UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS DE FLORESTAL
CIÊNCIA DA COMPUTAÇÃO

HEITOR PASSEADO SOARES(3055) E VITOR HUGO DE OLIVEIRA(3049)

META-HEURÍSTICAS

TRABALHO PRÁTICO 2

FLORESTAL, MG

25 de setembro de 2021

HEITOR PASSEADO SOARES(3055) E VITOR HUGO DE OLIVEIRA(3049)

META-HEURÍSTICAS
TRABALHO PRÁTICO 2

Documentação do 2º trabalho prático de Meta-heurísticas que tem como objetivo o desenvolvimento e a aplicação de algoritmo genéticos em dois problemas reais distintos

Orientador: Prof. Dr. Marcus Henrique Soares Mendes

FLORESTAL, MG

25 de setembro de 2021

Sumário

1 Introdução

Este trabalho consiste na implementação e aplicação de algoritmos genéticos para a solução de um problema matemático é outro problema de minimização do custo do combustível para 13 unidades geradoras, comparando os seus desempenhos e resultados obtidos com 30 iterações para cada problema.

2 Desenvolvimento

2.1 Linguagem e Packages

Antes de dar início a solução das atividades propostas a dupla tomo algumas decisões tais como linguagem e packages. A linguagem usada foi a python3,assim para o projeto foi-se definido a utilização de ambiente virtuais em python os chamados venv, para que a execução do projeto fica-se contido em um ambiente que não depende-se totalmente da máquina a qual este sera executado.

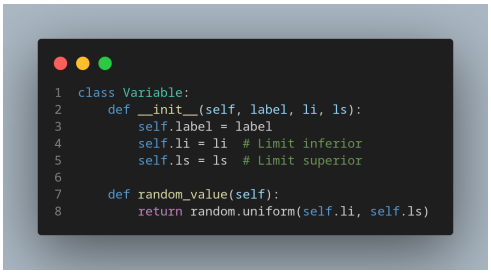
Para facilitar e otimizar o código, optamos pela utilização do package numpy este que será responsável por gerenciar aos vetores necessários durante o código. Como utilizações principais do numpy temos o vetor de genes e o vetor de indivíduos da população, ambos serão mais explicado durante o texto.

2.2 Algorítimo Genético

Foi realizado a criação do algoritmo genético de forma abstrata a qual permitisse que ambos os problemas focos, fossem resolvidos através do mesmo algoritmo, sendo somente necessário a alteração dos paramentos de cada problema. O único ponto ao qual o algorítimo não é maleável e quanto ao seu fator de minimização da função objetivo passada.

Antes de definirmos as principais classes referentes algoritmo genético iremos definir algumas classes de estruturas de dados básicas que dão suporte para as demais. a primeira destas é a classe Varible(figura 1) esta armazenar o parâmetro Label, referente ao nome verdadeiro da variável no problema, o parâmetro li, limite inferior e o parâmetro ls, limite superior.

Tabela 1 – Varible



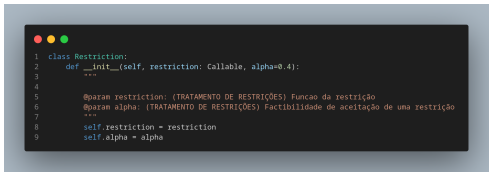
```python
class Variable:
    def __init__(self, label, li, ls):
        self.label = label
        self.li = li  # Limit inferior
        self.ls = ls  # Limit superior

    def random_value(self):
        return random.uniform(self.li, self.ls)
```

Outra classe de apoio é a Restriction(figura 2) esta que é responsável por armazenar os parâmetros de restrições a qual a função objetivo deve-se ter em consideração. Os

parâmetros dessa classe são restrictions, referente a própria função de restrição e alfa, Referente ao valor de factibilidade de aceitação de uma restrição .

Tabela 2 – Restriction



```python
class Restriction:
    def __init__(self, restriction: Callable, alpha=0.4):
        """

        @param restriction: (TRATAMENTO DE RESTRIÇÕES) Funcao da restrição
        @param alpha: (TRATAMENTO DE RESTRIÇÕES) Factibilidade de aceitação de uma restrição
        """
        self.restriction = restriction
        self.alpha = alpha
```

A primeira classe definida e relacionada diretamente com o algoritmo genético e o *Problem*, que armazenará as propriedades principais do problema e as propriedades do algorítimo genético para cada problema. Na figura 3 conseguimos visualizar a definição da classe *Problem*, neste passamos os parâmetros:

- **variables**(Tipo Variable): Define as variáveis do problema
- **objective**(Tipo Callable): Define a função objetiva
- **restrictions**(Tipo Restriction): Define as restrições que penalizaram o fitness
- **elitism_rate** (Tipo float): Porcentagem da população que será instantaneamente selecionada para próxima geração.
- **t_individuals** (Tipo inteiro) Numero de indivíduos que participarão do torneio
- **cut_point** (Tipo float) Porcentagem do corte para cruzamento truncado
- **n_generations** (Tipo inteiro) Número de gerações que o algoritmo genético executará

2.2.1 Restrições e Penalidade

Para nos ater as restrições foi utilizado o algoritmo de penalidade, o exponencial da penalidade foi setado como 3 e o fator de factibilidade foi setado como 1, mesmo com esse enfoque nas restrições o somatório dos valores de p do problema 2 ainda variavam mais do que o esperado, chegam a até 6 unidades de margem de erro.

2.2.2 Mutação

Em cada geração há uma chance de 1% de se aplicar o vetor de pertubação em um dos filhos, escolhido aleatoriamente, a partir do momento que um dos filhos é escolhido tenta-se encontrar valores de dna dentro de um laço infinito de forma que se respeite os limites inferior e superior daquele gene, ou variável.

```
self.boundaries = [
    Variable("cut_point", 0.2, 0.8),
    Variable("elitism_rate", 0.01, 0.1),
    Variable("n_generations", 100, 1000),
    Variable("ohm", 0.0, 0.1),
    Variable("t_individuals", 2, 6)
]
```

Figura 2 – Limites para o otimizador

2.2.3 Problema 2

2.2.3.1 Modelagem

Primeiramente foram definidas as 13 unidades geradoras:

```
units = [
    {'min': 0,  'max': 680, 'a': 0.00028,'b': 8.10,'c': 550, 'e': 300, 'f': 0.0035},
    {'min': 0,  'max': 360, 'a': 0.00056,'b': 8.10,'c': 309, 'e': 200, 'f':0.042 },
    {'min': 0,  'max': 360, 'a': 0.00056,'b': 8.10,'c': 307, 'e': 150, 'f':0.042 },
    {'min': 60, 'max':  180, 'a': 0.00324,'b': 7.74,'c': 240, 'e': 150, 'f':0.063 },
    {'min': 60, 'max':  180 , 'a': 0.00324,'b': 7.74,'c':240 , 'e': 150, 'f':0.063 },
    {'min': 60, 'max':  180 , 'a': 0.00324,'b': 7.74,'c':240 , 'e': 150, 'f':0.063 },
    {'min': 60, 'max':  180 , 'a': 0.00324,'b': 7.74,'c':240 , 'e': 150, 'f':0.063 },
    {'min': 60, 'max':  180 , 'a': 0.00324,'b': 7.74,'c':240 , 'e': 150, 'f':0.063 },
    {'min': 60, 'max':  180 , 'a': 0.00324,'b': 7.74,'c':240 , 'e': 150, 'f':0.063 },
    {'min': 40, 'max':  120 , 'a': 0.00284,'b':8.60 ,'c':126 , 'e': 100, 'f':0.084 },
    {'min': 40, 'max':  120 , 'a': 0.00284,'b':8.60 ,'c':126 , 'e': 100, 'f':0.084 },
    {'min': 55, 'max':  120 , 'a': 0.00284,'b': 8.60,'c':126 , 'e': 100, 'f':0.084 },
    {'min': 55, 'max':  120 , 'a': 0.00284,'b': 8.6,'c':126 , 'e': 100, 'f':0.084 },
]
```

Figura 1 – Unidades geradoras

O problema foi definido com o dna de cada indíviduo contendo um vetor de 13 posições para cada valor que p poderia assumir, o somatório de cada unidade geradora com a fórmula dada menos a penalidade definia a função fitness.

2.2.3.2 Restrições

A restrição dos limites de P são feitas durante a mutação(vide seção mutação), a restrição para o valor do somatório de P's ser igual a 1800 foi quebrada em duas restrições de desigualdade utilizando o valor de $= 0.0000000001$.

2.2.4 Escolha dos parâmetros e otimizador

Para termos uma noção exploratória primeiro, criamos uma classe Otimizer, que possui limites para as variáveis: ponto de corte, taxa de elitismo, número de gerações, sigma(mutação), tamanho t para torneio. O otimizador escolhia valores aleatórios para os intervalos definidos: E mostrava para uma população de 100 quais foram os fitness adquiridos do melhor indivíduo, foi escolhido manualmente valores bons para definir o valor dos parâmetros e executar as 30 vezes de teste:

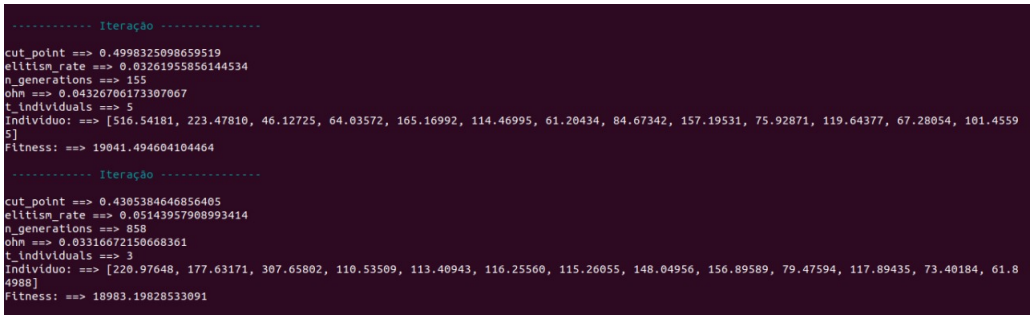


Figura 3 – Exemplos de saída do otimizador para o problema 2

3 Resultados obtidos

As configurações para o problema 1 e 2 foram definidas de acordo com a seção "Escolha de parâmetros"e o tamanho da população foi 400 pois acima disso já começava a ficar bem lento.

```
problem_1 = Problem([x1, x2], objective,
                    [Restriction(restriction1), Restriction(restriction2)],
                    elitism_rate=0.049543083097609256,
                    cut_point=0.7424666860388688,
                    t_individuals=2,
                    n_generations=262, )
```

Figura 4 – Especificações problema 1

```
problem_2 = Problem(get_variables(), objective_2, get_restrictions(),
                    elitism_rate=0.0667236920820883,
                    cut_point=0.6834153362370213,
                    t_individuals=3,
                    n_generations=175, )
```

Figura 5 – Especificações do problema 2

Para considerarmos as configurações a e b de cada problema, apenas alternamos a ordem para a segunda execução

3.1 Problema 1

Algoritmo	Mínimo	Máximo	Média	desvio padrão
AG(a)	-7938.39	-5340.32	-7309.44	643.531
AG(b)	-7954.12	-5727.08	-7221.73	620.055

Problema 1

3.2 Problema 2

Algoritmo	Mínimo	Máximo	Média	desvio padrão
AG(a)	18880.3	19147.5	18996.6	63.7258
AG(b)	18823.3	19154.2	18996.7	81.9115

Problema 2 13 variáveis definidas no documento de especificação

4 Conclusão