Algoritmos Genéticos
Frameworks em Python

Danilo Sipoli Sanches

Departamento Acadêmico de Computação
Universidade Tecnológica Federal do Paraná
Cornélio Procópio

Framework EasyGA

https://github.com/danielwilczak101/EasyGA/wiki

Framework EasyGA - Código básico

https://github.com/danielwilczak101/EasyGA/blob/version1/README.md

```
pip3 install EasyGA
```

Getting started with EasyGA(Basic Example):

The goal of the basic example is to get all 5's in the chromosome.

```
import EasyGA

# Create the Genetic algorithm
ga = EasyGA.GA()

# Evolve the whole genetic algorithm until termination has been reached
ga.evolve()

# Print out the current generation and the population
ga.print_generation()
ga.print_population()
```

Framework EasyGA - Saída das execuções

https://github.com/danielwilczak101/EasyGA/blob/version1/README.md

Output:

```
Current Generation      : 15
Current population:
Chromosome - 0 [7][4][4][5][3][5][5][8][3][7] / Fitness = 3
Chromosome - 1 [7][4][4][5][3][5][5][8][3][7] / Fitness = 3
Chromosome - 2 [7][4][4][5][3][5][5][8][3][7] / Fitness = 3
Chromosome - 3 [7][4][4][5][3][5][5][8][3][7] / Fitness = 3
Chromosome - 4 [7][2][4][5][3][5][5][8][3][7] / Fitness = 3
Chromosome - 5 [7][2][4][5][3][5][5][8][3][7] / Fitness = 3
Chromosome - 6 [5][8][8][6][10][10][5][7][2][7] / Fitness = 2
Chromosome - 7 [5][8][8][6][10][10][5][7][2][7] / Fitness = 2
Chromosome - 8 [5][8][8][6][10][10][5][7][2][7] / Fitness = 2
Chromosome - 9 [7][2][8][10][3][5][5][8][1][7] / Fitness = 2
```

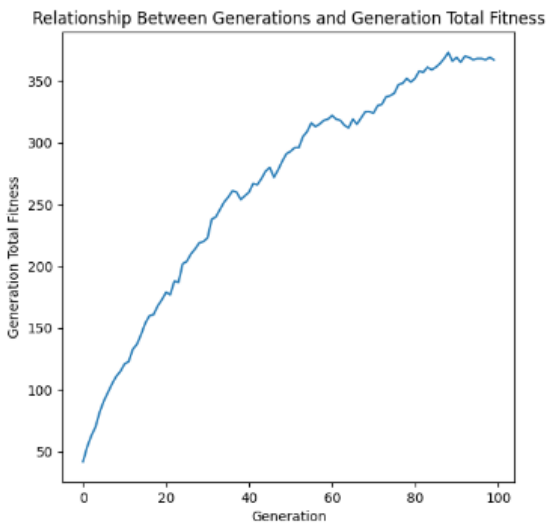
Framework EasyGA - Gráficos

https://github.com/danielwilczak101/EasyGA/wiki/Graph-Data

Graphing functions

Graphing is based on the **matplotlib** framework. The graph data is pulled using prebuilt database functions that can be found on the Store / Access page.

Graphing all generations total fitness:




```
import EasyGA

ga = EasyGA.GA()
ga.evolve()

ga.graph.generation_total_fitness()
ga.graph.show()
```

Graphing each generations total fitness. The standard EasyGA fitness function was used with 100 generations and a population size of 50.

Example of how total fitness is calculated

```
Current population:
Chromosome - 0 [5][5][5][5][5][5][5][5][5][5] / Fitness = 10
Chromosome - 1 [5][3][5][4][8][9][8][5][5][5] / Fitness = 5

# Current generation total fitness would be 15.
```

Framework DEAP

https://deap.readthedocs.io/en/master/

Introdução framework DEAP

Pacotes:

- base;
- creator;
- tools;
- algorithms.

Import:

- from deap import base;
- from deap import creator;
- from deap import tools;
- from deap import algorithms.

Introdução framework DEAP

Classe Creator:

- Necessário para criar indivíduos e função;
- creator.create("FitnessMax", base.Fitness, weights=(1.0,));
- creator.create("Individual", list, fitness=creator.FitnessMax).

Introdução framework DEAP

Classe base:

- Registra os elementos do algoritmo genético;
  - Tipo de dado dos indivíduos;
  - Formato da população;
- Individuos e população
  - Definir estrutura do individuo(list, set, etc)
  - Definir a função que irá gerar os alelos
  - Definir a função que irá gerar os individuos
  - Definir a função que irá gerar a popuulação

Classe base:

```
1  # Definindo a estrutura do individuo
2  creator.create("Individual", list, fitness=creator.
       FitnessMax)
3
4  # Gera o gene com alelos 0 ou 1 randomicamente uniforme
5  toolbox.register("attr_bool", random.randint, 0, 1)
6
7  # Gera o individuo (nome, Estrutura, funcao_geradora,
       tamanho)
8  toolbox.register("individual", tools.initRepeat, creator.
       Individual, toolbox.attr_bool, n=10)
9
10 # Funcao para gerar a populacao
11 toolbox.register("population", tools.initRepeat, list,
       toolbox.individual)
```

Listing: Classe base

Introdução framework DEAP

Classe tools:

- Permite utilizar os opradores genéticos;
- Operadores:
  - evaluate: operador para realizar o cálculo de fitness;
  - mate: operador para realizar cross over de indivíduos
  - mutate: operador para realizar a mutação dos indivíduos
  - select: operador para selecionar os melhores de uma geração para outra

Classe tools:

```
1     # registra funcao de fitness
2     toolbox.register("evaluate", evaluate)
3
4     # registra crossOver
5     toolbox.register("mate", tools.cxTwoPoint)
6
7     # registra mutacao com probabilidade de gene de 5%
8     toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
9
10    # registra o metodo de selecao como torneio de tamanho 3
11    toolbox.register("select", tools.selTournament,
      tournsize=3)
12
13
```

Listing: Classe tools

Classe Algorithms:

```
1     def main():
2       # cria populacao inicial
3       pop = toolbox.population(n=70)
4
5       # define as taxas
6       CXPB, MUTPB, NGEN =0.8, 0.02, 60
7
8       stats = tools.Statistics(key=lambda ind: ind.fitness.
    values)
9       stats.register("std", numpy.std)
10      stats.register("avg", numpy.mean)
11
12      pop, logbook = algorithms.eaSimple(pop, toolbox, CXPB,
     MUTPB, NGEN, stats=stats)
13
14      #Seleciona o melhor individuo da populacao resultante
15      best_ind = tools.selBest(pop, 1)
16
```

Listing: Classe algorithms

Dúvidas?