Relatório sobre o Trabalho 2 de IA
Nome: Brendon Mauro
Semestre: 2020/2

Obs.: Neste relatório, farei apenas uma breve explicação sobre o meu código.

O código foi implementado com a linguagem Python na versão 3, e pode ser executado com o comando python3 se o mesmo estiver instalado no seu computador. É apenas um arquivo (trabalho2_IA.py) e não foi utilizado conceitualmente o paradigma de orientação objeto (embora a linguagem utiliza objetos para todas as variáveis declaradas), portanto você terá as funções necessárias para a resolução do problema antes da main, que é a função que será executada.

A implementação do código foi baseado nos vídeos e conteúdo disponibilizados no AVA, sendo eles os conceitos sobre Algoritmos genéticos, que não necessariamente segue uma regra estrita de implementação, porém é importante utilizar os principais conceitos que são: indivíduos, população, crossover, mutação, seleção por torneio, elitismo, etc.

Sobre as constantes

```
#constantes
numBits = 16
qtdPopInicial = 10
tamMaximoPop = 100
qtdElitismo = 20
qtdGeracoes = 20
minX = -20
maxX = 20
k = 5
n = 2
```

Para os principais parâmetros utilizados na solução, declarei as constantes que pode ser vista na figura acima:
numBits: É o número de bits que cada indivíduo pode ter. No problema diz que eles terão exatamente 16 bits, mas coloquei ali com opção para poder explorar outros tamanhos de bits também.
qtdPopInicial: É referente ao número de indivíduos inicial criado aleatoriamente.
qtdElitismo: É referente ao número dos melhores indivíduos selecionados para a próxima geração.
qtdGerações: O número de gerações que será executada na solução
minX: O número mínimo de X da função descrita no problema
maxX: O número máximo de X da função descrita no problema
k = O número de números indivíduos selecionados aleatoriamente no torneio
n = O número de novos indivíduos gerados a cada torneio se houver crossover

Parte inicial

```
print('Iniciando a solução ...')
print()

# gerando os primeiros individuos
print('Gerando população inicial')
individuos = [gerarIndividuoAleatorio() for i in range(qtdPopInicial)]
print('imprimindo individuos em tuplas (binarios, decimal, transformaPraX, apitidao)')
[print ((individuo,individuoBinToDecimal(individuo),encontrarX(individuo), getIndividuoAptidao(individuo)[1])) for individuo in individuos]
[print for individuo in individuos]

individuosAptidoes = [getIndividuoAptidao(individuo) for individuo in individuos]
individuosAptidoes.sort(key=orderByAptidao)
melhorDaGeracao = individuosAptidoes[0][0]
print()
print('melhorDaGeracao')
print(((melhorDaGeracao,individuoBinToDecimal(melhorDaGeracao),encontrarX(melhorDaGeracao), getIndividuoAptidao(melhorDaGeracao)[1])))
```

Nesta etapa é selecionada os 10 primeiros indivíduos aleatoriamente, pois depois disso os novos indivíduos serão criados apenas a partir de crossover ou mutação. Nesta parte também é impresso os indivíduos apenas por questões de visualização. Também é impresso uma tupla com o melhor indivíduo da geração, o número dele em decimal, o X respectivo dele e sua aptidão.

Executando as gerações

```
print(((melhorDaGeracao,individuoBinToDecimal(melhorDaGeracao),encontrarX(melhorDaGeracao), getIndividuoAptidao(melhorDaGeracao)[1])))
for i in range(qtdGeracoes):
    individuos = aplicarElitismo(individuos)

    while len(individuos) < tamMaximoPop:
        aplicarTorneio(individuos)

    aplicarMutacoes(individuos)

    individuosAptidoes = [getIndividuoAptidao(individuo) for individuo in individuos]
    individuosAptidoes.sort(key=orderByAptidao)
    melhorDaGeracao = individuosAptidoes[0][0]

    print()
    print('melhorDaGeracao')
    print(((melhorDaGeracao,individuoBinToDecimal(melhorDaGeracao),encontrarX(melhorDaGeracao), getIndividuoAptidao(melhorDaGeracao)[1])))
```

Para executar as gerações eu segui a seguinte ordem: é feito o elitismo, depois é aplicado os torneios até atingir o tamanho máximo da população. Depois é aplicado as mutações, e por fim é ordenada por aptidão os individuos para escolher o melhor da geração, e o resultado impresso é da mesma forma da Parte Inicial.

Torneio

```
def aplicarTorneio(individuos):
    posicoes = []
    tam = len(individuos)
    qtd = k if k < tam else tam

    for i in range(qtd):
        posicao = random.randint (0,tam-1)

        while posicao in posicoes:
            posicao = random.randint (0,tam-1)

        posicoes.append(posicao)

    individuosSorteados = [getIndividuoAptidao(individuos[posicao]) for posicao in posicoes]
    individuosSorteados.sort(key=orderByAptidao)
    pais = individuosSorteados[:2]
    pais = [pai[0] for pai in pais]

    probabilidadeCrossOver = random.random()

    if probabilidadeCrossOver <= 0.6:
        for i in range(n):
            individuos.append(aplicarCrossOver(pais))

    #aplicarMutacoes(individuos)
```

Sobre o torneio, é escolhido uma quantidade k de indivíduos aleatoriamente, caso esse k seja maior que a população é escolhida a população toda. Após o sorteio dos indivíduos, é ordenado por aptidão, e é selecionado os pais com a melhor aptidão. Após isso é aplicado a probabilidade de 60% do crossover, e é gerado os filhos caso entre nas condições de 60% de probabilidade.

Crossover

O processo de crossover para gerar os filhos é bem simples: É selecionado a metade dos bits do lado esquerdo do primeiro pai, e metade dos bits do lado direito do segundo pai, então com esses bits é gerado o filho.

Mutação

```
def aplicarMutacoes(individuos):
    for individuo in individuos:
        for i in range(len(individuo)):
            probabilidadeMutation = random.random()
            if probabilidadeMutation <= 0.01:
                individuo[i] = 0 if individuo[i] == 1 else 1
```

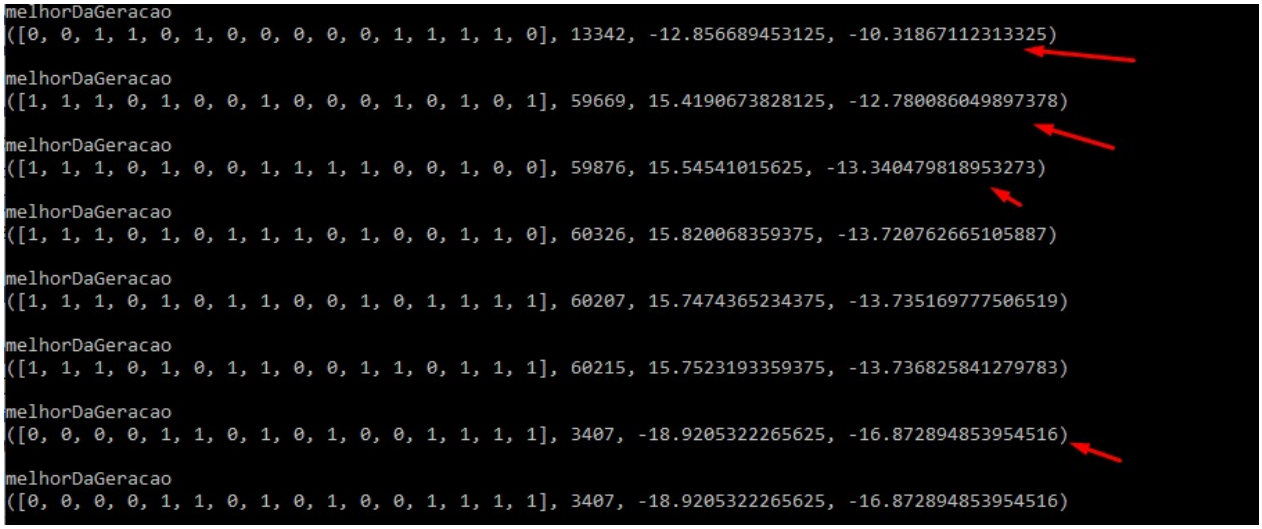
Foi feito inicialmente o processo de mutação dentro do torneio com os pais selecionados, e com a taxa de 1% de probabilidade de mutação. Porém percebi que a diversidade não aumentava desta forma, então eu simplesmente peguei toda a população, varri bit a bit e coloquei a probabilidade de 1% para cada bit, e nesse processo obtive resultados muito melhores quanto a diversidade, antes não importava o número de gerações, ele sempre acabava caindo nos mínimos locais, ao invés de gerar indivíduos com maior diversidade e com o passar das gerações direcionar para o mínimo global (melhor aptidão).

Elitismo

O processo de elitismo também é bem simples, no processo de geração é selecionado os indivíduos com as melhores aptidões, e esta quantidade pode ser alterada editando a constante qtdElitismo.

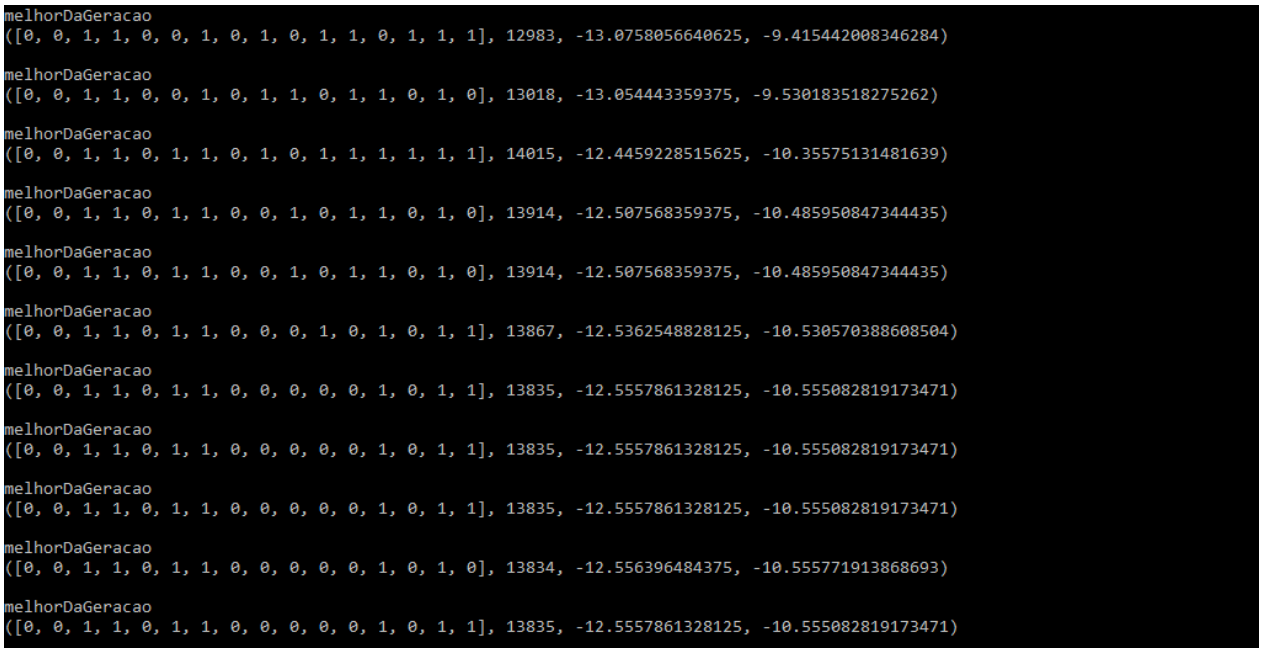
Resultados

Com 10 gerações, e após implementar a mutação conforme prevista acima, obtive resultados satisfatórios como este exemplo aqui:



Percebe-se que apesar dele ter se concentrado por algumas gerações na aptidão -13, ele conseguiu gerar uma população mais diversa, e encontrou o um mínimo menor ainda. Que é mais próximo do mínimo global.

Porém, não é sempre que os resultados são tão satisfatórios para apenas 10 gerações, em outros testes, podemos encontrar resultados como o seguinte abaixo:



Já para 20 gerações, embora muitas vezes acabe encontrando o valor próximo ao mínimo global, nem sempre isso pode ser garantido. O que faz com que seja importante explorar um número maior de gerações, para problemas maiores, ou também um número maior da população. Desde que esteja dentro da capacidade de processamento do computador utilizando, e não tenha que esperar um tempo exorbitante.