Programa Institucional de Bolsas de Iniciação Científica – PIBIC

RELATÓRIO PARCIAL

IDENTIFICAÇÃO DO PROJETO

Título do Projeto: **Estratégias de diversificação em meta-heurísticas aplicadas a problemas de Otimização em Engenharia Elétrica**.

Local de Realização Unidade/Instituto/Departamento/Laboratório): Escola de Engenharia / Departamento de Engenharia Elétrica

Endereço: Rua Passo da Pátria 156, Sala 421 Bloco D_____

Bairro: São Domingos _____**Cidade**: Niterói_____**UF**: RJ_____**CEP**: 24210-240 _____

DADOS DO ORIENTADOR

Nome: Rainer Zanghi_____

Matrícula Siape:2393115_____CPF:03420509740_____

Endereço: Rua Passo da Pátria 156, Sala 421 Bloco D

Bairro: São Domingos___ **Cidade**: Niterói_____

UF:RJ_____**CEP**: 24210-240 _____ **E-mail**:rzanghi@id.uff.br_____

Telefone 1: (21) 2629-5233 _____**Telefone 2**: (_____)_____

DADOS DO BOLSISTA

Nome: Dieison Mariano Farias Rocha

Matrícula: 618038102 **CPF**: 09551245903 **CR**: 7,9

Curso/Departamento/Instituto: Engenharia Elétrica/TEE/Universidade Federal Fluminense

Endereço: Rua dos Frades, n° 5

Bairro: Patronato **Cidade**: São Gonçalo

UF: RJ **CEP**: 24435-220 **E-mail**: dieisonmfr@id.uff.br

Telefone 1: (21) 998776201 **Telefone 2:** (_____)_____

DADOS DO BOLSISTA

Nome: Pedro Victor Rodrigues Veras

Matrícula: 622038083 **CPF**: 13206502799 **CR:** 6,0

Curso/Departamento/Instituto: Engenharia Elétrica/TEE/Universidade Federal Fluminense

Endereço: Rua Doutor Paulo Alves

Bairro: Inga **Cidade**:Niterói

UF: RJ **CEP**: 24219-900 **E-mail:** pedrovictorveras@id.uff.br

Telefone 1: (21) 999289987 **Telefone 2**: (_____)_____

INTRODUÇÃO

A operação dos sistemas elétricos de potência (SEP) é uma tarefa cada vez mais desafiadora, impulsionada pela necessidade de operar dentro de limites estritos de custo e segurança. A crescente complexidade nos SEP demanda análises computacionalmente custosas para garantir sua operação eficiente e confiável. Além disso, a busca por soluções ótimas é frequentemente dificultada pela falta de correlação direta entre os critérios de avaliação e as variáveis de decisão, o que torna a aplicação de métodos tradicionais de otimização desafiadora.

Diante desses desafios, as meta-heurísticas emergem como uma abordagem poderosa, oferecendo estratégias computacionais inspiradas em processos naturais. Essas técnicas permitem uma exploração eficaz do espaço de soluções, adaptando-se aos desafios encontrados.

Um exemplo concreto dessa abordagem é o problema de agendamento de intervenções em redes elétricas (AIRE), tratado como um problema de otimização. Nesse contexto, o uso de meta-heurísticas populacionais oferece uma maneira eficiente de explorar o espaço de soluções em busca de melhores resultados [1].

O objetivo principal deste projeto é o desenvolvimento de um framework de software de otimização em código aberto para aplicações dos SEP em Python, integrando duas bibliotecas do Python, uma com algoritmos de meta-heurísticas (e.g. NiaPy [2], DEAP [3]) e a outra para algoritmos de representação e análise dos SEP (e.g. Pandapower [4]) e da implementação de técnicas de diversificação apresentadas em [1]. Como objetivos secundários, é desejável que o framework desenvolvido seja capaz de:

- Resolver problemas de minimização em engenharia elétrica: Com foco em encontrar a melhor solução para diferentes cenários de otimização.
- Oferecer um ambiente flexível e modular: Permitir a aplicação do framework para diversos problemas e a inclusão de novas metaheurísticas.
- Utilizar bibliotecas de código aberto em Python: Para facilitar a implementação, o desenvolvimento e o compartilhamento do código.
- Validar o desempenho do framework: Com base em *benchmarks* de funções de otimização, como a função Rastrigin, e problemas reais de engenharia elétrica.

Este relatório descreve as atividades executadas durante o período de vigência do projeto. Desde uma revisão bibliográfica para investigação e aprofundamento de meta-heurísticas realizada pelo primeiro bolsista até avaliação das bibliotecas de meta-heurísticas em código aberto disponíveis, a inclusão de um operador de algoritmos evolutivos em uma biblioteca de código aberto e a implementação de um framework para otimização de problemas aplicados ao SEP.

1. METODOLOGIA

A estratégia inicial escolhida pelo orientador foi a de construir uma base teórica bem fundamentada com artigos sobre a área de pesquisa, leitura de bibliografia necessária para o entendimento das meta-heurísticas populacionais e conceitos usados assim como familiaridade com as bibliotecas a serem usadas, seguido de 4 resumos elaborados sobre os temas estudados, e por fim o desenvolvimento de uma nova framework. Para o desenvolvimento da framework, são adotadas as seguintes etapas:

1. Pesquisa sobre bibliotecas de código aberto para otimização usando meta-heurísticas de base populacional inspiradas em Algoritmos Evolutivos.
2. Definição de critérios de avaliação e escolha de uma das bibliotecas identificadas na etapa 1.
3. Implementação de Algoritmo Evolutivo com Elitismo simples na biblioteca escolhida.
4. Implementação de operador de Repopulação com Conjunto Elite na biblioteca escolhida.
5. Integração da biblioteca escolhida com o framework pandaPower.
6. Escolha de caso de uso com problema de otimização.
7. Simulações do framework integrado com o caso de uso escolhido.
8. Elaboração de Manual de Utilização do Framework e disponibilização em repositório.

1.1. RESUMOS ELABORADOS PARA FUNDAMENTAÇÃO TEÓRICA

1.1.1. TÉCNICAS DE OTIMIZAÇÃO COM META-HEURÍSTICAS POPULACIONAIS APLICADAS AOS SEP

Em um mundo cada vez mais complexo, a busca por soluções eficientes para problemas desafiadores se torna fundamental. Na engenharia elétrica, a otimização de sistemas, como redes de energia, se torna uma necessidade para garantir a eficiência, segurança e sustentabilidade.

Heurísticas são processos cognitivos empregados em decisões não racionais sendo definidas como estratégias que ignoram parte da informação com o objetivo de tornar a escolha mais fácil e rápida, em muitos casos espera-se que alcancem os valores ótimos da solução de problemas especialmente nas ocasiões em que partem de uma solução próxima do valor inicial ótimo.

Segundo UFCE[5], a partir de [6]:

> "As metaheurísticas são um conjunto de conceitos os quais podem ser usados para definir métodos heurísticos que podem ser aplicados a uma ampla gama de diferentes problemas. Portanto, trata-se de um conjunto de regras que podem servir de base para o projeto em modelagem computacional. Uma metaheurística pode ser escrito em um algoritmo modular

que pode ser aplicado a diferentes problemas de otimização com poucas modificações a serem realizadas na adaptação a um problema específico"

Em UFCE [5], encontram-se alguns conceitos importantes sobre metaheurísticas:

- **Vizinhança de uma solução:** São um conjunto de soluções próximas a uma solução específica, uma solução (variáveis de decisão) se refere a um conjunto de soluções próximas que podem ser alcançadas a partir da solução atual, e a metaheurística busca encontrar a solução ótima global, que possui o valor mínimo (0.0 para Rastrigin), explorando essas vizinhanças.
- **Problema de otimização**: Como representar um problema, a cada solução viável, deve ser relacionado um novo valor da função aptidão para minimização ou maximização.
- **Solução vizinha:** Pode ser uma alternativa próxima à solução atual, diferindo apenas por uma modificação dos valores próximos da solução global.
- **Perturbação:** É o mecanismo que gera soluções vizinhas a partir da solução atual, introduzindo pequenas alterações.
- **Busca local:** Explora sistematicamente as soluções vizinhas, guiada pela função objetivo, em busca de uma solução melhor;
- **Problemas com restrições:** É possível implementar algoritmos com penalidade aplicada a cada violação de restrições no valor da função objetivo.

As metaheurísticas, como ferramentas inteligentes de otimização, oferecem soluções de problemas complexos que desafiam métodos tradicionais. Neste trabalho, é explorado o uso de algoritmos evolutivos, uma classe de metaheurísticas, para construir um framework em Python, focado em otimizar problemas aplicados em engenharia elétrica em problemas de minimização.

Em [7], onde foi utilizado Otimização por Enxame de Partículas para otimizar os parâmetros do Compensador Série Síncrono Estático (CSSE) para melhor o limite de maximização de carregamento. O algoritmo é motivado a partir do comportamento de um bando de pássaros ou um cardume de peixes em busca de seu alimento. O movimento de cada indivíduo é influenciado pela velocidade anterior, pela própria experiência de cada membro e pela experiência dos outros indivíduos. Cada indivíduo é denominado uma partícula. Os parâmetros de um CSSE são otimizados para minimizar as perdas ativas e reativas do sistema.

Já em [8], a otimização seno-cosseno é usada para projetar os parâmetros de um estabilizador de sistema de potência para amortecer oscilações eletromecânicas em uma única máquina conectada a um grande sistema de potência.

Além disso, também existem artigos que comparam meta-heurísticas populacionais diferentes sendo usadas para o mesmo problema como em [9] onde são comparados algoritmos como Enxame de Partículas e Otimização Jaya, além da construção de um novo algoritmo que usa metaheurísticas

populacionais, para ser aplicado na previsão da produção de energia de um sistema de energia solar fotovoltaica e de turbina eólica.

1.1.2. ESTRATÉGIAS DE DIVERSIFICAÇÃO EM META-HEURÍSTICAS POPULACIONAIS

O trabalho [1] apresenta a boa parte do entendimento do orientado sobre como funciona uma meta-heurística populacional, especificamente o capítulo 3. Para um melhor entendimento, também foi usado o livro [10] e [11].

Um problema de otimização tem como objetivo obter a melhor solução considerando um conjunto de alternativas potenciais. Esta melhor solução representa a melhor solução possível que, satisfazendo a função objetivo, resolver adequadamente a formulação de otimização. A função objetivo, por sua vez, é aquilo que queremos otimizar no problema. Ela pode ser maximizada ou minimizada escolhendo as variáveis, ou variáveis de decisão, que satisfazem todas as restrições do problema, as variáveis de decisão determinam o valor da função objetivo e o conjunto de todas as soluções constituem o espaço de busca.

Os métodos de otimização por meta-heurísticas consistem de um conjunto geral de regras que podem ser aplicadas para resolver uma variedade de problemas de otimização. Eles coletam todo o conhecimento necessário sobre a estrutura de um problema de otimização usando a informação fornecida por todas as soluções avaliadas durante o processo de otimização. Então, esse conhecimento é empregado para construir novas soluções candidatas.

Nas meta-heurísticas populacionais, a população é definida por um conjunto de indivíduos (agentes) que representam soluções potenciais do problema de otimização. O número de agentes é conhecido como o tamanho da população e a função objetivo é denominada função de aptidão. Em geral, um agente pode ser representado como um vetor do qual os elementos são valores de variáveis de controle do problema de otimização.

As meta-heurísticas populacionais atuam, em geral, da seguinte maneira [6]:

- Codificam-se os indivíduos, onde cada indivíduo é uma possível solução.
- Define-se a função de aptidão.
- Calcula o valor na função de aptidão de cada indivíduo da população.
- Gera-se uma nova geração da população aplicando os três tipos básicos de operadores: seleção, cruzamento e mutação.
- Repita os dois últimos processos até que o algoritmo termine.

Quando o algoritmo termina, a solução ótima é o melhor indivíduo da última geração, isto é, o que possui o melhor valor na função de aptidão, onde o melhor é definido por maximizar ou minimizar a função dada.

A maneira como os indivíduos são representados no problema é de grande importância para a eficiência de uma meta-heurística populacional. Portanto, a primeira decisão a tomar é como representá-los. De acordo com [1], a codificação é um processo importante na solução do problema, pois uma escolha errada pode resultar em perda de informação. Alguns métodos usados são codificação binária, *gray*, real e inteira.

Também em [1], a seleção é o operador responsável por selecionar os indivíduos para a combinação. Os mesmos são escolhidos baseado em seus valores na função de aptidão, dando maior probabilidade de escolha aos indivíduos com melhores valores na função de aptidão. Entretanto, indivíduos piores não devem ser totalmente descartados, pois selecioná-los pode levar à criação de um indivíduo útil para o processo.

Os métodos de seleção diferem na forma como selecionam os melhores indivíduos, assim como se descartam ou não os piores indivíduos. Alguns exemplos são seleção por torneio, seleção por roleta e seleção baseada em classificação. Como citado em [6], "a probabilidade de um indivíduo ser selecionado é proporcional ao seu valor na função de aptidão". Isso significa que quanto melhor for o valor de um indivíduo na função objetivo, maior sua probabilidade de ser escolhido. O problema desse método, antes de aplicar cruzamento e mutação, é que pode resultar em ter uma população sendo todos cópias de um mesmo indivíduo.

O operador de cruzamento, como citado em [1], "tem o propósito de procurar pelo espaço de busca de modo a criar boas sequências cromossômicas e de combinar estas sequências para formar outras ainda melhores". Isto é, o processo de trocar bits ou variáveis de decisão entre dois indivíduos, resultando em um novo indivíduo ou a cópia de um dos dois.

A mutação refere-se a mudança aleatória de um ou mais bits ou genes de um indivíduo que ocorre a cada geração com uma certa probabilidade definida. Como mencionado em [1], a mutação "deve ser aplicada após o cruzamento e tem como objetivo restabelecer genes e sequências cromossômicas perdidas durante o processo evolutivo do algoritmo". Segundo [6], o impacto que a mutação causa é maior quando temos uma geração de indivíduos com valores similares na função de aptidão. Se todos os indivíduos forem iguais, somente a mutação é capaz de causar uma alteração na população. Ao contrário da seleção e do cruzamento, a mutação é um operador unitário uma vez que só é aplicado a um indivíduo.

Os algoritmos genéticos (AG) foram propostos por John Holland [1] e buscam simular o processo de evolução das espécies, trabalhando com uma população de indivíduos no qual cada um destes representa uma solução do problema. Em problemas de otimização, a aptidão (*fitness*) de cada indivíduo simboliza o valor da função objetivo desta solução. Cada gene do indivíduo é armazenado em um valor numérico que representa parte da solução do problema em análise. Com o avanço do uso dos AG e de outros algoritmos bio inspirados como ferramentas de otimização, surge o conceito mais

abrangente de Algoritmos Evolutivos (AE), que amplia o AG original incorporando outras estratégias. Os AE trabalham com populações de soluções e em geral bio inspirados na natureza, usando os mesmos princípios do GA de população, seleção, recombinação e mutação mas com estratégias evolutivas com base na Seleção natural, esses algoritmos trabalham com uma única solução e investigam a sua vizinhança por meios de mecanismo de busca local.

Os campos da computação que utilizam algoritmos bio inspirados e os campos da biologia que tratam dos processos evolutivos articulam relações que permitem o trânsito de ideias e informações nos dois sentidos [1]. Um exemplo deste tipo de interação pode ser encontrado no Livro Homo Deus[12], de Yuval Harari (pág.279):

> [...] O que vai acontecer quando algoritmos nos suplantarem nas ações de lembrar, analisar e reconhecer padrões? A ideia de que os humanos sempre terão uma aptidão exclusiva, além do alcance de algoritmos não conscientes, é uma quimera. A atual resposta da ciência a esse sonho impossível pode ser resumida em três princípios simples:
>
> 1. Organismos são algoritmos. Todo animal — inclusive o Homo sapiens — é uma montagem de algoritmos orgânicos modelada pela seleção natural durante milhões de anos de evolução
> 2. [...]
> 3. Não há razão para pensar que algoritmos orgânicos possam fazer coisas que algoritmos não orgânicos não serão capazes de igualar ou de superar. Enquanto os cálculos continuarem válidos, o que importa se os algoritmos se manifestem em carbono ou em silício?

Com o DEAP [4], é possível construir diferentes tipos de algoritmos evolutivos utilizando conceitos e tipos de dados diferentes para criar uma população inicial e final customizada, com indivíduos de diferentes tipos (fenótipos) e permitindo a escolha de operadores. Estes parâmetros de entrada podem ser lidos de um arquivo JSON.

Um algoritmo de meta-heurística deve ser equipado com duas características principais para a descoberta do ótimo global: diversificação e intensificação. Conforme citado em [1], "a diversificação se refere à exploração ampla do espaço de busca e a intensificação está relacionada com a busca intensa na vizinhança de uma determinada solução, priorizando a experiência acumulada na história do processo de busca". Em outras palavras, diversificação é a pesquisa no espaço de busca e intensificação é a exploração das melhores soluções encontradas.

Na diversificação, regiões não exploradas devem ser visitadas para assegurar que todas as regiões do espaço de busca estão sendo exploradas equilibradamente. Na intensificação, as regiões promissoras são exploradas profundamente, para poder encontrar soluções melhores [1].

Na geração da população inicial, o principal critério a ser tratado é a diversificação. Se a população inicial não for bem diversificada, pode ocorrer uma convergência prematura, isto é, resultando numa solução abaixo do ideal [10]. É devido a isso que existem heurísticas ou algoritmos de meta-heurísticas (e.g. algoritmos gulosos, entre outros) para lidar com a geração de uma população diversificada.

Podemos dividir em quatro categorias [11]: geração aleatória, diversificação sequencial, diversificação paralela e inicialização heurística.

Normalmente, a população inicial é gerada randomicamente. A população pode ser gerada usando números pseudo-aleatórios ou uma sequência de números quasi-aleatórios.

Em diversificação sequencial, as soluções são geradas em sequência de forma que a diversidade seja otimizada. Essa estratégia tem alto custo computacional. No entanto, garante uma boa distribuição da população. Alguns processos de inicialização não requerem calcular o valor de cada indivíduo na função de aptidão [10].

Em diversificação paralela, as soluções de uma população são geradas de forma paralela e independente. Esta estratégia fornece uma boa distribuição aleatória geral da população e não requer qualquer avaliação da função de aptidão [11].

Por fim, em inicialização heurística, qualquer heurística pode ser usada para inicializar a população. Esta estratégia, porém, pode perder diversidade populacional, o que gerará uma convergência prematura da população [11].

1.1.3. MODELAGEM COMPUTACIONAL EFICIENTE PARA ANÁLISES NO SEP

O primeiro orientado teve dificuldades em encontrar artigos no CAPES para este tópico, preferindo assim usar livros voltados ao assunto, mas o orientado não conseguiu tempo para ler as partes importantes neles.

1.1.4. PANDAPOWER, NIAPY e DEAP

Pandapower é uma biblioteca de análise de sistemas de energia para Python destinada à automação de análises estáticas e quase estáticas e otimização de sistemas de energia balanceados, de acordo com [2]. Como seu foco é em análise estático dos sistemas de potência trifásicos, isso permite a análise de sistemas de transmissão e subtransmissão.

NiaPy, por sua vez, é uma biblioteca para a construção de algoritmos baseados na natureza. A biblioteca já vem inclusa alguns algoritmos implementados, como Otimização por Enxame de Partículas, Colônia Artificial de Abelhas, entre outros [3].

Uma das vantagens da biblioteca é sua facilidade em integrar novos algoritmos nela, fazendo com que o usuário não dependa apenas dos algoritmos que já foram implementados pelos desenvolvedores da biblioteca. Além disso, isso permite aos pesquisadores uma facilidade em comparar diferentes algoritmos sendo aplicado a um mesmo problema.

Devido a falta de artigos referenciando ambas as bibliotecas, o orientado iria se familiarizar com elas estudando ambas as bibliotecas tanto por tutoriais no site de ambas quanto lendo seu código fonte, mas ele nunca chegou a conseguir começar a fazer isso.

A escolha de uma biblioteca para um projeto de otimização é crucial, e uma análise de requisitos bem definida é essencial para tomar a decisão correta, visto que é necessário uma biblioteca que seja de código aberto e que tenha funcionalidades de Algoritmos Evolutivos com parâmetros Genéticos. No caso de problemas de Sistemas Elétricos de Potência (SEP), a biblioteca DEAP [4] surge como uma candidata.

O algoritmo evolutivo foi construído através de uma biblioteca em Python específica para estas classes de algoritmos. A biblioteca DEAP [4] (*Distributed Evolutionary Algorithms in Python*) é uma ferramenta de código aberto para otimização evolutiva em Python. Ela fornece uma estrutura flexível e modular para criar e implementar uma ampla gama de algoritmos evolutivos, incluindo algoritmos genéticos, programação genética e estratégias de evolução, tornando-o adequado para a otimização de sistemas complexos como os encontrados em SEP. Com a DEAP [4], é possível definir tipos de dados personalizados para representar seus indivíduos, criar operadores genéticos para evolução, definir funções de avaliação para medir o desempenho dos indivíduos e executar os algoritmos evolutivos de forma distribuída, facilitando a sua aplicação para problemas de grande escala, comuns nos SEP. A DEAP [4] simplifica o desenvolvimento de algoritmos evolutivos, permitindo manter o foco na lógica específica do problema, reduzindo o esforço com os detalhes de implementação. A biblioteca fornece ferramentas para criar, manipular e evoluir populações de indivíduos, definir funções de aptidão (*fitness*), e executar os algoritmos evolutivos. A documentação abrangente e uma comunidade ativa garantem suporte e aprendizado contínuos durante o desenvolvimento do projeto.

1.2. IMPLEMENTAÇÃO DO FRAMEWORK

1.2.1. PESQUISA SOBRE BIBLIOTECAS DE CÓDIGO ABERTO PARA OTIMIZAÇÃO

O primeiro bolsista realizou pesquisa de referências bibliográficas e elaborou os resumos que constam neste relatório. O Segundo bolsista, avançou na avaliação e escolha de uma biblioteca de código aberto que possui suporte a Algoritmos Evolutivos.

Após uma pesquisa em sites de bibliotecas de otimização, foram identificadas as bibliotecas candidatas descritas na Tabela 1.

Tabela 1- Bibliotecas de Otimização por Meta-heurísticas em Código Aberto com Algoritmos Evolutivos

Bibliotecas	**DEAP [4]**	**PyGMO**	**Pyevolve**	**PyParadiseo [13]**
URLs	https://deap.readthedocs.io/en/master/	https://esa.github.io/pygmo/index.html	https://pyevolve.sourceforge.net/0_6rc1/	https://paradiseo.gitlabpages.inria.fr/pyparadiseo/
Comunidade ativa	**SIM**	**NÃO**	**NÃO**	**SIM**
Número de metaheurísticas implementadas	**20+**	**50+**	**5+**	**10+**
Tipo de licença	**BSD**	**LGPL**	**LGPL**	**GPLv3**
Suporte a operadores	**SIM**	**SIM**	**SIM**	**SIM**
Foco principal	Computação evolutiva para prototipagem rápida e teste de ideias. Procura tornar os algoritmos explícitos e as estruturas de dados transparentes. Funciona em perfeita harmonia com mecanismos de paralelização como multiprocessamento.	Fornece um grande número de problemas de otimização e algoritmos sob a mesma abstração de paralelização construída em torno do paradigma *modelo de ilha generalizada*. Os algoritmos disponíveis são todos a**utomaticamente paralelizados** (abordagem de granulação grossa, de forma assíncrona).	Foco em otimização em algoritmos genéticos e possui suporte de visualização	Fornece funções de fábrica que permitem aos usuários instanciar componentes e algoritmos ParadisEO (originalmente escrito em C++).

Fonte: Autoria Própria

1.2.2. DEFINIÇÃO DE CRITÉRIOS DE AVALIAÇÃO E ESCOLHA DE UMA DAS BIBLIOTECAS

Para escolha da biblioteca foram adotados os seguintes critérios:

- Programação em linguagem Python – devido à facilidade de reuso do framework
- Comunidade ativa – indica que a biblioteca está em constante aperfeiçoamento e correção de bugs.

- Tipo de licença – deve garantir reuso e modificação do código original para fins científicos e acadêmicos
- Suporte a implementação de novos operadores – para permitir incluir facilmente os operadores apresentados em [1].

Considerando os critérios adotados, foi escolhida a biblioteca DEAP [4] sendo elegível também a biblioteca PyParadiseo[13]. No entanto, a escolha da DEAP[4] se deu pelo fato da biblioteca ter sido desenvolvida originalmente em Python, enquanto a PyParadiseo[13] ter sido portada de C++ para Python.

1.2.3. IMPLEMENTAÇÃO DE ALGORITMO EVOLUTIVO COM ELITISMO SIMPLES NA BIBLIOTECA ESCOLHIDA

O segundo bolsista implementou um Algoritmo Evolutivo na biblioteca DEAP [4]. Como configuração inicial, foram utilizados os operadores de seleção por Torneio, cruzamento de 2 pontos e mutação gaussiana considerando um elitismo simples que preserva apenas o indivíduo com a melhor aptidão entre as gerações. O código foi implementado em Python usando a plataforma Google Colab.

Para o problema de otimização, foi selecionada a função de benchmark *Rastrigin[14]* que pode ser escalonada para um número qualquer de variáveis de decisão reais, com ótimo global conhecido onde toda a validação do algoritmo evolutivo será feita com esta função objetivo.

1.2.4. IMPLEMENTAÇÃO DE OPERADOR DE REPOPULAÇÃO COM CONJUNTO ELITE

O segundo bolsista implementou um operador proposto em [1] para fomentar a diversidade em meta-heurísticas evolutivas. A Repopulação com Conjunto Elite (RCE) [1] é uma estratégia utilizada em algoritmos evolutivos para promover a diversidade e a inovação dentro da população ao longo das gerações. Essa técnica serve para evitar a convergência prematura da população para soluções globais e ótimas e garantir uma exploração eficiente do espaço de busca em busca de soluções de alta qualidade.

O funcionamento do RCE pode ser dividido em 3 etapas. Primeiramente, é importante destacar que o processo de RCE ocorre em ciclos, onde a população é reinicializada após um determinado número de gerações. Durante esse processo, um grupo seleto de indivíduos de melhor aptidão, conhecido como conjunto elite, é preservado e reintroduzido na nova população, enquanto os demais indivíduos são substituídos.

A seleção dos membros do conjunto elite é feita com base na qualidade dos indivíduos em relação ao melhor da geração anterior. Isso é realizado em duas etapas distintas. Na primeira etapa, uma lista restrita de candidatos ao conjunto elite é formada, selecionando os melhores indivíduos em termos

de qualidade em relação ao melhor da geração anterior. Os candidatos escolhidos possuem uma certa diversidade e representatividade na população.

Na segunda etapa, a lista de candidatos é refinada para eliminar redundâncias e manter a diversidade do conjunto elite. Isso é feito comparando-se cada candidato com os outros através de suas variáveis de decisão com um critério de diferença entre eles e descartando aqueles que representam soluções muito semelhantes. Essa abordagem garante que o conjunto elite contenha uma variedade de soluções de alta qualidade e diversificadas, capazes de promover a inovação na nova população.

Na Terceira etapa, cada geração de novos indivíduos aleatórios é feita após a reinicialização da população, utilizando uma nova semente para garantir a introdução de diversidade e evitar possíveis erros de distribuição da população inicial.

Para construir o framework, seguiu-se um processo similar ao que um engenheiro de software usaria para criar um sistema robusto, modular e adaptável.

Adaptado de Len Bass, Paul Clements e Rick Kazman em "Software Architecture in Practice”

“Sistema robusto é aquele que é capaz de lidar com erros e falhas de forma eficiente, sem comprometer a sua funcionalidade ou a integridade dos dados, ele é capaz de se lidar com erros, continuar funcionando em condições adversas e evitar que erros menores se propaguem e causem falhas maiores, sistema modular é aquele que é dividido em módulos independentes e interconectados, cada módulo tem uma função específica e bem definida, o que facilita o desenvolvimento, a manutenção e a reutilização do código e um sistema adaptável é aquele que pode ser facilmente modificado e estendido para atender a novas necessidades e requisitos onde pode ser flexível e permite a adição de novas funcionalidades sem comprometer a sua estrutura central.”

Um framework sendo robusto, modular e adaptável é capaz de garantir a qualidade, a manutenção e a escalabilidade do sistema. Ao seguir os princípios de POO como abstração para separar classes e inclusão de banco de dados, o software pode atender os levantamento de requisitos dos usuários e que possa ser facilmente adaptado às mudanças futuras pela própria comunidade sendo escrito em uma linguagem de programação open source como o *Python*.

1.2.5. INTEGRAÇÃO DA BIBLIOTECA ESCOLHIDA COM PANDAPOWER

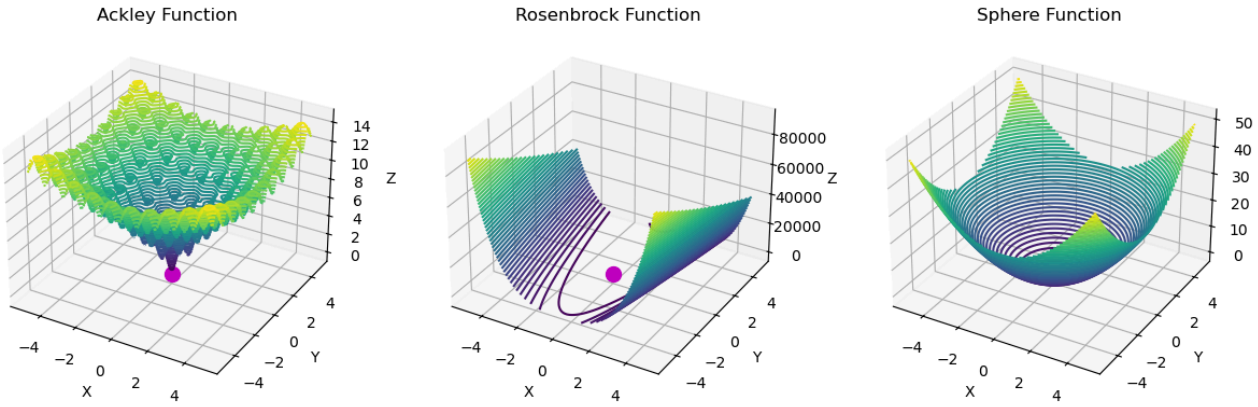
Esta etapa da metodologia não foi executada e será direcionada para um projeto subsequente. Maior esforço e dedicação foi empregado na implementação do framework, melhorando sua usabilidade.

1.2.6. ESCOLHA DO CASO DE USO COM PROBLEMA DE OTIMIZAÇÃO

Como foi redirecionado o esforço para a usabilidade do framework, decidiu-se utilizar uma função de *benchmark*[14] para testar o funcionamento da implementação.

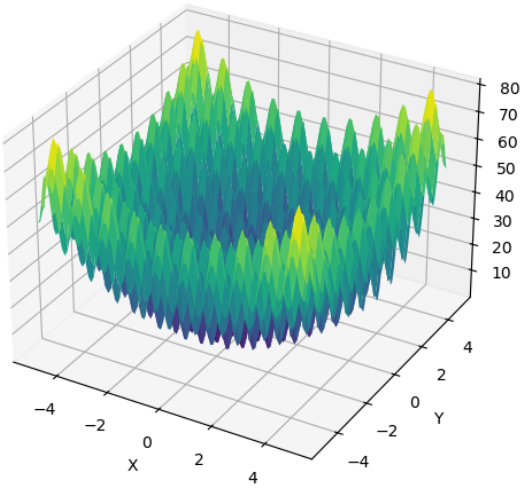
Com o desenvolvimento crescente de técnicas e algoritmos de otimização, tornou-se necessário criar funções que pudessem ser utilizadas como problemas de otimização para testes de eficácia e eficiência da implementação de algoritmos desses algoritmos. Estas funções representam matematicamente problemas de otimização difíceis. Algumas destas funções podem ser encontradas em [14]. Neste trabalho foram investigadas as funções **Rastrigin, Sphere, Ackley e Rosenbrock** que admitem 2 ou mais variáveis. Nas figuras 1 e 2 são ilustrados gráficos com estas funções com 2 variáveis de decisão.

Figura 1 - Gráficos das funções Ackley e Rosenbrock e Sphere de duas variáveis



Fonte: Autoria própria

Figura 2 - Gráfico de Benchmarking Rastrigin para duas variáveis.



Fonte: Autoria própria

A função de minimização escolhida como *benchmarking* da otimização foi a *Rastrigin* [14], que tem seu mínimo global em 0,0 para valores de variáveis de decisão iguais a 0,0. A função Rastrigin é

uma função não convexa usada como um problema de teste de desempenho para algoritmos de otimização. Essa função é um exemplo típico de função multimodal não linear. Encontrar o mínimo desta função é um problema bastante difícil devido ao seu grande espaço de busca e seu grande número de mínimos locais.

1.2.7. SIMULAÇÕES DO FRAMEWORK

Utilizando as funções de otimização [14] foi executado o framework variando os 6 parâmetros do algoritmo de otimização implementado com objetivo de avaliar a eficácia e eficiência da implementação. Foi usada a função **Rastrigin** [14] como função objetivo de minimização e escolhidas 10 variáveis de decisão do tipo *float*, com limites de -5,12 a +5,12 (conforme [14]), usando operadores de seleção por torneio com 3 indivíduos, elitismo simples entre gerações. Utilizando uma funcionalidade implementada durante este trabalho, esses parâmetros foram obtidos de um arquivo JSON de configuração.

Para considerar a natureza estocástica dos algoritmos implementados, nas simulações, foram realizados testes executando o algoritmo 20 vezes com cada conjunto de parâmetros. Cada parâmetro escolhido foi variado individualmente. Os valores escolhidos para cada variação do conjunto de parâmetros são descritos na Tabela 2. Para o AE, foram escolhidos dois parâmetros comumente ajustados na literatura associada à taxa de aplicação dos operadores de mutação (TOM) e de cruzamento (TOC). Para a estratégia RCE[1], foram escolhidos os parâmetros definidos em [1]:

- O número de gerações para executar o RCE, como um percentual do número total (PNT) de gerações definido pelo AE.
- Critério de seleção por qualidade (CSQ), como um limite percentual da aptidão do melhor indivíduo.
- Critério de diferença mínima (CDM) entre valores de variáveis de decisão, como um valor absoluto.
- Número mínimo de variáveis (NMV) de decisão diferentes entre soluções.

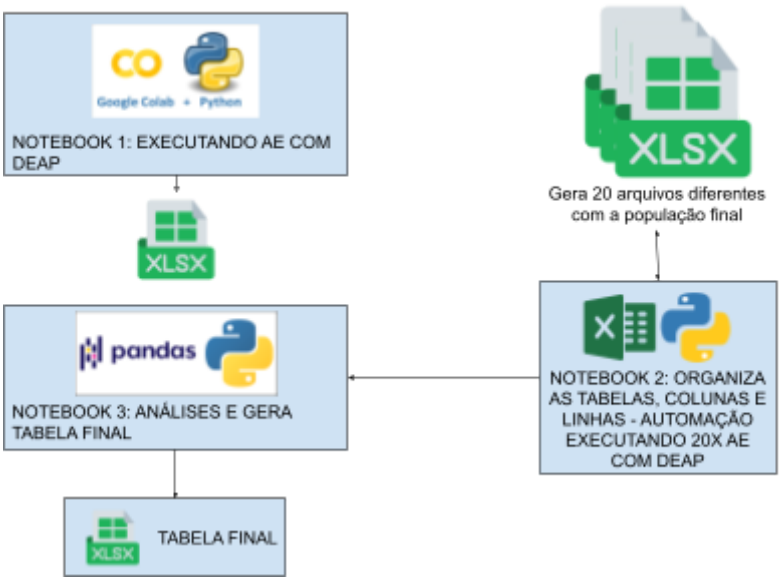
Tabela 2: Valores de parâmetros que foram testados nas simulações

Taxa do operador		**RCE**			
Crossover	**Mutação**	**PNT**	**CSQ**	**CDM**	**NMV(delta)**
0,2	**0,05**	**5%**	**30%**	**1**	**0,1**
0,4	**0,1**	**10%**	**60%**	**2**	**0,5**
0,6	**0,15**	**20%**	**100%**	**3**	**1**
0,9	**0,2**	**50%**	**300%**	**5**	**3**

Fonte: Autoria própria

Foram criados 3 Jupyter Notebooks com diferentes scripts para automação do uso do framework. Um notebook foi feito para ter as Classes *Setup*, *DashBoard* e Algoritmo Evolutivo RCE com seu funcionamento básico na função *Run* e gerando seus resultados para apenas uma execução do algoritmo. O seu resultado retorna a população final gerada, suas estatísticas, menor geração da solução encontrada, melhor aptidão e melhores indivíduos com suas variáveis de decisão. Este Notebook1 pode ser utilizado individualmente para usar o framework em apenas uma execução. Uma variação desse notebook (Notebook2), foi adaptado para executar a função *main* 20 vezes coletando os dados de cada execução do algoritmo evolutivo. Os dados coletados vão para 20 planilhas diferentes com seus respectivos resultados. Um terceiro notebook (Notebook3) foi criado para analisar esses dados. Com todas as planilhas dentro de um diretório, o script varre todos os arquivos dessa pasta e agrupa os valores em um único *DataFrame* da biblioteca *Pandas* do Python. Dessa forma é possível fazer análises e verificar as mudanças dos valores de entrada no arquivo JSON e de cada resultado. Com alguns ajustes manuais usando a biblioteca *Python Openpyxl*, é possível juntar em uma tabela final os valores fixos com os valores que estão variando a partir do arquivo JSON. Os melhores resultados foram separados para uma tabela resumo onde também foi calculado o CV da melhor aptidão e da menor geração em que foi encontrada cada solução.

Figura 3: Ilustração da automação das simulações



Fonte: Autoria própria

Os resultados das simulações descritas nesta seção são apresentados nas Tabelas de 4 a 9 presentes na seção RESULTADOS deste documento.

Para determinação de um valor base ou *default* para os parâmetros do algoritmo de otimização foram realizados testes e simulações e foi adotado o critério de escolha do conjunto de parâmetros

correspondente à melhor solução encontrada. Para as simulações com diferentes parâmetros vindo do arquivo JSON, foram alterados cada um dos parâmetros individualmente, mantendo os demais com os valores *default*. Os valores padrão podem ser encontrados na Tabela 3.

Tabela 3: Valores default utilizados nos testes de validação do framework

Indivíduo		RCE			
Crossover	Mutação	Percentual Elite	Número Variável Decisão diferentes	Delta	Quantidade Gerações para Aplicar RCE
60%	15%	30%	1	0,50	20

Fonte: Autoria própria

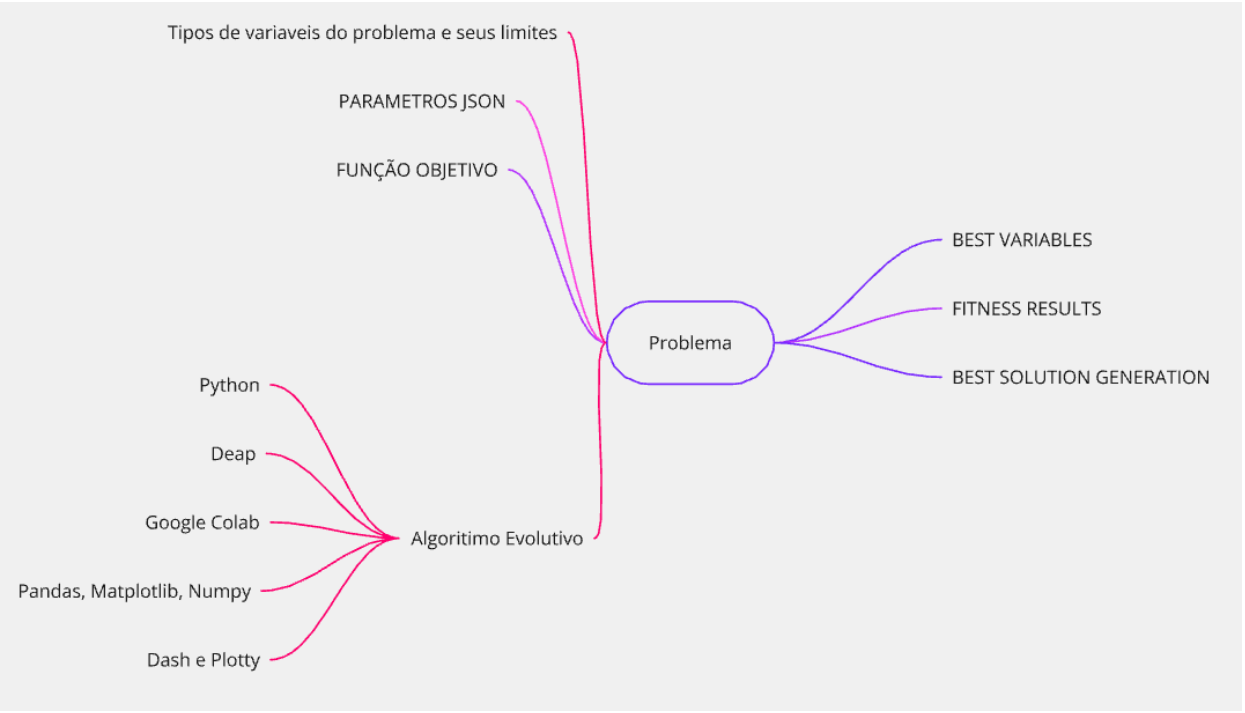
1.2.8. ELABORAÇÃO DO MANUAL DE UTILIZAÇÃO DO FRAMEWORK E DISPONIBILIZAÇÃO EM REPOSITÓRIO

Segundo [5]

> *" Espera-se que, em muitos casos, as heurísticas alcancem os valores ótimos da solução de problemas NP-Árduos ou mais complexos, especialmente nas ocasiões em que partem de uma solução próxima do valor ótimo."*

Existem diversos problemas em engenharia que podem ser resolvidos utilizando metaheurísticas. O framework implementado neste trabalho tem foco na facilidade de uso e produtividade. Para problemas de otimização em Sistemas Elétricos de Potência (SEP), o framework pode ser utilizado para abordar diversos aspectos, como minimização do custo de produção de energia, reduzir as perdas nas linhas de transmissão ou encontrar a melhor configuração para um sistema de armazenamento de energia. Na Figura 4, são apresentadas relações entre as tecnologias empregadas, sua aplicação na resolução de problemas e alguns conceitos relacionados à otimização e algoritmos evolutivos.

Figura 4: Mapa Mental - Problema



Fonte:Autoria Própria

Alguns possíveis problemas de otimização que podem ser resolvidos por este framework são listados a seguir:

- Custo e Consumo de Energia Elétrica residencial.
- Cálculo da eficiência de uma Bateria de lítio.
- Controle inteligente acionando alarmes com tipos de circuitos diferentes para segurança.

Foi utilizada a linguagem *Python* por sua facilidade de uso, grande variedade de bibliotecas e comunidade ativa de desenvolvedores. As bibliotecas principais utilizadas na implementação do framework foram:

- DEAP: Biblioteca de código aberto para algoritmos evolutivos.
- NumPy: Biblioteca para cálculos numéricos e manipulação de arrays.
- SciPy: Biblioteca com funções matemáticas e científicas.
- Pandas: Biblioteca para manipulação e análise de dados.
- MatplotLib:: Biblioteca para gerar gráficos matemáticos

A documentação oficial do *DEAP[4]* demonstra como utilizar a biblioteca. Este exemplo está ilustrado no Código 1, que mostra como criar os indivíduos e sua população, registrando seus atributos

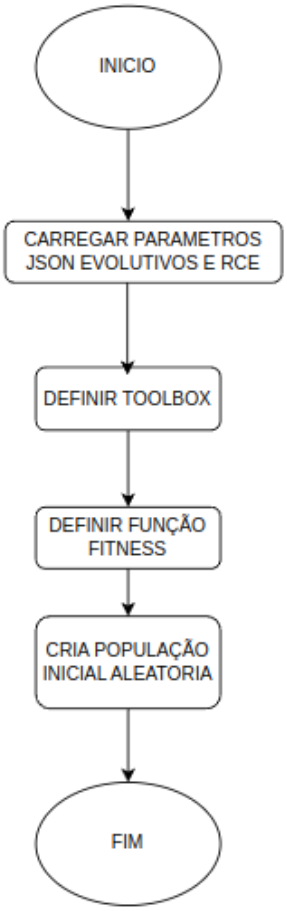
genéticos com ajuda de um operador *Toolbox* fornecido pela biblioteca. Este operador oferece diversos tipos de operadores. Por 40 gerações, o algoritmo evolutivo é executado e retorna os 10 melhores resultados de indivíduos na população final.

Código 1: Exemplo básico da documentação do DEAP

```python
import random
from deap import creator, base, tools, algorithms

creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()

toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=100)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

def evalOneMax(individual):
    return sum(individual),

toolbox.register("evaluate", evalOneMax)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

population = toolbox.population(n=300)

NGEN=40
for gen in range(NGEN):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5, mutpb=0.1)
    fits = toolbox.map(toolbox.evaluate, offspring)
    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit
    population = toolbox.select(offspring, k=len(population))

top10 = tools.selBest(population, k=10)
```

A biblioteca do *DEAP* tem uma forma de modificação personalizada de um algoritmo genético e evolutivo com o uso da ferramenta *ToolBox*. Para acelerar a criação de indivíduos e sua população foi criada uma classe Setup que tem o objetivo de abstrair o usuário utilizando o framework para gerar os indivíduos e populações personalizadas. O framework utiliza como dados de entrada um *array* de variáveis de decisão e uma função objetivo. A classe *Setup* ilustrada na Figura 5, configura o framework com o tipo das variáveis de decisão do indivíduo, sua população inicial e os parâmetros do AG vindo de um arquivo JSON. Dessa forma, é possível implementar problemas com um conjunto de variáveis de decisão de tipos específicos e funções objetivo particulares.

Figura 5 - Fluxograma classe Setup



Fonte: Autoria própria

1.2.8.1. O ALGORITMO EVOLUTIVO

Foi implementado um algoritmo evolutivo baseado na biblioteca *DEAP[4]* em linguagem *Python*, seguindo os passos principais:

-Definição da Função Fitness: Foi criada uma função que avalia a qualidade de uma solução candidata, levando em conta os parâmetros e restrições do problema, foi usado como padrão a Rastrigin[14].

-Criação da População Inicial: Um conjunto inicial de soluções candidatas, representando possíveis configurações do sistema elétrico.

-Seleção de Indivíduos: Os indivíduos mais "aptos" da população, com base na função fitness, são preservados para a próxima geração para continuar no processo evolutivo.

-Cruzamento: Características de indivíduos são combinados e selecionados para gerar novas soluções, com o objetivo de encontrar combinações melhores.

-**Mutação:** Pequenas alterações aleatórias nas soluções, com o objetivo de permitir maior variabilidade genética na população, impedindo que a busca fique estagnada em um mínimo local.

-**Convergência:** O processo iterativo continua até que o algoritmo atinja um critério de parada, como um número máximo de gerações ou uma solução com qualidade suficiente.

O trecho de Código 2 mostra o processo do algoritmo evolutivo implementado. O loop principal executa interativamente as etapas de seleção, cruzamento, mutação e avaliação de *fitness* da população, aplicando a estratégia RCE em intervalos definidos e registrando as estatísticas no *logbook* a cada geração.

Foram criadas algumas funções como sub rotinas para deixar o código mais legível:

- **self.avaliarFitnessIndividuos:** Calcula e atribui o valor de fitness para cada indivíduo da população.
- **self.checkDecisionVariablesAndFitnessFunction:** Verifica e configura as variáveis de decisão e a função de fitness para o problema de otimização.
- **self.aplicar_RCE:** Aplica a estratégia de Repopulação com Conjunto Elite (RCE) para manter a diversidade e melhorar a qualidade da população.
- **self.elitismoSimples:** Preserva o melhor indivíduo de cada geração para a próxima geração.
- **self.registrarDados:** Armazena os dados relevantes da geração atual, como as melhores soluções e o valor de fitness.

Código 2: Loop Algoritmo Evolutivo usando DEAP

Python

```
#! Main LOOP
    def run(self, RCE=False, decision_variables=None, fitness_function=None, num_pop=0):

        if self.setup.DADOS_ENTRADA:
            population = [self.POPULATION, self.POP_OPTIMIZATION]
        else:
            population = [self.POPULATION]

        # Avaliar o fitness da população inicial
        self.avaliarFitnessIndividuos(population[num_pop])

        # Selecionando as variaveis de decisao e afuncao objetivo
        self.checkDecisionVariablesAndFitnessFunction(
            decision_variables, fitness_function
        )

        #! Loop principal através das gerações
        for current_generation in range(self.setup.NGEN):
```

```
            # Selecionar os indivíduos para reprodução
            offspring = self.setup.toolbox.select(
                population[num_pop], k=len(population[num_pop])
            )

            # Clone the selected individuals
            offspring = [self.setup.toolbox.clone(ind) for ind in offspring]

            # Aplicar crossover
            for child1, child2 in zip(offspring[::2], offspring[1::2]):
                if random.random() < self.setup.CXPB:
                    self.setup.toolbox.mate(child1, child2)
                    del child1.fitness.values
                    del child2.fitness.values

            # Aplicar mutação
            for mutant in offspring:
                if random.random() < self.setup.MUTPB:
                    self.setup.toolbox.mutate(mutant)
                    del mutant.fitness.values

            #  Avaliar o fitness dos novos indivíduos
            invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
            fitnesses = map(self.setup.toolbox.evaluate, invalid_ind)
            for ind, fit in zip(invalid_ind, fitnesses):
                ind.fitness.values = [fit]

            #! Aplicar RCE
            if RCE and ((current_generation + 1) % self.setup.num_repopulation == 0):
                #! copia pop aleatória modificada retornada para pop atual
                new_population = self.aplicar_RCE(
                    current_generation + 1, population[num_pop]
                )
                population[num_pop][:] = new_population
            else:
                population[num_pop][:] = offspring

            # Registrar estatísticas no logbook
            self.elitismoSimples(population[num_pop])
            self.registrarDados(current_generation)
            record = self.stats.compile(population[num_pop])
            self.logbook.record(gen=current_generation, **record)

        # Retornar população final, logbook e elite
        return population[num_pop], self.logbook, self.hof[0]
```

Classes foram usadas para separar funcionalidades, onde o usuário se preocupa apenas em formular seu problema por via de parâmetros. Para funcionar o algoritmo, é preciso instanciar as classes e utilizar a leitura dos parâmetros do arquivo JSON passando para uma variável onde é passada como parâmetro para a classe Setup. É utilizada a função run para executar o loop onde seus resultados são a variável de decisão do melhor indivíduo da população (solução do problema), função aptidão e a geração onde foi encontrada essa solução.

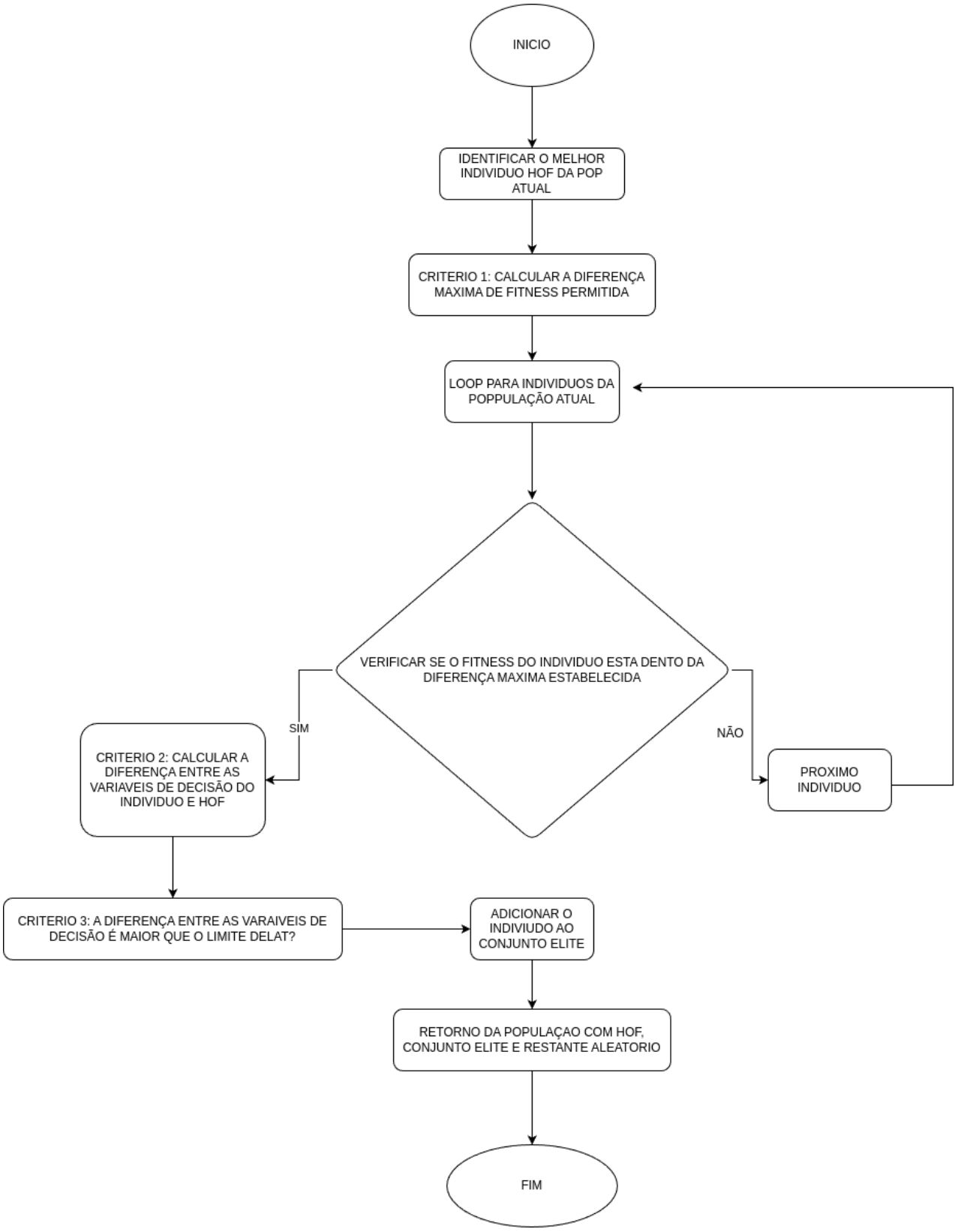
Foi criada uma classe DataExploration que cuida exclusivamente da parte visual do framework onde é construído seus gráficos *Matplotlib* e *Plotly* em relação aos dados obtidos do loop de evolução. Foi realizada implementação preliminar com salvamento dos dados em um banco de dados SQL. Essa classe será utilizada na fase futura do projeto onde será desenvolvida interface gráfica moderna e interativa usando *DASH Html* em *Python* onde o usuário pode ter uma melhor visualização e experiência com os dados sendo modificados em tempo real com gráficos dinâmicos e inteligentes com IA.

1.2.8.2. A ESTRATÉGIA DE DIVERSIFICAÇÃO RCE

Foi implementada a Estratégia de Repopulação Conjunto Elite (RCE) [1] para melhorar a capacidade de exploração do algoritmo evolutivo e evitar convergência prematura. A Figura 6 ilustra os passos dessa estratégia em um fluxograma simplificado. Nesta implementação HOF representa o nome dado pela biblioteca DEAP para o indivíduo elite, com a melhor aptidão encontrada. O RCE:

- **Mantém um conjunto Elite:** Seleciona e preserva os indivíduos mais aptos da população, garantindo a retenção de soluções de alta qualidade.
- **Renova a População Periodicamente:** Gera novas soluções aleatórias, introduzindo diversidade no espaço de busca.
- **Combina Diversidade e Qualidade:** Aumenta a probabilidade de encontrar uma solução globalmente ótima.

Figura 6 - Fluxograma Estratégia Conjunto Elite



Fonte: Adaptado de [15]

1.2.8.3. EXEMPLO DE USO DO FRAMEWORK

O usuário do framework encontrará na pasta compartilhada (neste link) três arquivos com extensão jupyter notebook que podem ser abertos diretamente no Google Colab. O Notebook 1 pode ser utilizado para apenas uma execução do AE. Para este fim, o usuário deverá seguir os seguintes passos:

1) Crie um arquivo chamado `*parameters.json*`.

2) Pegue o exemplo de parâmetros no arquivo localizado neste link.

3) Copie e cole esses valores no arquivo de parâmetros criado.

4) Adapte a função objetivo e as variáveis de decisão para o seu problema de otimização. Crie um array multidimensional de valores float ou int e crie uma função em Python que represente o problema. No trecho de Código 3, é ilustrada a definição de um array `ind1` para as variáveis de decisão e uma função `evaluate` que representa a função objetivo.

Código 3: Exemplo de indivíduo e função objetivo

```python
ind1 = [1,2,3,4,5,6,7,8,9,10]  # Exemplo de individuo de tamanho 10

def evaluate(individual):
        """Função objetivo do problema """
        a = sum(individual)
        b = len(individual)
        return a / b
```

5) No trecho de Código 4, é ilustrado o funcionamento ao instanciar os objetos criados das três classes do framework. Neste exemplo são utilizados o array `ind1` e a função `evaluate`, criados anteriormente. Ao executar a função run, o usuário escolhe se deseja usar a estratégia RCE [1]. Esta função retorna a população final gerada e o melhor indivíduo da geração, gerando seu gráfico com esses mesmos parâmetros.

Código 4: Código Main para execução do framework

```python
if __name__ == "__main__":
    # Instancia dos objetos
    setup = Setup(params)
    alg = AlgoritimoEvolutivoRCE(setup)
    data_visual = DataExploration()

 # Função que executa o algoritmo evolutivo
    pop_with_repopulation, logbook_with_repopulation, best_variables = alg.run(
        RCE=True,
        fitness_function=evaluate, # Nome da função objetivo do problema
        decision_variables=(ind1), # Nome do array das variaveis de decisão
    )
    print("\n\nEvolução concluída  - 100%")

    # Visualização dos resultados
    data_visual.show_rastrigin_benchmark(logbook_with_repopulation, best_variables)

    best_solution_generation, best_solution_variables, best_fitness = data_visual.visualize(
        logbook_with_repopulation, pop_with_repopulation, repopulation=True
    )
```

6) Execute todas as células desse Notebook.

7) Para obter resultados e gráficos diferentes, modifique os parâmetros evolutivos do arquivo JSON, salve e execute novamente.

2. RESULTADOS

Os algoritmos implementados em Python na plataforma Google Colab possuem as seguintes funcionalidades:

- **Leitura da Programação Inicial do Usuário**: O algoritmo começa lendo a programação inicial fornecida pelo usuário, que pode incluir os parâmetros do algoritmo evolutivo (AE) de um arquivo JSON e outras configurações relevantes. O Código 5 mostra um exemplo de uma função em Python criada para ler um arquivo JSON e passada para uma variável onde será usada como parâmetro para Classe Setup.

Código 5: Leitura de parâmetros

```python
import json
def load_params(file_path):
    with open(file_path, "r") as file:
        params = json.load(file)
    return params

params = load_params(
    r"/content/drive/MyDrive/Engenharia Elétrica /assets/parameters.json"
)
```

- **Leitura dos Parâmetros do AE do Usuário**: Em seguida, o algoritmo lê os parâmetros específicos do algoritmo evolutivo definidos pelo usuário, como o número máximo de repopulações e outras configurações relacionadas ao processo evolutivo. No Código 6, são mostrados os valores *default* utilizados da Classe *Setup* pelo framework. O framework permite que o usuário realize a chamada passando os parâmetros que customizam o código do algoritmo evolutivo. Para características de seu problema específico, como variáveis de decisão e função objetivo é feita uma customização através do método run, explicado na seção "Exemplo de Uso do Framework".

Código 6: Class Setup

```python
class Setup:
    def __init__(self, params, dadosEntrada=None):
        #! Parametros JSON
        self.params = params

        # Dados da entrada do usuario
         self.DADOS_ENTRADA = dadosEntrada
        self.CXPB = params["CROSSOVER"]
        self.MUTPB = params["MUTACAO"]
        self.NGEN = params["NUM_GENERATIONS"]
        self.POP_SIZE = params["POP_SIZE"]
        self.SIZE_INDIVIDUAL = params["IND_SIZE"]
        self.TAXA_GENERATION = params["RCE_REPOPULATION_GENERATIONS"]
        self.CROSSOVER, self.MUTACAO, self.NUM_GENERATIONS, self.POPULATION_SIZE = (
            self.CXPB,
            self.MUTPB,
            self.NGEN,
            self.POP_SIZE,
        )
        self.NUM_VAR_DIF = params["NUM_VAR_DIFERENTES"]
        self.porcentagem = params["PORCENTAGEM"]
        self.delta = params["VALOR_LIMITE"]

         #!Criando individuo pelo deap com seus atributos
```

```
self.toolbox = base.Toolbox()

# Função de fitness de minimização
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))

# parâmetros
creator.create(
    "Individual", list, fitness=creator.FitnessMin, rce=str, index=int
)

# Variavel de decisão
self.toolbox.register("attribute", random.uniform, -5.12, 5.12)

# registrando os individuos
self.toolbox.register(
    "individual",
    tools.initRepeat,
    creator.Individual,
    self.toolbox.attribute,
    n=self.SIZE_INDIVIDUAL,
)
#! paramentos evolutivos
self.toolbox.register(
    "population", tools.initRepeat, list, self.toolbox.individual
)
self.toolbox.register("mate", tools.cxTwoPoint)
self.toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.1)
self.toolbox.register("select", tools.selTournament, tournsize=3)
self.toolbox.register("evaluate", self.evaluate_fitness)
```

- **Laço das Repopulações**: O algoritmo entra em um laço que continua enquanto o número de repopulações é menor ou igual ao máximo definido pelo usuário.
- **Geração da População Inicial**: Para cada repopulação, o algoritmo gera uma população inicial de indivíduos, que é essencialmente um conjunto de soluções candidatas.
- **Laço das Gerações:** Dentro de cada população, o algoritmo inicia um novo laço para cada geração, que continua enquanto o número de gerações for menor ou igual a um valor determinado (normalmente denotado como 'g'). O Código 2 demonstra o funcionamento do loop onde é utilizado a seleção, crossover, mutação e avaliação de acordo com o funcionamento do DEAP[4].
- **Cálculo da Função de Aptidão**: Para cada indivíduo na população, o algoritmo calcula sua função de aptidão, que avalia quão boa é a solução representada pelo indivíduo em relação ao problema em questão. O Código 7 demonstra a função Rastrigin[14] como função aptidão por padrão, onde busca a minimização para o valor global 0,0. Na seção "2.2.8.3 Exemplo de uso do Framework" é demonstrado como configurar o framework para a função objetivo do usuário.

Código 7: Função Objetivo Rastrigin

```python
def rastrigin(self, individual):
        self.evaluations += 1
        rastrigin = 10 * self.SIZE_INDIVIDUAL
        for i in range(self.SIZE_INDIVIDUAL):
            rastrigin += individual[i] * individual[i] - 10 * (
                math.cos(2 * np.pi * individual[i])
            )
        return rastrigin
```

- **Preservação do Indivíduo com Melhor Aptidão (Elitismo):** Durante cada geração, o algoritmo preserva o indivíduo com a melhor aptidão, garantindo que as soluções de alta qualidade sejam mantidas na população. No Código 8 é ilustrado como preservar o melhor indivíduo usando a ferramenta *ToolBox* onde possui o objeto HOF (Hall of Fame) onde retorna o melhor indivíduo da população atual onde seu clone é passado para a população seguinte preservando esse indivíduo.

Código 8: Função de elitismo simples usando funções nativas do DEAP

```python
def elitismoSimples(self, pop):
        self.hof.update(pop)
        pop[0] = self.setup.toolbox.clone(self.hof[0])
        return pop
```

- **Execução das Operações Genéticas para Evolução**: O algoritmo executa operações genéticas, como seleção, cruzamento e mutação, para permitir que a população evolua ao longo das gerações. Essas operações são fundamentais para gerar novas soluções com base nas soluções existentes.
- **Geração da Nova População**: Com base nas operações genéticas, uma nova população é gerada, substituindo a população anterior.
- **Preservação do Indivíduo com Melhor Aptidão em Conjunto Elite Automático**: Além da preservação do melhor indivíduo na população atual, o algoritmo também preserva um conjunto elite automático de indivíduos de alta qualidade, garantindo diversidade e qualidade na próxima geração.
- **Finalização do Relatório com Programação Otimizada**: Ao final de todas as repopulações, o algoritmo finaliza a execução e fornece um relatório com a

programação otimizada, que representa a melhor solução encontrada durante o processo evolutivo.

Para validar a implementação, foram realizados os testes descritos na seção "2.2.7 Simulações do Framework". Nas Tabelas de 4 a 9 são mostrados os resultados dos testes de validação do framework realizado. Cada linha da tabela contém estatísticas geradas a partir de 20 execuções do AE para cada conjunto de parâmetros. Estas tabelas ilustram as possibilidades de uso do framework como ferramenta de simulação e validam as funcionalidades implementadas.

Tabela 4 - Testes 20x de Benchmarking - Cruzamento

Melhor Aptidão				Número de gerações até a solução					
Melhor	Média	Desvio Padrão	CV	Variáveis Decisão	Menor Geração da Solução	Média	Desvio Padrão	CV	Cruzamento
0,01	0,55	2,7	20%	[0,006; 0,003; -4,448; -0,093; -0,013; 2,961; -3,912; -3,87; -0,683; 3,969]	97	68	25,35	268%	**0,20**
0,08	0,16	0,83	19%	[0,02; 0,004; -3,506; 3,995; 4,28; 4,383; 5,084; 4,614; 3,493; -4,662]	31	60	21,11	284%	**0,40**
0,02	0,13	0,71	18%	[-0,009; 0,001; 4,015; -2,667; 1,382; 6,034; -0,479; -0,573; 3,317; 0,325]	84	60	24,45	245%	**0,60**
0,08	0,17	0,75	23%	[0,007; -0,018; 4,789; 1,872; 4,376; -1,471; 4,127; 2,502; -0,563; 4,007]	55	71	15,27	465%	**0,90**

Fonte: Autoria Própria

Tabela 5 - Testes 20x de Benchmarking - Mutação

Melhor Aptidão				Número de gerações até a solução					
Melhor	Média	Desvio Padrão	CV	Variáveis Decisão	Menor Geração da Solução	Média	Desvio Padrão	CV	MUTAÇÃO
0,24	0,3	0,6	50%	[0,03; 0,018; -3,396; 3,752; 3,763; -2,554; 6,304; -4,783; -4,213; 0,735]	28	61	27,5	222%	**0,05**
0,03	0,41	2,68	15%	[0,006; -0,01; -4,607; 3,413; 4,448; -4,973; 2,453; 4,31; 0,294; -1,926]	27	60	23,84	252%	**0,10**
0,33	0,77	2,4	32%	[-0,04; -0,008; -0,785; 0,686; -0,411; 0,499; -2,357; -1,572; 1,969; -0,682]	69	68	24,18	281%	**0,15**
0,04	1,1	4,87	23%	[0,01; -0,011; -5,777;3,084; -4,68; 0,010; -5,49; 0,179, -2,894	92	64	22,26	288%	**0,20**

Fonte: Autoria Própria

Tabela 6 - Testes 20x de Benchmarking - Percentual Elite (Critério 1)

Melhor Aptidão				Número de gerações até a solução					
Melhor	Média	Desvio Padrão	CV	Variáveis Decisão	Menor Geração da Solução	Média	Desvio Padrão	CV	PERCENTUAL ELITE
0,03	0,04	0,07	57%	[0,004; -0,011; -0,654; -4,935; -4,312; 1,841; -3,955; -2,355; -1,921; 2,93]	11	62	27	230%	**30**
0,01	0,11	0,2	55%	[-0,007; -0,002; 3,863; -1,8; -3,887; 3,723; -2,059; 1,391; 1,909; 4,017]	79	66	18,9	349%	**60**
0,01	0,1	0,23	43%	[0,005; 0,003; 4,288, -0,918; 1,709; 4,519; 3,82; 4,485; 1,197; 0,538]	57	71	19,94	356%	**100**
0,0078	0,12	0,23	52%	[0,003; -0,001; 0,601; 1,905; 1,541; 1,162; 0,541; 1,662; -4,809; -5,355]	63	64	24,97	256%	**300**

Fonte: Autoria Própria

Tabela 7 - Testes 20x de Benchmarking - Número Variáveis de Decisão (critério 2)

Melhor Aptidão				Número de gerações até a solução					
Melhor	Média	Desvio Padrão	CV	Variáveis Decisão	Menor Geração da Solução	Média	Desvio Padrão	CV	NÚMERO DE VARIÁVEIS DE DECISÃO
0,04	0,16	0,25	64%	[-0,011; -0,008; -0,077; 4,156; -4,664; -4,801; -0,639; -2,121; -3,589; -2,257]	96	72	19,96	361%	1
0,01	0,08	0,13	62%	[0,005; -0,005; -5,076; 4,014; 4,502; 2,376; -0,789; 2,024; -4,988; -4,566]	12	72	27,73	260%	2
0,42	1,48	3,73	40%	[-0,014; 0,044; 3,937; 0,734; 2,564; 1,333; -2,212; 4,623; -0,226, 4,865]	54	69	25,31	273%	3
1,48	0,08	0,11	73%	[0,004; 0,039; -0,031; 0,047; -0,042; 0,031; -0,001; -0,004; -0,007; -0,004]	95	92	6	1533%	5

Fonte: Autoria Própria

Tabela 8 - Testes 20x de Benchmarking - Delta (critério 3)

Melhor Aptidão				Número de gerações até a solução					
Melhor	Média	Desvio Padrão	CV	Variáveis Decisão	Menor Geração da Solução	Média	Desvio Padrão	CV	DELTA
0,02	0,11	0,12	92%	[-0,01; -0,002; -4,464; 0,057; -4,71; -2,69;, 4,645; -3,133; -2,329; 4,298]	29	60	24,78	242%	0,10
0,0039	0,09	0,15	60%	[0,001; 0,001; -4,083; -2,299; -4,035; 2,488; 2,415; -0,485; -2,411; -1,015]	61	67	23,69	283%	0,50
0,03	0,1	0,22	45%	[-0,008; -0,008, -2,044; 2,557; -5,207; -2,1356; 1,116; 1,584; 3,974; 2,629]	45	77	18,45	417%	1
0,01	0,1	0,13	77%	[0,004; -0,007; -4,098; -2,354; -2,118; -4,7; 1,168; -1,191; -5,346; -2,609]	43	68	19,15	355%	3

Fonte: Autoria Própria

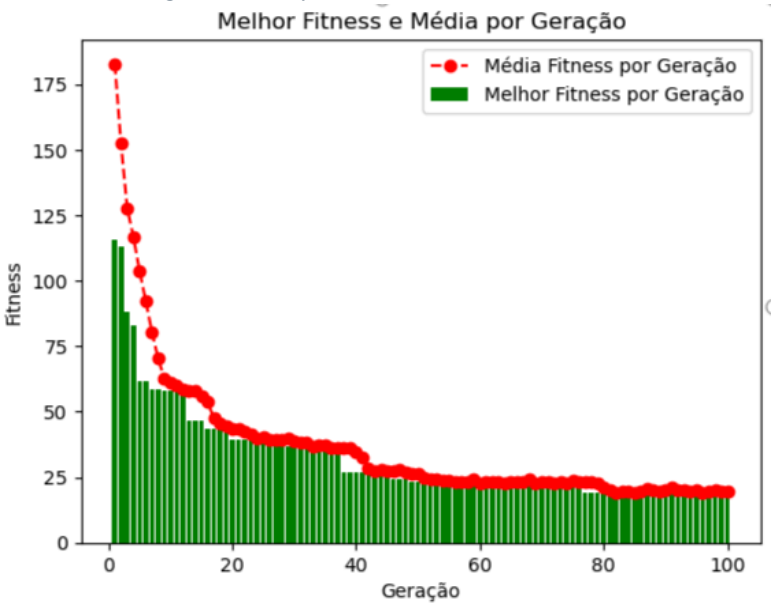
Tabela 9 - Testes 20x de Benchmarking - Quantidade de aplicações RCE

Melhor Aptidão				Número de gerações até a solução					
Melhor	Média	Desvio Padrão	CV	Variáveis Decisão	Menor Geração da Solução	Média	Desvio Padrão	CV	QUANTIDADE DE APLICAÇÕES RCE
0,18	0,14	0,18	78%	[0,001; -0,03; 4,837; 0,21; 0,289; 3,185; -4,016; -2,713; -0,929; -3,792]	12	69	30,26	228%	**5**
0,09	0,11	0,17	65%	[0,02; 0,006; -4,938; 2,852; -5,134; 1,661; 1,397; 6,542; -6,132; -3,61]	82	65	23,87	272%	**10**
0,01	0,05	0,06	83%	[-0,002; -0,008; 3,426; -4,323; -2,743; 3,546; 1,62; -5,408; 5,439; -3,257]	80	79	24,45	323%	**15**
0,02	0,07	0,1	70%	[0,006; 0,008; -0,417; -1,912; -2,184; 2,572; -4,633; -4,15; 2,32; -1,657]	58	74	19;=,65	377%	**50**

Fonte: Autoria Própria

Foi usado como parâmetro uma variável *RCE* que admite valores True ou False para que o usuário adote ou não a estratégia RCE [1]. Na figura 4, é ilustrado um exemplo de minimização sem a estratégia RCE onde é possível observar os valores da função aptidão diminuindo a cada geração.

Figura 4: Gráfico de minimização sem RCE



Fonte: Autoria própria

Na Tabela 10, é apresentada uma amostra de 10 indivíduos gerados pelo framework na última geração do algoritmo evolutivo sem o RCE, onde é possível encontrar os valores das variáveis de decisão e seus respectivos valores de função aptidão. Um dos desafios no desenvolvimento das estratégias de diversificação do framework foi controlar a presença de indivíduos clones. Considerando a pressão seletiva [1], é possível que nas gerações finais, clones dos melhores indivíduos ocupem todas as populações até a última geração, podendo trazer indivíduos com os mesmos valores de variáveis de decisão e função aptidão. Uma coluna diversidade foi adicionada para identificar indivíduos clones. Este índice é calculado somando os valores dentro do Array de variáveis de decisão, sendo possível identificar facilmente se o indivíduo é clone ou não a partir desse valor.

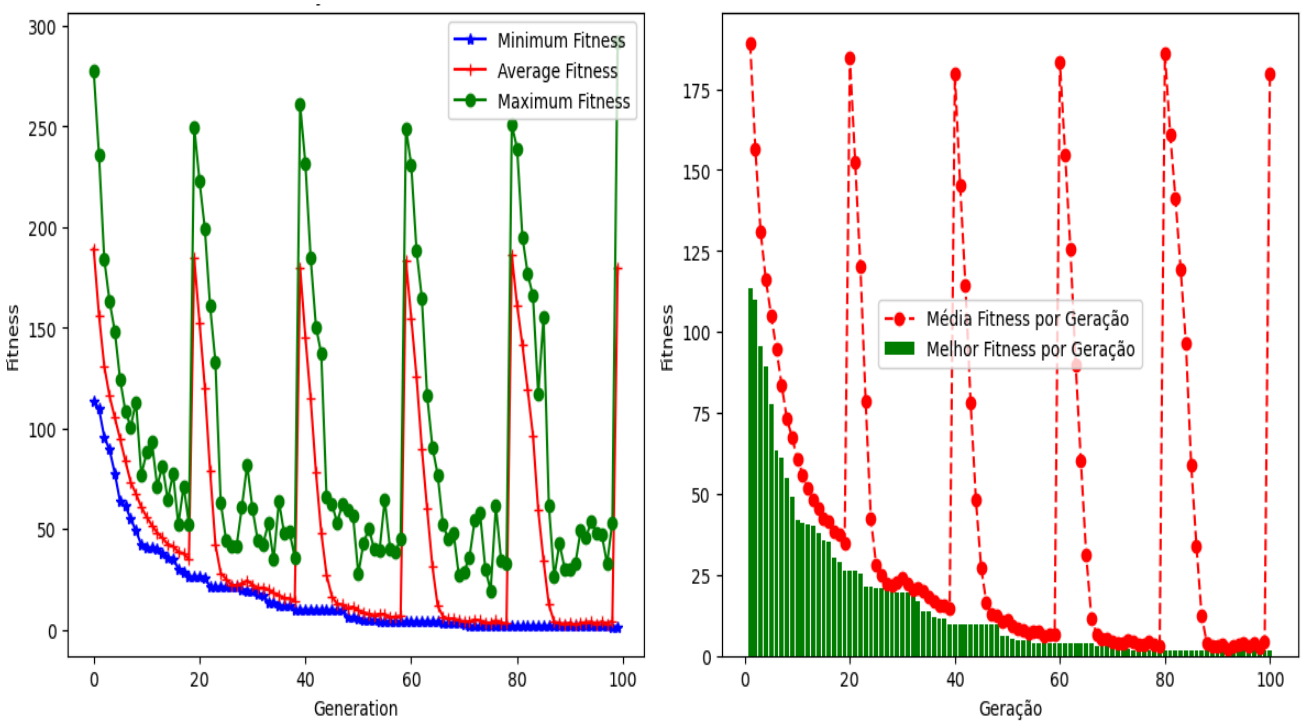
Tabela 10: Tabela resultado final

Índice	Variáveis de Decisão	Aptidão	Diversidade
0	[-1,014,-0,048,0,027,0,034,0,020,-0,028 , 0,007, -0,999 -0,013, 0,058]	3,882308247	-2,024109153
1	[-1,014; -0,048; 0,027; -0,034; 0,020; -0,028; 0,007; -0,999; -0,013; 0,031]	3,413604166	-2,050666008
2	[-1,014; -0,048; 0,027; -0,034; 0,020; -0,028, 0,007; -0,999; -0,013; 0,031]	3,413604166	-2,050666008
3	[-1,014; -0,0486; 0,027; -0,289; 0,020; -0,028; 0,007; -0,405; -0,013; 0,031]	3,13826679	-1,71090229
4	[-1,014; -0,048; 0,027; -0,034; 0,020; -0,028; 0,007; -0,999; -0,013; 0,031]	3,413604166	-2,050666008
5	[-1,014; -0,048; 0,027; -0,034; 0,020; -0,028; 0,007; -0,999; -0,013; 0,031]	3,413604166	-2,050666008
6	[-1,014; -0,048; 0,027; -0,034; 0,020; -0,028; 0,007; -0,999; -0,013; 0,031]	3,413604166	-2,050666008
7	[-1,014; -0,048; 0,027; -0,034; 0,020; -0,028; 0,007; -0,999; -0,013; 0,031]	3,413604166	-2,050666008
8	[-1,014; -0,048; 0,027; -0,034; -0,488; -0,028; 0,007; -0,999; -0,013; 0,031]	23,54587841	-2,560095403
9	[-1,014; -0,048; 0,027; -0,034; 0,020; -0,028; 0,007; -0,999; -0,013; 0,031]	3,413604166	-2,050666008
10	[-1,014; -0,048; 0,027; -0,034; 0,020; -0,028; 0,007; -0,999; -0,013; 0,031]	3,413604166	-2,050666008

Fonte: Autoria própria

Habilitando o parâmetro RCE, são obtidos resultados diferentes e interessantes, ilustrados na Figura 5. Foram encontrados picos nos valores de média e máximo da função aptidão durante a geração onde a estratégia é aplicada. Esse comportamento indica que as populações ficam mais diversificadas após a aplicação da estratégia RCE.

Figura 5: Algoritmo de minimização com RCE desligado



Fonte: Autoria própria

Na tabela 10, é ilustrada uma amostra do resultado final usando a estratégia RCE[1]. Nota-se que é preservado o melhor indivíduo (HOF), os indivíduos do conjunto elite e o restante da população aleatória. Na Tabela 10 é possível identificar que os indivíduos do RCE possuem o fitness menor do que o restante da população, comprovando que este conjunto possui melhores resultados e maior diversificação. O framework permite que seja gerado um arquivo .xlsx da população final para o usuário salvar e analisar os resultados obtidos.

Tabela 10: Exemplo de amostra de 10 indivíduos da população final com HOF e conjunto elite + aleatório com RCE ligado

Índice	Generations	Variáveis de Decisão	Fitness	RCE	Diversidade
0	100	[0,026; 0,026; -0,041; -0,033; -0,059; -0,003; 0,003; -0,006; 0,011; -1,016]	2,671954659	HOF	-1,092993587
1	100	[0,026; 0,026; -0,001; 0,992; 0,022; 1,038; 0,003; -0,006; 0,011; -1,016]	3,87166862	SIM	1,09759862
2	100	[0,026; 0,026; -0,001; 0,992; -0,059; 0,022; 0,003; -0,006; 0,011; -1,016]	3,19700785	SIM	-0,000614642
3	100	[0,026; 0,026; -0,041; 0,992; -0,059; -0,003; 0,003; -0,006; 0,011; -1,016]	3,445277488	SIM	-0,066925593
4	100	[0,026; 0,026; -0,041; 0,992; -1,003; -0,003; 0,003; -0,006; 0,011; -1,016]	3,758407292	SIM	-1,011124625
5	100	[0,007; 1,41; 0,782; -4,440; -4,856; -0,153; 3,211; 0,918; 4,732; -3,848]	172,2948716		-2,235588156
6	100	[-1,209; -2,266; -5,027; -1,957; -3,334; -5,081; 4,061; -3,415; -2,426; -3,512]	212,6703767		-24,16913919
7	100	[2,038; 2,297; 2,081; 1,145; 4,268; -3,190; 1,443; -0,865; -0,911; -3,427]	138,0224418		4,878388311
8	100	[4,485; -2,134; -3,269; 1,846; 4,856; 2,434; 0,778; 5,095; -2,467; 3,859]	211,0289823		15,48501801
9	100	[-1,108; -2,808; -3,256; -2,877; 0,463; 2,658; 3,3913; -4,764; 0,164; -3,936]	174,6393264		-12,07365613
10	100	[-0,751; 1,821; -2,301; -3,928; 1,015; -3,062; 2,272; 2,874; -4,205; 3,063]	128,4772309		-3,202018882

Fonte: Autoria Própria

A biblioteca DEAP[4], com a ferramenta *ToolBox*, gera uma tabela de estatísticas descritivas atualizada a cada geração executada. São calculados os valores mínimo, máximo, média e desvio padrão da função aptidão sem a necessidade de fazer os cálculos de forma manual dentro do algoritmo. Um exemplo dessa tabela com uma amostra de 10 resultados está ilustrada na tabela 11.

Tabela 11: Tabela estatística descritiva da população final fornecido pelo DEAP

Generation	Min Fitness	Average Fitness	Max Fitness	Std Fitness
0	2,864421198	34,53589423	67,73881861	13,45998905
1	2,864421198	23,04505099	48,19686701	10,93279366
2	2,679943571	13,43169385	44,67122588	9,144120321
3	2,225828417	7,09271247	25,67664082	5,136410944
4	2,225828417	3,563298376	13,04757152	1,862387326
5	2,225828417	2,703021388	6,025143002	0,405251682
6	2,225828417	2,847286338	21,44636462	2,475669375
7	2,225828417	2,55234679	22,02930334	2,10526435
8	2,225828417	2,372793915	16,73790061	1,44386421
9	2,225828417	2,658082529	24,8907014	2,746485857
10	2,225828417	2,265749411	6,217927791	0,397208872

Fonte: Autoria própria

3. PRODUÇÃO TÉCNICO-CIENTÍFICA

Não foi gerado nenhuma publicação relacionada ao projeto até o momento.

Foi criado um repositório do framework onde os usuários podem sugerir melhorias e sugestões de código em conjunto da comunidade, por ser uma linguagem *open source* ,onde pode ser encontrado em:https://github.com/PedroVic12/Repopulation-With-Elite-Set

Nesta pasta compartilhada podem ser encontrados exemplos de caso de uso em Jupyter Notebook que podem ser usados no Google Colab: https://drive.google.com/drive/folders/1_1ockci-U5fLBE3QOtIINglJvqPp9rk7

4. CONCLUSÕES

Na primeira fase do projeto houve uma grande ênfase na parte teórica para o desenvolvimento da framework. Infelizmente, o primeiro orientado teve problemas pessoais e não conseguiu desenvolver o framework nem apresentar resultados. O segundo bolsista conseguiu avançar na escolha da biblioteca em código aberto para meta-heurísticas e já implementou um algoritmo evolutivo básico com uma função de benchmark. Também já implementou um elitismo simples e desenvolveu o operador de Repopulação com Conjunto Elite na biblioteca escolhida.

Durante o desenvolvimento do framework em Python, utilizando algoritmos evolutivos e a estratégia RCE, os resultados obtidos proporcionam a base para resolver problemas de minimização em

engenharia elétrica. O framework obteve resultados significativos para a função benchmark Rastrigin, demonstrando sua capacidade de encontrar soluções eficientes para problemas complexos com múltiplos mínimos locais. A metaheurística RCE se mostrou eficiente para aumentar a diversidade da população e evitar convergência prematura, resultando em soluções de alta qualidade. O framework, baseado em Python e bibliotecas de código aberto, é intuitivo de configurar e usar, facilitando a aplicação para diferentes problemas e a inclusão de novas metaheurísticas. Através do framework foi possível realizar simulações usualmente realizadas na aplicação de algoritmos evolutivos para problemas de otimização. A alteração dos parâmetros por arquivo JSON facilita múltiplas execuções do algoritmo. Nas simulações foi possível observar um comportamento do processo evolutivo, através de gráficos. Notou-se claramente o efeito da mutação e dos parâmetros RCE nos gráficos de aptidão média. Também verificou-se como resultado uma população diversificada modulada pela escolha dos parâmetros da estratégia RCE.

Aplicações:

O framework desenvolvido possui um grande potencial de aplicação para diversos problemas em Engenharia Elétrica, como:

- Otimização do Despacho de Energia: Minimizar custos de produção e perdas de energia em redes de energia elétrica.
- Planejamento da Expansão de Redes: Encontrar a melhor configuração para expansão de redes de energia, considerando custos e restrições técnicas.
- Gerenciamento de Armazenamento de Energia: Otimizar o uso de baterias e outros sistemas de armazenamento de energia em redes elétricas ou em veículos com motores elétricos.
- Agendamento de intervenções em redes de energia elétrica.

5. REFERÊNCIAS BIBLIOGRÁFICAS

[1] ZANGHI, Rainer. Meta-heurísticas aplicadas ao Agendamento de Intervenções em Redes Elétricas, Tese de Doutorado, Instituto de Computação, Universidade Federal Fluminense, [pdf]. Disponível em: http://www.ic.uff.br/PosGraduacao/frontend-tesesdissertacoes/download.php?id=741.pdf&tipo=trabalho , [Acessado em Fevereiro de 2024], 2016.

[2] THURNER, Leon. et al. pandapower - an Open Source Python Tool for Convenient Modeling, Analysis and Optimization of Electric Power Systems, in IEEE Transactions on Power Systems, vol. 33, no. 6, pp. 6510-6521, Nov. 2018

[3] GREGA, Vrbančič et al. NiaPy: Python microframework for building nature-inspired algorithms, DOAJ Directory of Open Access Journals, Journal of open source software, 2018-03, Vol. 3 (23), p.613

[4] FORTIN, Félix-Antoine; DE RAINVILLE, François-Michel, GARDNER, Marc-André; PARIZEAU, Marc; GAGNÉ, Christian. "*DEAP*: Evolutionary Algorithms Made Easy", Journal of Machine Learning Research, pp. 2171-2175, no 13, jul 2012

[5] PRATA, Bruno , UFCE, Introdução as Metaheurísticas. Disponível em: https://www.dropbox.com/scl/fi/vv3f3sc2tqikbxevnqstq/MH-Aula-1.pdf ?rlkey=ge3mc5v7nammaltjpo15paft8&e=1&dl=0 , [Acessado em Agosto de 2024],

[6] BLUM, C., ROLI, A., "Metaheuristics in combinatorial optimization: Overview and conceptual comparison", ACM Computing Surveys, Vol. 35, Issue 3, pp.268-308, Set. 2003.

[7] MURKHERJEE, Debanjan., MALICK, Sourav., ABHISHEK, Rajan. A Levy Flight motivated meta-heuristic approach for enhancing maximum loadability limit in practical power system. Applied Soft Computing, vol. 114, Jan. 2022, doi: 10.1016/j.asoc.2021.108146

[8] KHAWAJA, A. Waheed et al. Design of a Damping Controller Using the SCA Optimization Technique for the Improvement of Small Signal Stability of a Single Machine Connected to an Infinite Bus System, Energies, Energies 2021, 14(11), 2996, doi: 10.3390/en14112996

[9] ZAMEE, M. Ahsan., WON, Dongjun. Novel Mode Adaptive Artificial Neural Network for Dynamic Learning: Application in Renewable Energy Sources Power Generation Prediction. Energies 2020, 13(23), 6405, doi: 10.3390/en13236405

[10] RADOSAVLJEVIRĆ, Jordan. Metaheuristic Optimization in Power Engineering. The Institution of Engineering and Technology, 2018.

[11] TALBI, El-Ghazali. Metaheuristics - From Design to Implementation, John Wiley & Sons, 2009.

[12] HARARI, Yuval. Homo Deus. Uma breve história do amanhã, , 2013

[13] DREO , Johann; LIEFOOGHE , Arnaud; VEREL, Sébastien; SCHOENAUER, Marc; MERELO , Juan J.; QUEMY, Alexandre; BOUVIER, Benjamin; GMYS, Jan, Paradiseo: from a modular framework for evolutionary computation to the automated design of metaheuristics —22 years of Paradiseo—, GECCO'21: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 1522–1530 (2021).

[14] WIKIPEDIA, Funções de Benchmarking, Disponível em: https://en.wikipedia.org/wiki/ Test_functions_for_optimization, [Acessado em agosto de 2024], 2024.

[15] ZANGHI, R., Souza, J. C. S. , Filho, M. B. C., Assis, T. M. L., Optimized coordination of transmission network outages in interconnected power grids, Electric Power Systems Research, Volume 170, 2019, Pages 72-80, ISSN 0378-7796, https://doi.org/10.1016/j.epsr.2019.01.016.

6. AUTO AVALIAÇÃO DOS BOLSISTAS

6.1. AUTO-AVALIAÇÃO DO BOLSISTA DIEISON MARIANO FARIAS ROCHA

No início, devido ao feedback constante do meu orientador nas reuniões, as coisas estavam indo bem, mas devido a problemas pessoais, eu não consegui mais ter foco na pesquisa.

Eu comecei procurando por teses, livros, artigos, entre outros que eram relacionados aos 4 tópicos dos resumos a serem realizados no CAPES, como indicado pelo orientador eu procurei por volta de 4 ou 5 referências para cada tópico.

Infelizmente, eu subestimei minha capacidade de entender artigos científicos levando um pouco mais do que o esperado para conseguir extrair as informações que me eram importantes, assim como saber quão importante o artigo para aquilo que eu precisava, além da falta de conhecimento para alguns tópicos mencionados nos artigos. O tempo gasto nessa parte acabou afetando na demora de conseguir entregar resumos.

Outro problema foi que eu tive dificuldades em gerenciar meu tempo para focar tanto na faculdade quanto na pesquisa, acabando dando mais tempo para a faculdade. Acreditei que eu iria recuperar o tempo perdido inicialmente, mas não imaginei que problemas pessoais iriam acontecer.

Ver os dias passarem e eu não conseguir fazer nada para a pesquisa foi um peso tão grande na consciência e um desgaste mental que afetava até meu dia a dia enquanto cuidava da minha mãe, ao ponto que conversar com meu orientador sobre desistir da pesquisa foi um alívio mental.

Devido a todos esses problemas, eu não consegui terminar um resumo sequer. E, além disso, não consegui nem começar a framework e por isso, nenhum resultado foi obtido. O que foi apresentado nesse relatório parcial é apenas aquilo que eu consegui entender enquanto preparava para fazer os resumos.

Gostaria de agradecer ao meu orientador por toda ajuda que me deu durante todo esse processo, assim como tudo que ele me ensinou nas reuniões que tivemos.

6.2. AUTO AVALIAÇÃO DO ALUNO PEDRO VICTOR RODRIGUES VERAS

Após pesquisas e estudos, foi possível fortalecer o raciocínio lógico e matemático, que utiliza funções matemáticas em conjunto com programação para encontrar soluções em problemas reais. Isso ajudou a fortalecer o senso crítico da modelagem computacional, aprimorando o entendimento de algoritmos, a organização de código e a criação de um próprio framework chamando sua própria classe e utilizando como parâmetros os dados de entrada do algoritmo evolutivo, as variáveis de decisão do problema e sua função de aptidão que busca facilidade de uso. Isso despertou muita curiosidade da minha parte sobre como é possível realizar maximização ou minimização em problemas voltados para Engenharia Elétrica, através de funções de benchmark como a de Rastrigin, possibilitando a comparação

entre soluções viáveis ou não. Também fortaleceu o entendimento de matemática computacional e problemas bio inspirados ao comparar seu uso com outros tipos de algoritmos de metaheurísticas e de *machine learning* de aprendizado supervisionado.

A importância do aprendizado das meta-heurísticas, da matemática e da programação reside na capacidade de resolver problemas complexos de maneira eficiente e otimizada. As meta-heurísticas fornecem técnicas poderosas para explorar o espaço de soluções em busca das melhores opções, enquanto a matemática fornece os fundamentos teóricos necessários para compreender e modelar os problemas, por sua vez, a programação permite a implementação prática dessas soluções, transformando conceitos abstratos em ferramentas reais para resolver problemas do mundo real. A combinação desses três elementos possibilita a criação de soluções inovadoras e eficazes para uma ampla gama de desafios.

Ao desenvolver do zero um framework foi um grande desafio, tenho experiência prática em programação em Python a 3 anos e fiquei muito satisfeito pelo resultado. Foi legal a experiência aluno e professor, visto que nunca tive uma pessoa com tempo e paciência para me explicar e auxiliar no processo de desenvolvimento. Busquei sempre boas práticas de código limpo ao desenvolver e o professor foi fundamental para resolver bugs e melhorar implementações de forma eficaz no algoritmo fortalecendo minhas habilidades de programação.

Foi realizado alguns testes em alguns meses até ser dominado o funcionamento da biblioteca DEAP seguindo sua documentação, fiz uma versão própria de funcionamento seguindo como referência o funcionamento do Arduino em C++ onde possuo experiência prévia, tendo uma classe Setup e uma classe do Algoritmo Evolutivo (Loop), essa abordagem serviu para separar as responsabilidades onde acredito que seja um forma de facilitar novos usuários ao usarem o framework visto que foi muito usado programação orientada a objetos (POO). Foi muito interessante colocar em prática a estratégia RCE desenvolvida pelo professor visto que realmente para conseguir os resultados satisfatórios e com de diversidade foi preciso testar e verificar tudo ocorrendo conforme os critérios estabelecidos, durante o desenvolvimento e foi muito gratificante ver que estava de acordo com o era previsto na pesquisa, o professor foi bem rigoroso sobre seu funcionamento e para mim foi muito importante o feedback para realmente entregar um software funcionando levando em conta os seus requisitos. Foi também desenvolvido um front-end em Python com Dash HTML onde eu como aluno pretendo entregar um projeto mais moderno possível, 100% open source e fácil de usar onde até a experiência do usuário seja levado em consideração para quem for usar essa ferramenta além do ambiente do Google Colab.