Universidade Federal de Viçosa
Campus Florestal - Ciência da Computação
CCF 480 - META HEURÍSTICAS
Aryel Penido - 3500
Taianne Motta - 2679

Trabalho prático 2:

Algoritmo genético

Florestal
2021

Introdução

Esse trabalho teve como objetivo a implementação de um algoritmo genético. O algoritmo deveria ser implementados para Minimizar as seguintes funções objetivo:

1. $G6(x) = (x1 - 10)^3 + (x - 20)^2$ para as seguintes restrições de domínio
    a. 13 <= x1 <= 100
    b. 0 <= x2 <=100

   E sujeita às seguintes restrições :
    c. g1(x) = -(x1 -5)² - (x2 - 5)² + 100 <= 0
    d. g2(x) = (x1-6)² + (x2 - 5)² + 82.81 <=0

As configurações utilizadas foram:

Configuração A		
nVariaveis	tamanho população	número iterações
2	100	10

Configuração B		
nVariaveis	tamanho população	número iterações
2	1000	100

Implementação

O código para os algoritmos do trabalho foram desenvolvidos em python com a ajuda de um tutorial que está referenciado ao final deste documento. O algoritmo genético foi implementado com representação real, o *método de cruzamento* foi o aritmético, o de *seleção* foi a proporcional ao fitness (aqui a função fitness é a função objetivo) e o *critério de parada* foi o número de iterações definidos nas tabelas acima. A penalidade escolhida para as restrições foi a de morte, os indivíduos que não fossem válidos para g1 e g2 eram descartados. O Arquivo AGValidacao tem uma implementação do algoritmo ajustado para função x² com limites entre 0 e 10. O resultado esperado é 0 ou algo muito próximo de 0, o resultado de uma execução é mostrado abaixo.

```
#########################################################
geração:  0
melhor solucao intermediaria:  struct({'x1': 0.01047905253724335, 'sol': 0.00010981054207830628})
#########################################################
geração:  1
melhor solucao intermediaria:  struct({'x1': 0.01047905253724335, 'sol': 0.00010981054207830628})
#########################################################
geração:  2
melhor solucao intermediaria:  struct({'x1': 0.01047905253724335, 'sol': 0.00010981054207830628})
#########################################################
geração:  3
melhor solucao intermediaria:  struct({'x1': 0.01047905253724335, 'sol': 0.00010981054207830628})
#########################################################
geração:  4
melhor solucao intermediaria:  struct({'x1': 0.01047905253724335, 'sol': 0.00010981054207830628})
#########################################################
geração:  5
melhor solucao intermediaria:  struct({'x1': 0.01047905253724335, 'sol': 0.00010981054207830628})
#########################################################
geração:  6
melhor solucao intermediaria:  struct({'x1': 0.01047905253724335, 'sol': 0.00010981054207830628})
#########################################################
geração:  7
melhor solucao intermediaria:  struct({'x1': 0.0023665024387740097, 'sol': 5.600333792723336e-06})
#########################################################
geração:  8
melhor solucao intermediaria:  struct({'x1': 0.0023665024387740097, 'sol': 5.600333792723336e-06})
#########################################################
geração:  9
melhor solucao intermediaria:  struct({'x1': 0.0023665024387740097, 'sol': 5.600333792723336e-06})
melhor solução:  struct({'x1': 0.0013302026302320913, 'sol': 1.7694390374763739e-06})
```

execução para x²

O problema e os parâmetros foram definidos utilizando a biblioteca structure do ypstruct.

```
#problema
problema = structure()
problema.funcao = G6
problema.nVariaveis = 2
problema.limitInf = [13,0]
problema.limitSup = [100,100]

#parametros
params = structure()
params.maxIt = 100
params.npop = 1000 #aumentar depois
params.sigma = 0.1

#individuo
individuo = structure()
individuo.x1 = None
individuo.x2 = None
individuo.sol = None
```

estrutura de indivíduos

O cruzamento foi feito utilizando a fórmula $Y = up1 + (1 - u) * p2$ onde p1 e p2 são os indivíduos pais.

```
def cruzamento(p1x1, p1x2, p2x1, p2x2):
    f1.x1 = p1x1
    f1.x2 = p1x2
    f2.x1 = p2x1
    f2.x2 = p2x2
    alpha = np.random.uniform(0,1)

    f1.x1 = alpha*p1x1+(1-alpha)*p1x2
    f1.x2 = alpha*p1x2+(1-alpha)*p1x1

    f2.x1 = alpha*p2x1+(1-alpha)*p2x2
    f2.x2 = alpha*p2x2+(1-alpha)*p2x1
```

cruzamento

A mutação foi feita utilizando $vk = \sigma(bk - ak)(2U(0,1) - 1)$

```
#mutacao
def mutacao(x):
    y = x.deepcopy()
    u = np.random.uniform(0,1)
    sigma = 0.1
    aux = y.x1
    aux2 = y.x2
    y.x1 = sigma*(problema.limitSup[0] - problema.limitInf[0])*((2*u) - 1)
    y.x1 = y.x1 + aux
    y.x2 = sigma*(problema.limitSup[1] - problema.limitInf[1])*(2*u)
    y.x2 = y.x2 + aux2
    return y
```

mutação

Resultados

Para a configuração A e B os resultados encontrados estão descritos na tabela abaixo

Algoritmo	Mínimo	Máximo	Média	Desvio-padrão
AG configuração A	-7535.610848315226	3183.598796581167	-2305.2059551845846	2898.302943203686
AG configuração B	-7818.1681165828095	-4239.0234341096	-6395.040656411191	962.7966803229893

o menor valor encontrado nas 30 iterações para configuração A foi
G6 = -7535.610848315226 com x1 = 17.032378903676694 e
x2 = 0.09764882424958543 durante a execução 25

o menor valor encontrado nas 30 iterações para configuração A foi
G6 = -7818.1681165828095 com x1 = 15.569422475791814 e
x2 = 0.007566977376638473 durante a execução 23

Gráficos com boxplot

Gráfico da Configuração A:

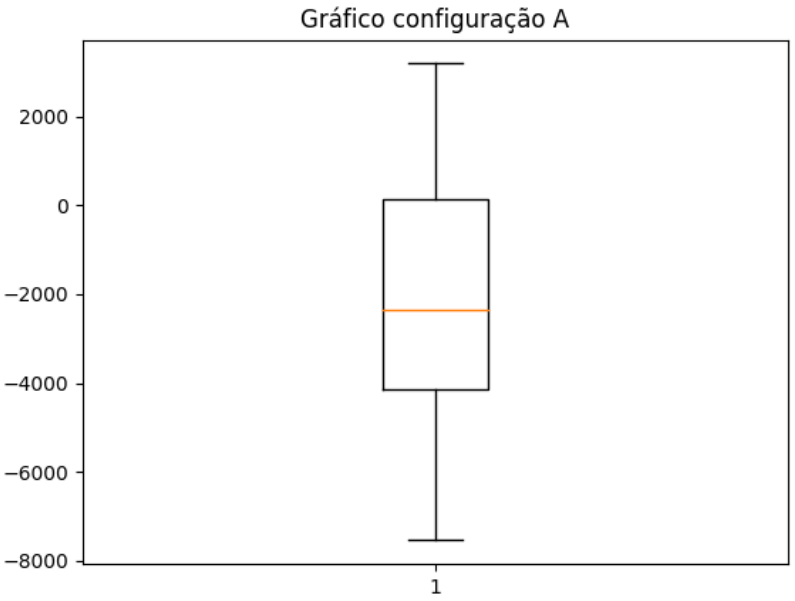
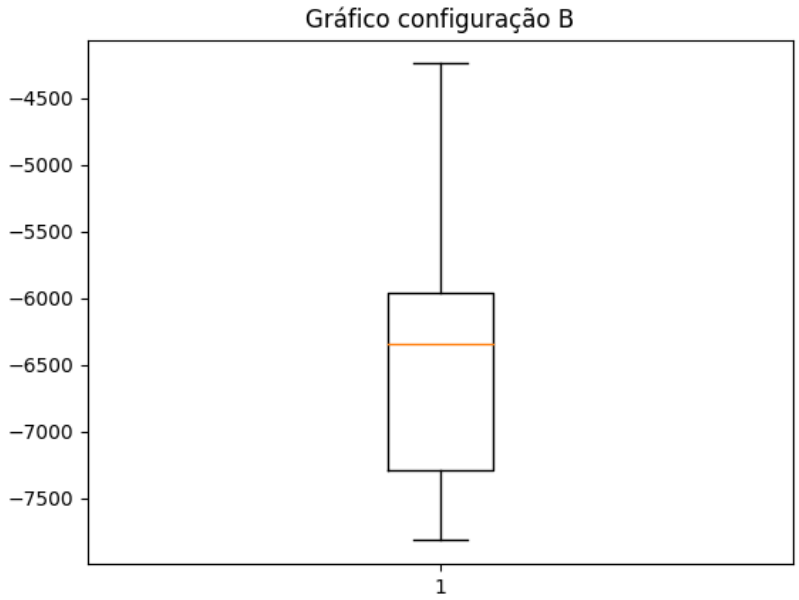


Gráfico da Configuração B:



Conclusão

Os resultados que encontramos mostram que aumentar o tamanho da população e o número de iterações faz com que a distribuição das soluções encontradas fique mais próxima, como mostra o desvio padrão menor em B. Além disso, os mínimos gerais também diminuíram, o que mostra que quanto maior a população alinhada a um número grande de iterações melhor é a solução encontrada.

Referências

Genetic Algorithm in Python - Part A - Practical Genetic Algorithms Series *Youtube. Disponível em:* <https://www.youtube.com/watch?v=PhJgktRB1AM>*. Acesso em: 24 de set. 202021. 0:42:39*

Genetic Algorithm in Python - Part B - Practical Genetic Algorithms Series. *Youtube.Disponível em:* <https://www.youtube.com/watch?v=gIIygj3UlBs&t=447s>*. Acesso em: 24 de set. 202021. 0:33:49*