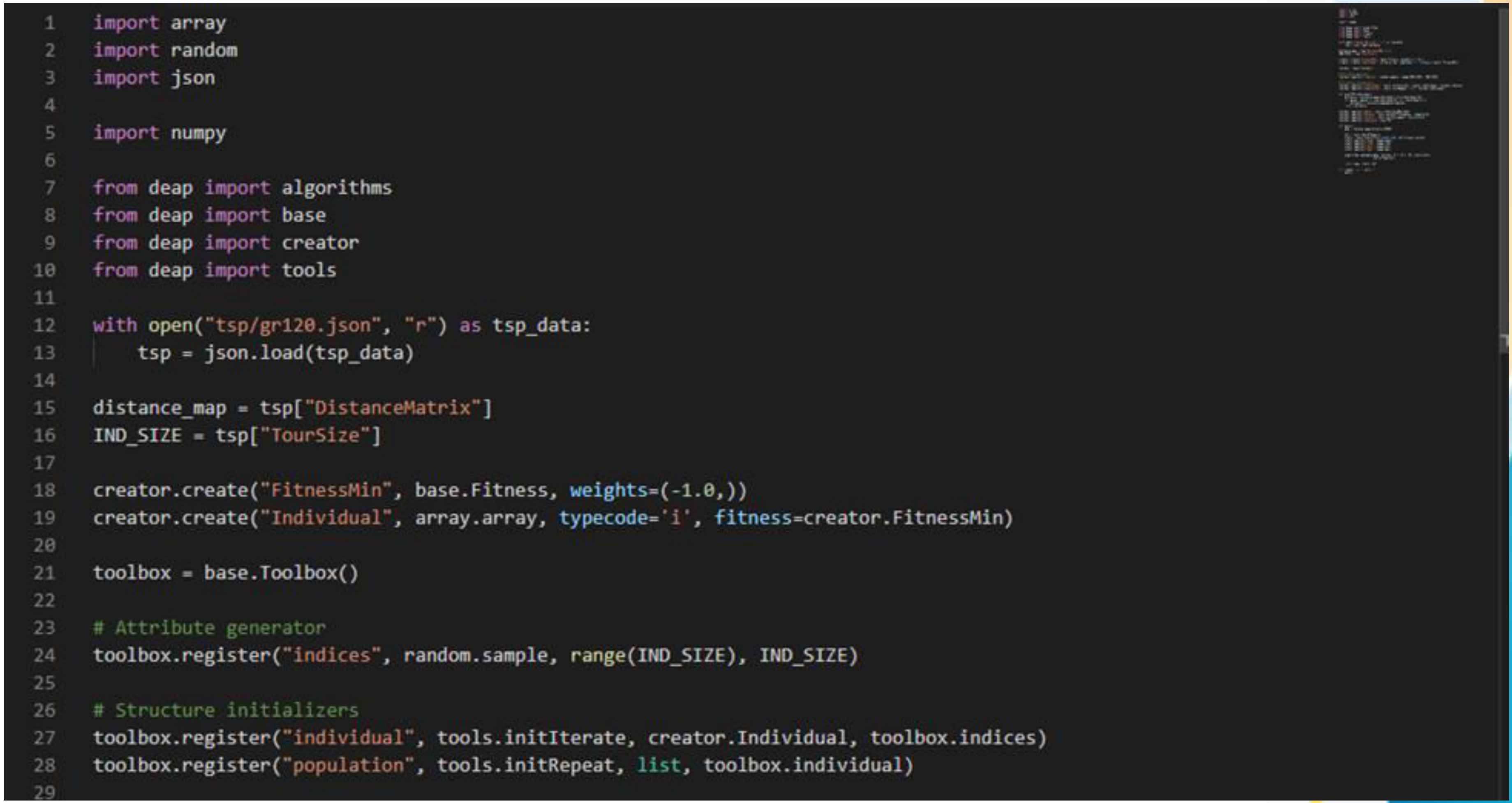
- pygenetic performs better than the other sequential GA APIs genetics and pygalib
- pygenetic is scalable when deployed using more workers, efficiency approaches that of DEAP.

Project Results: User Friendliness

Let’s compare pygenetic with DEAP over the critical aspect of user friendliness.

Here is an example of DEAP that solves a Travelling Salesman Problem (data in json)

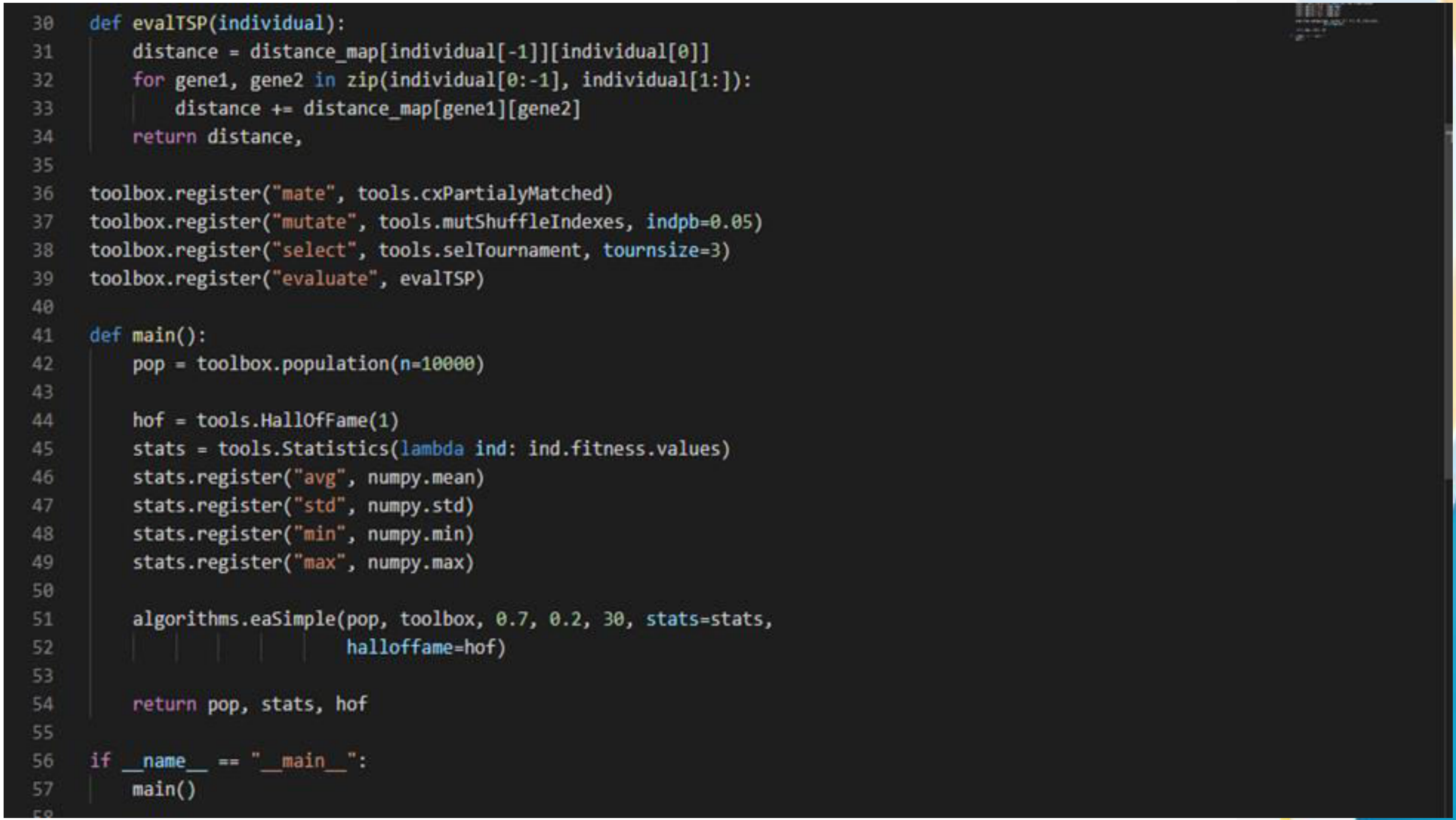
Project Results



```python
import array
import random
import json

import numpy

from deap import algorithms
from deap import base
from deap import creator
from deap import tools

with open("tsp/gr120.json", "r") as tsp_data:
    tsp = json.load(tsp_data)

distance_map = tsp["DistanceMatrix"]
IND_SIZE = tsp["TourSize"]

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", array.array, typecode='i', fitness=creator.FitnessMin)

toolbox = base.Toolbox()

# Attribute generator
toolbox.register("indices", random.sample, range(IND_SIZE), IND_SIZE)

# Structure initializers
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.indices)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

continued..

Project Results

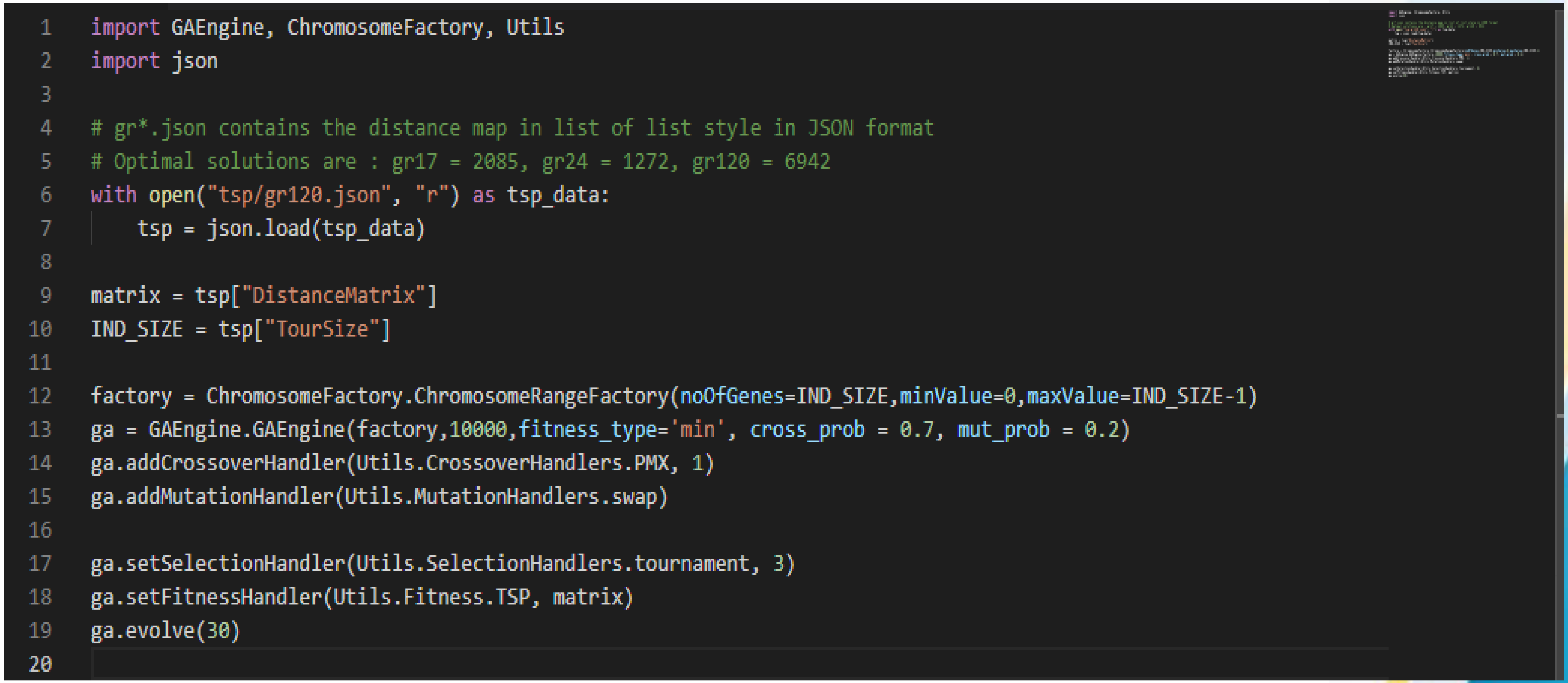


```
def evalTSP(individual):
    distance = distance_map[individual[-1]][individual[0]]
    for gene1, gene2 in zip(individual[0:-1], individual[1:]):
        distance += distance_map[gene1][gene2]
    return distance,

toolbox.register("mate", tools.cxPartialyMatched)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evalTSP)

def main():
    pop = toolbox.population(n=10000)

    hof = tools.HallOfFame(1)
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", numpy.mean)
    stats.register("std", numpy.std)
    stats.register("min", numpy.min)
    stats.register("max", numpy.max)

    algorithms.eaSimple(pop, toolbox, 0.7, 0.2, 30, stats=stats,
                        halloffame=hof)

    return pop, stats, hof

if __name__ == "__main__":
    main()
```

Project Results

- There are a lot of instantiations and classes to remember.
- Difficulty in creating custom chromosomes
- Difficulty in adding statistics
- Too many complex parameters to keep note of. Although intended to make the API generic, This makes it very difficult for users to make more generic operations.

Project Results


```python
import GAEngine, ChromosomeFactory, Utils
import json

# gr*.json contains the distance map in list of list style in JSON format
# Optimal solutions are : gr17 = 2085, gr24 = 1272, gr120 = 6942
with open("tsp/gr120.json", "r") as tsp_data:
    tsp = json.load(tsp_data)

matrix = tsp["DistanceMatrix"]
IND_SIZE = tsp["TourSize"]

factory = ChromosomeFactory.ChromosomeRangeFactory(noOfGenes=IND_SIZE,minValue=0,maxValue=IND_SIZE-1)
ga = GAEngine.GAEngine(factory,10000,fitness_type='min', cross_prob = 0.7, mut_prob = 0.2)
ga.addCrossoverHandler(Utils.CrossoverHandlers.PMX, 1)
ga.addMutationHandler(Utils.MutationHandlers.swap)

ga.setSelectionHandler(Utils.SelectionHandlers.tournament, 3)
ga.setFitnessHandler(Utils.Fitness.TSP, matrix)
ga.evolve(30)

```

Project Results

- Just few lines of pure function-calling code (you need to know the parameters though) and you are done. You have solved a problem using genetic algorithms.
- Less classes to remember
- Simple parameters
- Generic + User Friendly

Project Results: Overview

1. Succeeded in making a “user friendly” + “generic” GA python API.
2. Sequential performance is better than other python sequential GA APIs.
3. Parallel execution mode is very scalable and comparable to DEAP.
4. Our Website successfully implements Online GA Execution using long polling.

Further Enhancements

- Support for 2 dimensional chromosomes and 2 dimensional chromosome based operations.
- More functions can be added to Utils.
- More efficient parallelization can be implemented in many other types of selection handlers.
- Also more optimizations like long and short term memory can be implemented in newer versions.

Planned Effort Vs Actual Effort

Activity	Hours/Person Planned	Actual Hours/Person
Feasibility Study • Examining existing frameworks • Shortcomings of existing frameworks	20	20
Literature Survey • Papers on efficiency, parallelization and ML implementations	28	20
Requirement Specification	20	15
High and Low level Design	28	24
Coding and Implementation Python API coding Web application frontend and backend	90	102
Testing	40	55
API documentation	20	15
Report	10	5
Total Effort/Person	256	256

PES UNIVERSITY

Lessons Learnt

- Deep understanding on working of genetic algorithms.
- Understanding of approaching optimizations in various GA.
- Ability to model any state space search problem as a GA.
- Thinking ahead of users needs and making a generic API.
- Working experience on PySpark.
- Collaborating even when not being able to meet for a week.
- Learning about and using various testing frameworks provided.

Thank You !