Atividade Avaliativa III - Algoritmos Genéticos

Giovanni F. S. Rebouças[1], Felipe A. da S. Biazatti[2], Pedro A. B. Bispo[3]
[1,2,3]Institute of Technological Sciences, Universidade Federal de Itajubá, Itabira, Minas Gerais, Brazil
RA: 34270[1], 34452[2], 2016002449[3]
giovannireboucas@hotmail.com, felipeamorimsbiazatti@gmail.com, pedro_augustobastos@hotmail.com

***Resumo*—Este trabalho visa fazer uma breve explicação sobre o que são funções multimodais, quais as aplicações, quais são as dificuldades inerentes à aplicação destas e, então, exposto esse conhecimento, implementar um Algoritmo Genético que aja como ferramenta para encontrar o mínimo global dessa função num espaço de busca de números inteiros com 4 casas decimais de precisão.**

Este trabalho oferece ainda todos os elementos acessórios à implementação do algoritmo genético, como modelagem do indivíduo, tamanho de população, tipos e taxas de cruzamento e mutação, método de seleção, número de geração, técnicas de elitismo utilizadas e análises estatísticas dos resultados e dos valores de média, variância e desvio padrão da *fitness*, que permitam a análise da confiabilidade do resultado obtido pela implementação de algoritmos genéticos deste trabalho.

***Palavras-chave*—Algoritmos Genéticos, Minimização, Python, Computação Natural, Função Multimodal.**

I. Introdução

Durante o desenvolvimento e evolução da computação e suas diversas ferramentas, um dos fenômenos mais comumente observados é o uso de referências do mundo real como inspiração para a solução de problemas computacionalmente complexos. Dessa inspiração, a teoria da evolução é a principal fonte para o desenvolvimento de uma meta heurística de otimização chamada de algoritmos evolucionários. Esses algoritmos evolucionários faziam uso principalmente de operadores como reprodução, mutação, recombinação e seleção de indivíduos como sua base funcional. Um tipo de algoritmo evolucionário são os chamados algoritmos genéticos, que fazem uso de um subconjunto específico de operadores genéticos: mutação, cruzamento e seleção.

Os algoritmos genéticos fazem uso desses operadores genéticos para manipular a composição de um número de gerações que, por sua vez, são compostas por indivíduos cujos objetivos seriam atingir um certo valor de adequação (do inglês, *fitness*) à um parâmetro, valor ou função previamente definida. Algoritmos genéticos encontram especialização e amplo uso como geradores de soluções para problemas de otimização e problemas de busca.

II. Revisão de Literatura

Baseado no objetivo deste trabalho, é de extrema necessidade que seja feita uma concisa e breve explicação sobre os componentes majoritários que compõem o problema e a solução proposta. Primeiro, uma breve apresentação sobre como é estruturado um Algoritmo Genético, o que são e para que servem Funções Multimodais e uma breve explicação da Função de Interesse.

A. Algoritmo Genético

Uma população de indivíduos, cada um representando uma possível solução para um determinado problema, é mantida dentro do espaço de busca de um algoritmo genético. Esses indivíduos são codificados como vetores de comprimento finito dos componentes ou variáveis, com base em um alfabeto, comumente, o binário 0,1. Cada um desses indivíduos são comparados com cromossomos, e cada variável é análoga aos genes, de modo que cada cromossomo (solução) é composto de diversos genes (variáveis). [1]

Uma população inicial de indivíduos, chamada de uma geração, é gerada de maneira aleatória, de modo que cubra, de maneira abrangente, todo o espaço de busca. Em cima dessa e das subsequentes gerações, seguimos o seguinte processo:

- O *fitness* de todos os indivíduos é calculado;
- Os indivíduos são selecionados de maneira estocástica através de métodos como roleta ou por torneio;
- Seleções especializadas, como Elitismo, são feitas para garantir que os melhores genes sejam perpetuados para a próxima geração;
- Em cima dos indivíduos selecionados, opera-se o cruzamento para que cada par de genitores gere proles que componham parte da próxima geração;
- Por fim, indivíduos são selecionados aleatoriamente para sofrerem mutações, finalizando, assim, a composição da nova geração.

As operações citadas são feitas em cima de diversas gerações, uma após a outra, até que se atinja um número específico de gerações ou que uma condição seja satisfeita.

B. Funções Multimodais

Durante a resolução ou busca de soluções de problemas matematicamente modelados, temos o conceito de soluções ótimas locais (também chamado de mínimo local), cuja definição matemática é: [2]

$$f(x^*) \leq f(z), \forall z \in N(x^*) \tag{1}$$

Uma função matemática pode ser definida com base na quantidade de ótimos locais que essa possuí, de tal modo que, funções que possuem múltiplos ótimos locais, são chamadas de funções multimodais. [3]

Essas funções encontram aplicabilidade principalmente nas demonstrações de performance de algoritmos de busca e de otimização devido à razoável facilidade em que se pode aumentar a complexidade dos estados de espaços de busca criados

por elas. As funções multimodais podem ser classificadas de acordo com alguns parâmetros como: [3]

- Separabilidade - Se uma função de p variáveis pode ser escrita como na soma de p funções de uma variável, ela é dita separável;
- Regularidade - Se uma função é diferenciável em cada ponto do seu domínio, ela é dita regular; e
- Dimensionalidade - Diz respeito à quantidade de dimensões à qual a função é sujeita.

C. Função de Interesse

Para este trabalho, desejamos encontrar o mínimo global da função multimodal abaixo, para um par de valores x_1 e x_2, onde x_1 e $x_2 \in$ [-10, 10] e x_1 e $x_2 \in \mathcal{Q}$ com 4 casas decimais de precisão.

$$f(x) = \left(\sum_{i=1}^{5} i\cos\left(i+1\right)x_1 + i\right)\left(\sum_{i=1}^{5} i\cos\left(i+1\right)x_2 + i\right) \tag{2}$$

Essa função pode ser classificada de acordo com os parâmetros citados na seção anterior, de tal modo que essa é uma função não separável, regular, multimodal e bidimensional. Um esboço dessa função dentro do espaço de busca proposto pode ser visto na figura 1

A implementação desta função na linguagem de programação escolhida para a resolução do problema proposto, bem como a maneira de invocar, suas interconexões e seus resultados serão discutidos na seção de Função Fitness.

III. Materiais e métodos

Através da definição do problema, foi concedida a liberdade de escolha de modo de implementação da solução e, por razão disto, foi feito a escolha da linguagem Python e o uso de alguns *Packages* de Terceiros Utilizados que auxiliaram na implementação do algoritmo genético. Os detalhes referentes à implementação da Função de Interesse está descrita na subseção que trata da Função Fitness, enquanto detalhes da Codificação do Indivíduo, dos Parâmetros e Hiperparâmetros escolhidos e dos Métodos de Seleção, Mutação e Cruzamento estão descritos separadamente.

A. Python

A linguagem Python nasceu no começo da década de 90, criada pelo matemático e programador Guido Van Rossum. Ele a projetou para dar ênfase ao trabalho do desenvolvedor, simplificando a escrita de um código limpo, simples e legível. Apesar de ser uma linguagem de alto nível, é utilizada nas mais diversas aplicações, web, servidores e ciência de dados.

Python é uma linguagem de programação de alto nível, interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. A linguagem segue um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation.

B. Packages de Terceiros Utilizados

O uso da linguagem *Python* nos permite acesso à uma variedade de *packages* desenvolvidos por terceiros que nos auxiliam com diversas tarefas. Para o escopo deste trabalho foram usados apenas os packages `MatPlotLib` e `NumPy`, que tratam de visualização de dados e de operações matemáticas complexas, respectivamente.

Importante frisar que não foram utilizados **nenhum** *package* de *machine-learning* ou de inteligência artificial em geral para a implementação do algoritmo genético deste trabalho e que os outros *packages* que existem dentro do arquivo de dependências `requirements.txt` estão lá pois são dependências dos 2 *packages* citados no parágrafo anterior.

C. Função Fitness

Como descrito anteriormente na subseção Função de Interesse, a função que desejemos obter o mínimo global é uma função multimodal e, por razão disto e do paralelo traçado através de algoritmos genéticos, caracteriza o que chamamos da função *fitness*.

Visto que é uma função de 2 variáveis, a função espera 2 valores x_1 e x_2 como parâmetro, tal que temos:

```
def evalFunction(x1, x2)
```

Então, calculamos os valores de cada componente de cada uma das somatórias, separadamente utilizando *list-comprehension* e salvamos os valores em uma lista. Após calculados os valores de 1 à 5 para cada uma das duas somatórias, calculamos o produto entre a somatória dos valores das 2 listas e retornamos esse valor como o resultado da função, tal que, por fim, temos a função completamente descrita em III-C.

É importante salientar, como pode ter ficado aparente da porção de código em III-C, que a função opera sobre 2 números de ponto flutuante (*floats*) e, desta maneira, os valores x_1 e x_2 devem ser convertidos de sua representação genotípica em string de bits para sua representação fenotípica.

D. Codificação do Indivíduo

Visto que o nosso espaço de busca é contínuo, que a nossa solução e, por consequência, o nosso indivíduo são compostos por pares de números reais, devemos fazer a codificação do indivíduo para uma representação mais adequada para operação utilizando algoritmos genéticos.

Esta codificação representa diretamente os parâmetros de *design* do indivíduo e, portanto, evitam etapas intermediárias de *encoding* e *decoding*, tal que podemos representar o genótipo e fenótipo de uma solução qualquer, como sendo:

Genótipo	x	y
Fenótipo	5.2856	-4.1093

onde x e y são representações codificadas dos valores 5.2856 e -4.1093.

A codificação do indivíduo levou em consideração que cada indivíduo é composto por 2 valores reais que variam de -10 à 10 com 4 casas decimais de precisão e, portanto, seu genótipo

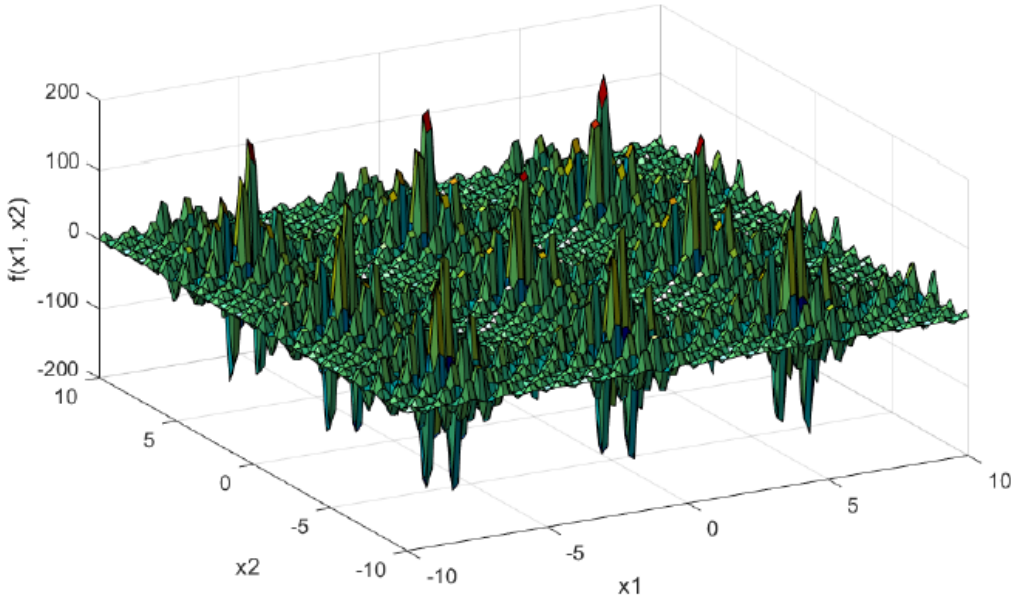

Fig. 1. Esboço da função multimodal dentro do espaço de busca proposto

Listing 1. Função fitness

```
def evalFunction(x1, x2):
    """
    Function of interest - Multimodal function
    """
    # Calculate each part of each sum separately
    listValuesFirstSum = [index * np.cos(((index + 1) * x1) + index) for index in range(1, 6)]
    listValuesSecondSum = [index * np.cos(((index + 1) * x2) + index) for index in range(1, 6)]

    # Calculate the final value of f(x) by summing and multiplying the values in the list
    fx = sum(listValuesFirstSum) * sum(listValuesSecondSum)

    return fx
```

necessita ser composto pela concatenação das codificações dos 2 valores que à compõe.

Deste modo, e sabendo que, para qualquer variável de *design* x, tal que $X_L \leq x \leq X_U$, onde X_L é o limite inferior da variável de *design* (neste caso, -10.0000) e que X_U é o limite superior da variável de *design* (neste caso, 10.0000) e, se ϵ é a **precisão** necessária, então o comprimento da *string* n deve ser igual à: [4]

$$n = log_2\left(\frac{X_U - X_L}{\epsilon}\right) \tag{3}$$

De posse dos limites superior e inferior da nossa variável de *design*, foi necessário a definição da precisão necessária para codificar os possíveis valores desta.

Sabendo que a precisão, ou precisão alcançável, ϵ está contida no intervalo [0,1] e que para o valor hipotético $\epsilon = 0.5$, temos que $4.05 \equiv 4$ ou que $4.49 \equiv 4$ e, consequentemente que $4.50 \equiv 4.5$ ou que $4.99 \equiv 4.5$ e assim sucessivamente, teremos que para uma precisão de 4 casas decimais precisamos de uma precisão alcançável de $\epsilon = 0.00005$.

Munidos dos valores de limite inferior, limite superior e de precisão alcançável, podemos agora calcular o tamanho n da *string* de bits que irá ser o genótipo de cada um dos valores do par que representa uma solução para nossa função *fitness*.

$$n = log_2\left(\frac{10 - (-10)}{0.00005}\right) \approx 18.61 \approx 19bits \tag{4}$$

Visto que o resultado da equação anterior foi de aproximadamente 18.61 e, como não é possível representar 0.61 bit, arredondamos o valor para o inteiro mais próximo maior que 18.61, ou seja, 19.

Esse resultado nos diz que cada valor do par de números reais que compõe uma possível solução possuirá uma representação binária composta por 19 bits e, consequentemente, um indivíduo completo, que é composto pela concatenação dos 2 valores que o compõem terá um total de 38 bits. Partindo dessa informação, a criação da primeira geração é feita gerando, aleatoriamente, indivíduos compostos por *strings* de 38 bits e adicionado-os à uma lista.

Quando necessário, por exemplo, para se avaliar o valor de *fitness* de um indivíduo, a decodificação do indivíduo é feita através dá seguinte equação

$$X = X_L + \left(\frac{X_U - X_L}{2^n - 1} * X_B \right) \tag{5}$$

Onde X é a representação em números reais de um genótipo e X_B é a representação em decimal do genótipo ao qual o valor X foi codificado e mapeado. Por exemplo, se quisermos saber qual é a representação X do genótipo 1111111111111111111 para o nosso problema onde $X_L = -10$, $X_U = 10$, $n = 19$ e X_B é o valor decimal da *string* de bits que está codificado em binário, teremos:

$$X = -10 + \left(\frac{10 - (-10)}{2^(19) - 1} * 524287 \right) = 10 \tag{6}$$

Da equação acima fica claro que a *string* 1111111111111111111 representa, apropriadamente o valor 10.0000 e fica óbvia a demonstração que 0000000000000000000 representa o valor -10.0000.

A implementação feita para a equação que executa a conversão do genótipo completo de 38 bits para 2 valores reais pode ser vista na porção de código III-D.

E. Parâmetros e Hiperparâmetros

Na implementação de algoritmos genéticos, a analogia com a teoria da evolução rende um conjunto de parâmetros que devem ser estabelecidos e que influenciam diretamente o funcionamento e os resultados obtidos pelo algoritmo. Uma vez que esses parâmetros não dizem respeito à função objetivo e nem necessariamente falando com as soluções, eles são comumente chamados de hiperparâmetros.

Para a implementação do algoritmo genético feito para resolver o problema proposto por este trabalho, os hiperparâmetros e valores utilizados foram:

- Tamanho da população: 500 indivíduos;
- Número de gerações: 100 gerações;
- Taxa de cruzamento: 95%; e
- Taxa de mutação: $\frac{1}{n}$, onde n é o número de genes.

Outros parâmetros que foram utilizados, mas que são consequências da especificação do problema ou das soluções, foram:

- Número de genes (n): 38;
- Intervalo de existência para as variáveis x_1 e x_2 no espaço de busca: [-10, 10]; e
- Precisão alcançável (ϵ): 0.00005 (precisão necessária para atingir 4 casas decimais).

Os hiperparâmetros foram escolhidos por inspeção e análise dos resultados obtidos e da Progressão Evolutiva para Execução Única proveniente dos testes feitos durante a fase de implementação, correção e ajustes do algoritmo genético. Por outro lado, o intervalo de existência para as variáveis foram fornecidos como parte do enunciado do problema e os parâmetros de número de genes e precisão alcançável foram obtidos empiricamente através de uma análise do enunciado e das equações expostas em Codificação do Indivíduo.

F. Métodos de Seleção, Mutação e Cruzamento

Além dos parâmetros e hiperparâmetros usados para regrar a execução do algoritmo genético, as maneiras como os operadores genéticos de seleção, mutação e cruzamento foram implementados são de suma importância e são os principais responsáveis por um funcionamento adequado do algoritmo e obtenção do objetivo desejado.

Para a implementação deste trabalho, os operadores genéticos foram especificados da seguinte maneira:

- Seleção - Por torneio com valor de k = 2 e fazendo uso de Elitismo;
- Cruzamento - Por Ponto Único, onde 1 ponto é escolhido aleatoriamente para que os genes dos pais cruzem e gerem 2 novos filhos; e
- Mutação - Por *bit string*, garante uma taxa de mutação de 1 por mutação e indivíduo selecionado para mutação.

IV. Resultados

Abaixo temos detalhamentos sobre as análises dos resultados obtidos pela ótica da Progressão Evolutiva para Execução Única e das Análises de Relevância Estatística do Algoritmo Genético. Os Problemas Encontrados e análises em relação à causa raiz e tentativas de correção também serão expostas. Por fim, uma breve explicação do Modo de Execução da implementação feita para este trabalho é feita.

A. Progressão Evolutiva para Execução Única

O algoritmo genético foi executado utilizando os hiperparâmetros com os valores especificados na seção Parâmetros e Hiperparâmetros e os dados do melhor indivíduo, do pior indivíduo e do indivíduo mediano de cada geração em relação ao *fitness* alcançado foi coletada e então um gráfico demonstrando como ocorreu a progressão evolutiva da população de indivíduos se tomou. Este gráfico pode ser visto na figura 2.

Nesse gráfico temos a progressão evolutiva por geração, onde podemos ver a inconsistência referente aos piores indivíduos da geração e a relativa consistência no melhoramento contínuo dos melhores indivíduos. Fica difícil a visualização, todavia o melhor indivíduo é encontrado por volta da geração 40, na qual há uma quase insignificante redução do valor de *fitness* para esse indivíduo. Para esta execução, o valor convergido foi de -186.7309.

B. Análises de Relevância Estatística do Algoritmo Genético

Para garantir que a implementação do algoritmo genético feita pelos autores deste trabalho tivesse alguma validade e que este é confiável para a resolução do problema proposto, foi necessário a obtenção de um conjunto maior de dados para que os resultados possuíssem relevância estatística.

Visto essa necessidade, o algoritmo foi executado 100 vezes e os dados referentes ao *fitness* alcançado pelo melhor indivíduo de cada iteração foi coletado. Essa informação foi esboçada com relação ao número da execução à qual aquele indivíduo pertence e o gráfico resultante destas execuções pode ser visto na figura 3.

Listing 2. Função de Decodificação

```
def binaryMappingToDecimal(fullChromosome):
    # Our full chromosomes are presumed to be even sized (len(fullchromosome)%2 == 0)
    # But we don't assume size, get the value of half it's length
    n = len(fullChromosome) // 2

    # Split the full lenght chromosome, into 2 half chromosomes
    # This is needed because our GA's objective function takes 2 arguments
    firstSubChromosome = fullChromosome[:n]
    secondSubChromosome = fullChromosome[n:]

    # Convert both of our subchromosomes from binary to integer
    firstChromosomeXb = int(firstSubChromosome, 2)
    secondChromosomeXb = int(secondSubChromosome, 2)

    # Calculate our conversion values
    x1 = common.XL + (((common.XU - common.XL) / ((2**n) - 1)) * firstChromosomeXb)
    x2 = common.XL + (((common.XU - common.XL) / ((2**n) - 1)) * secondChromosomeXb)

    return x1, x2
```

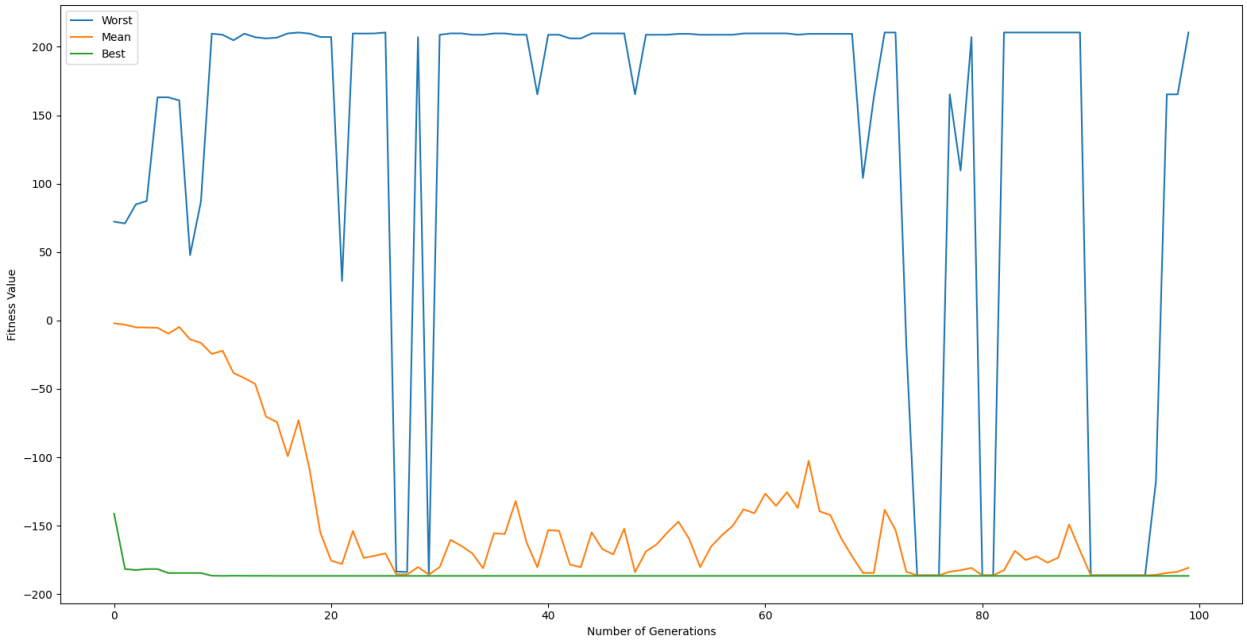

Fig. 2. Progressão evolutiva do Algoritmo Genético para uma execução única

Ao se analisar o comportamento do gráfico, podemos ver que a maioria dos valores de melhor *fitness* obtidos nas diferentes iterações do algoritmo (curva azul) se mantêm nos arredores da média (curva laranja) dos 100 *fitness* obtidos com exceção de 1 indivíduo na geração 76 cujo *fitness* varia em 0.003 em relação à média obtida para as 100 iterações de -186.7307 podendo, assim, ser tratado como uma anomalia estatística (*outlier*).

Mais do que isso, é importante frisar que o valor obtido para o desvio padrão foi de $\sigma \approx 0.0004$ e da variância foi de $\sigma^2 \approx 1.45 * 10^{-7}$. Do valor de desvio padrão temos que o intervalo de confiança de 3σ (curvas cianas) compreende todos os valores entre [-186.7319, -186.7295].

Da heurística chamada de "Regra de 3σ", que diz que se quase todos os valores podem ser encontrados dentro de um intervalo de 3σ para cada lado do valor da média, e sabendo que, satisfeita essa condição, temos uma probabilidade de 99.73% de encontrar qualquer valor da nossa população neste intervalo, então, podemos tratar empiricamente (por conveniência) 99.73% como absoluta certeza. Para o nosso experimento, desconsiderando o *outlier* da geração 76, podemos ver que **todos** os nossos melhores indivíduos estão dentro do intervalo de confiança, ou seja, o algoritmo é estatisticamente adequado.

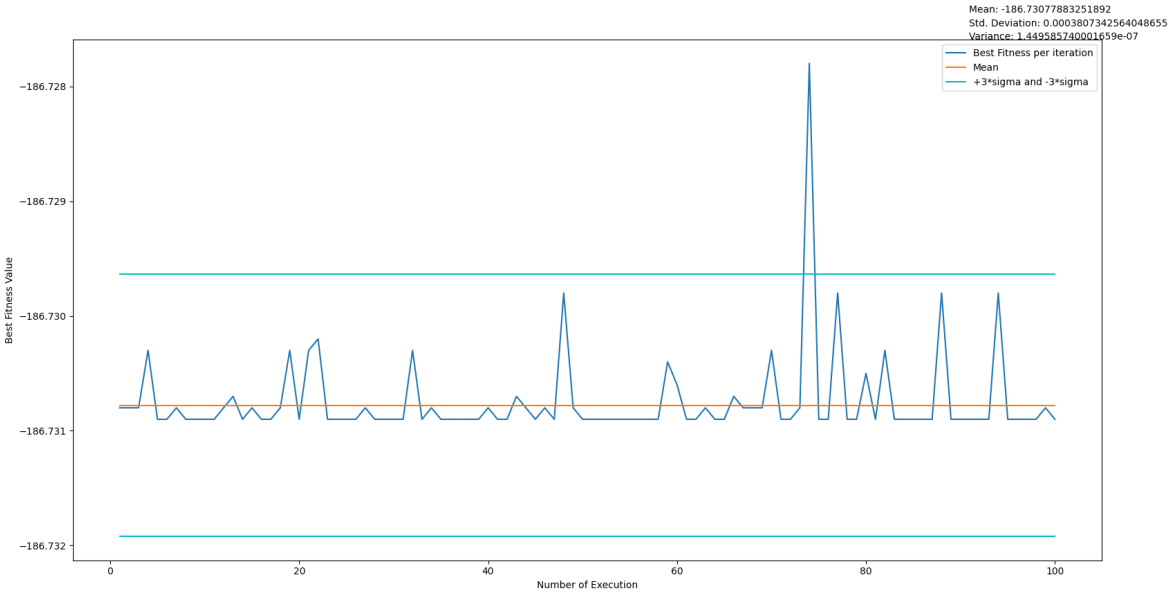

Fig. 3. *Fitness* dos melhores indivíduos em 100 iterações do Algoritmo Genético

C. Problemas Encontrados

Durante os estágios iniciais da implementação do algoritmo, houveram diversos problemas de convergência prematura da população, uma vez que toda a população convergia para valores dentro das 15 primeiras gerações, sugerindo problemas de baixa variabilidade genética ou viés advindo da pseudo-aleatoriedade da criação da população inicial, da seleção para torneio, da seleção para cruzamento ou da seleção para mutação. Ao se alterar o *seeding* das funções probabilísticas utilizadas durante a execução do código, foi possível garantir uma maior aleatoriedade, assim garantindo que não haveria nenhum viés introduzido artificialmente na execução.

Algoritmos genéticos são notoriamente sensíveis ao tamanho populacional em relação ao espaço de busca, especialmente em problemas de minimização de funções multimodais, onde mínimos locais podem ser encontrados e incorretamente selecionados como possíveis soluções. Visto esse problema e o fato do algoritmo não estar completamente otimizado, foi necessário uma população de 500 indivíduos por geração para que se pudesse obter valores de desvio padrão aceitáveis que, para este trabalho foi definido como um desvio padrão na ordem da quarta casa decimal (décimo de milésimo).

D. Modo de Execução

Uma vez que o algoritmo foi desenvolvido em *Python*, é necessário que o usuário que deseja executá-lo tenha a versão 3.7.0 ou superior da implementação padrão da linguagem (conhecida como CPython).

Tendo uma versão suportada do *Python* instalada e configurada da maneira correta, o procedimento de execução é:

1) Navegar até a pasta do projeto;
2) Criar um *virtual environment* através do **venv**: `python -m venv env`;
3) Ativar o *virtual environment* ao executar o **Activate**: `env \Scripts \Activate`;
4) Instalar as dependências através do **pip**: `pip install -r requirements.txt`; e
5) Executar o script: `python geneticAlgorithmMinimize.py`.

Note que há 2 variáveis de controle que foram implementadas para fazer um controle grosseiro para garantir que fosse possível mudar o modo de execução do *script*, elas são editáveis e **DEVEM** ser editadas pelo usuário para que este possa mudar o comportamento do mesmo. Estas são:

- `singleExecution`: Variável booleana que trata se o algoritmo será executada apenas 1 vez e o gráfico de progressão evolutiva será plottado ou se o algoritmo será executada o número de vezes especificado pela variável `numExecutions`; e
- `numExecutions`: Caso a variável `singleExecution` seja `False`, o código será executada um número de vezes especificado por esta variável.

V. Conclusão

Motivado pela necessidade de otimização da velocidade e da eficiência dos métodos, metodologias, heurística e/ou meta heurísticas de busca, e inspirado nos mecanismos biológicos que regem a evolução das espécies, foi criado a meta heurística de busca chamada Algoritmos Genéticos.

Os algoritmos genéticos nos permitem flexibilidade na busca pela otimização de problemas cujas características são as

mais diversas possíveis, como variáveis contínuas, variáveis descontínuas, analógicas, digitais, binárias, decimais ou outras. Essa flexibilidade permite a resolução de problemas de diferentes ordens e complexidades com relativa facilidade, a partir do momento que seja possível codificar as possíveis soluções como indivíduos e medida de adequação como o *fitness*.

Durante o desenvolvimento deste trabalho, foi desenvolvido e implementado um algoritmo genético em *Python* para resolver o problema da busca do mínimo global de uma função multimodal trigonométrica. Ao se deparar com um espaço de busca de mais de 40 milhões de possíveis soluções válidas, a busca por exaustão se demonstra demasiadamente ineficiente para a tarefa de buscar a solução globalmente ótima, podendo atingir tempo de execução na ordem de **dias**.

O algoritmo genético implementado neste trabalho utilizou de uma função de codificação e decodificação de valores contínuos e de ponto flutuante para valores binários que representam apropriadamente estes valores e, desta maneira, possibilita a criação dos operadores genéticos de acordo com a especificação da meta-heurística.

Através da execução desta implementação em cima do espaço de busca e função especificadas pelo problema proposto e, após uma longa etapa de escolha e ajuste dos parâmetros e hiper-parâmetros que regem este algoritmo, foi possível obter uma resposta média de -186.7307 com um desvio padrão e variância de 0.0004 e $1.45 * 10^{-7}$ respectivamente. Através do estudo de intervalos usando a regra de 3σ foi possível atestar que, salvo um *outlier*, **todos** os valores obtidos pela população de execuções do algoritmo estão contidas no intervalo de confiança.

REFERÊNCIAS

[1] Manoj Kumar Pratibha Bajpai. Genetic algorithm - an approach to solve global optimization problems. *Indian Journal of Computer Science and Engineering*, 2010.

[2] Frederico Gadelhas Guimarães. Conceitos básicos. http://www2.decom.ufop.br/imobilis/wp-content/uploads/2012/06/02_conceitos.pdf, 2012.

[3] Lillia S. Barsante Deylon C.F. Couto, Carlos A. Silva. Otimização de funções multimodais via técnica de inteligência computacional baseada em colônia de vaga-lumes. https://ssl4799.websiteseguro.com/swge5/PROCEEDINGS/PDF/CILAMCE2015-0506.pdf, 2015.

[4] Debasis Samanta. Encoding techniques in genetic algorithms. https://cse.iitkgp.ac.in/~dsamanta/courses/sca/resources/slides/GA-02%20Encoding%20Techniques.pdf, 2018.

[5] M. D. Vose. Modeling simple genetic algorithms. *Evolutionary Computation*, 3(4):453–472, 1995.

[6] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[7] Leonard J. Kazmier. *Business statistics : based on Schaum's outline of theory and problems of business statistics.* McGraw-Hill, New York, 2003.

[8] Ahmad Hassanat, Khalid Almohammadi, Esra'a Alkafaween, Eman Abunawas, Awni Hammouri, and V. B. Surya Prasath. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information - MDPI*, 10(12):390, 2019.

[9] Ms. Shikha Malik and Mr. Sumit Wadhwa. Preventing premature convergence in genetic algorithm using dgca and elitist technique. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(6):410–418, 2014.