Computação Evolucionária

Implementação da Substituição por Tempo de Vida

Por: Franklin Nascimento

Pré requisitos

- Para implementar o método de substituição por tempo de vida (**tv**), cada cromossomo da população deve ter uma idade e um tv.

```python
class Chromossome:

    SIZE = 8

    def __init__(self, chromossome: list = None, lifeTime: int = 0) -> None:
        if chromossome == None:
            self.shape = np.random.randint(0, 2, (self.SIZE,))
            self.adaptation = 0
            self.fitness()
            self.age = 0
            self.lifeTime = lifeTime
        elif len(chromossome) == self.SIZE:
            self.shape = np.array(chromossome)
            self.adaptation = 0
            self.fitness()
            self.age = 0
            self.lifeTime = lifeTime
        else:
            print(f'Chromossome\'s lenght is not compatible!')
            exit()
```

Pré requisitos

- A cada geração, a idade de cada cromossomo é incrementada em 1. O cromossomo cuja idade ultrapasse o seu tv, não passará para a próxima geração.

```
def addAgeAll(self) -> None:
    for chromossome in self.chromossomes:
        chromossome.addAge()

def deathByAge(self):
    survivors = []

    for i in range(len(self.chromossomes)):
        if self.chromossomes[i].age <= self.chromossomes[i].lifeTime:
            survivors.append(self.chromossomes[i])

    self.chromossomes.clear()
    self.chromossomes = survivors
```

Pré requisitos

- A idade de cada novo cromossomo é determinada com base na adaptação máxima e mínima e no tv mínimo da população atual. Para isso, a população guardará essas informações. O tv da 1ª geração é gerado aleatoriamente.

```
class Population:

    SIZE = 10
    OFFSET = 5              # n | n < SIZE
    EVOLUTION_RATE = 1      # 0 - 10
    CROSSOVER_RATE = 0.4    # 0 - 1
    MUTATION_RATE = 0.1     # 0 - 1
    INVERTION_RATE = 0.1    # 0 - 1
    ROULETTE_SIZE = 10

    def __init__(self) -> None:
        self.chromossomes = self.generate()
        self.offspring = []
        self.minLifeTime = 999999
        self.minAdaptation = 999999
        self.maxAdaptation = -999999
        self.setMinLifeTime()
        self.setMinAdaptation()
        self.setMaxAdaptation()
```

Pré requisitos

- O cálculo do tv de cada cromossomo pode ser feito usando a estratégia proporcional, linear ou bilinear. Para esta implementação, selecionamos o método linear dado pela seguinte expressão:

$$tv = min_{tv} + 2\eta \frac{f[i] - min_{|f|}}{max_{|f|} - min_{|f|}}$$

- Onde η é uma constante de evolução que pode variar de 0 a 10.

```
def calculateLifeTime(self, adaptation: int) -> int:
    return int(self.minLifeTime + 2*self.EVOLUTION_RATE*((adaptation - self.minAdaptation)/(self.maxAdaptation - self.minAdaptation)))
```

Execução

- Desta forma, aplicamos os seguintes passos para realizar a substituição por tempo de vida:
  - **merge()**: juntamos a população atual com a população de descendentes
  - **addAgeAll()**: adicionamos +1 na idade de todos os cromossomos
  - **deathByAge()**: aqui excluímos todos os cromossomos que ultrapassaram o seu tempo de vida
  - **updateHyperParameters()**: atualizamos os hiperparâmetros da população (tv mínimo e adaptações mínima e máxima)
  - **sort()**: ordenamos os cromossomos baseados na sua adaptação

```
def lifeTime(self):
    self.merge()
    self.addAgeAll()
    self.deathByAge()
    self.updateHyperParameters()
    self.sort()
```

FIM