Projeto de Graduação

DEE DEPARTAMENTO DE ENGENHARIA ELÉTRICA

18 de dezembro de 2024

GADEMO Web: Uma Abordagem Computacional para Simulação e Otimização Evolutiva em Algoritmos Genéticos

Paloma Fernanda Loureiro Sette
Renan Sued Oliveira Castro

PUC RIO

Projeto de Graduação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

GADEMO Web: Uma Abordagem Computacional para Simulação e Otimização Evolutiva em Algoritmos Genéticos

Autor: Paloma Fernanda Loureiro Sette

Co-autor: Renan Sued Oliveira Castro

Orientador: Marley Maria Bernardes R. Vellasco

Co-orientador: Karla Tereza Figueiredo Leite

Trabalho apresentado como requisito parcial para a conclusão do curso de Engenharia de Controle e Automação na Pontifícia Universidade Católica do Rio de Janeiro do Rio de Janeiro, Brasil.

Agradecimentos

Gostaria de expressar meus sinceros agradecimentos a todas as pessoas que, de diversas formas, contribuíram para a realização deste trabalho.

Primeiramente, aos meus pais, Cristiane Sette e Sérgio Sette, que sempre me ofereceram apoio incondicional, incentivo e amor ao longo da minha jornada acadêmica e de vida. Agradeço também às minhas tias, que estiveram ao meu lado, em especial à tia Ana Sette, que, com sua presença e apoio constantes, foi fundamental para o meu crescimento pessoal e profissional. Por meio dela, tive a oportunidade de conhecer a Dra. Anna Leite, que se tornou muito mais do que uma terapeuta - foi uma mentora que me acompanhou durante grande parte da minha árdua jornada acadêmica e profissional, contribuindo significativamente para minha evolução e maturidade em todos os aspectos.

Aos meus professores, desde os primeiros anos escolares até o ensino superior, que me transmitiram conhecimento e valores que vão muito além das salas de aula. Em particular, às professoras Karla Figueiredo e Marley Vellasco, agradeço profundamente pela confiança, orientação e paciência durante o desenvolvimento deste trabalho. Suas contribuições foram essenciais para que eu pudesse trilhar esse caminho com segurança e determinação.

> "O único meio de escapar da corrupção causada pelo elogio é continuar trabalhando." — Marie Curie

Resumo

Este trabalho apresenta o desenvolvimento da plataforma GADEMO Web, uma ferramenta interativa e acessível para processamento de algoritmos genéticos em ambiente web. Inspirado na versão desktop original, o GADEMO foi modernizado com uma arquitetura de microsserviços, integrando um backend em Python com a biblioteca DEAP para execução dos algoritmos e um frontend em HTML, CSS e JavaScript para visualização dos resultados. A plataforma permite a otimização de funções por meio de algoritmos genéticos com parâmetros ajustáveis, como taxa de crossover, mutação, e tamanho da população, oferecendo uma interface intuitiva e gráficos que facilitam o acompanhamento da evolução das soluções. Este trabalho visa contribuir para o ensino e pesquisa na área de algoritmos evolutivos, fornecendo uma ferramenta robusta e escalável para otimização de funções matemáticas.

Palavras-chave: Algoritmos Genéticos, Otimização Evolutiva, Simulação Computacional, inteligência artificial, aprendizado de máquina, Gademo, Plataforma Web, Microsserviços, Deap, Processamento Evolutivo, Visualização de Dados, Ensino e Pesquisa

GADEMO Web: Uma Abordagem Computacional para Simulação e Otimização Evolutiva em Algoritmos Genéticos

Abstract

This work presents the development of the GADEMO Web platform, an interactive and accessible tool for processing and simulating genetic algorithms in a web environment. Inspired by the original desktop version, GADEMO was modernized with a microservices architecture, integrating a Python backend with the DEAP library for algorithm execution and a frontend built with HTML, CSS, and JavaScript for results visualization. The platform enables function optimization through genetic algorithms with adjustable parameters, such as crossover rate, mutation rate, and population size, offering an intuitive interface and graphical outputs that facilitate tracking the evolution of solutions. This project aims to contribute to the field of evolutionary algorithms in education and research by providing a robust and scalable tool for simulating complex optimization processes. Preliminary results indicate that the new GADEMO version not only enhances the tool's usability and accessibility but also enables more efficient analysis of genetic algorithm performance across various test scenarios.

Keywords: genetic algorithms, evolutionary optimization, computational simulation, artificial intelligence, machine learning, GADEMO, web platform, microsservices, DEAP library, evolutionary processing, data visualization, education and research

Sumário

List of Figures

1 Prefácio

Os Algoritmos Genéticos representam uma área fundamental da computação evolucionária, destacando-se como uma das principais técnicas de otimização inspiradas em processos biológicos (MITCHELL, 1998 [1]; GOLDBERG, 1989 [2]). Como parte integrante do campo de Machine Learning, os AGs tem demonstrado crescente relevância tanto no âmbito acadêmico quanto em aplicações comerciais, sendo amplamente utilizados em problemas complexos de otimização, desde o planejamento de rotas até o *design* de circuitos eletrônicos (EIBEN; SMITH, 2015 [3]). De acordo com Zhang et al. (2021) [4], a versatilidade dos AGs tem contribuído significativamente para sua adoção em diversos setores industriais e de pesquisa, demonstrando resultados promissores em problemas de otimização multiobjetivo.

A natureza abstrata dos conceitos envolvidos nos Algoritmos Genéticos, combinada com a complexidade matemática subjacente, cria uma barreira inicial significativa para muitos estudantes (KUMAR; VERMA, 2019 [5]). Esta situação evidencia a necessidade de ferramentas educacionais que possam tornar o aprendizado destes conceitos mais acessível e interativo, permitindo que alunos visualizem e compreendam na prática o funcionamento dos AGs. Estudos recentes de Silva e Santos (2020) [6] indicam que o uso de ferramentas interativas no ensino de computação evolucionária pode aumentar significativamente o engajamento e a compreensão dos estudantes.

2 Objetivos

O objetivo principal do presente trabalho é desenvolver e implementar uma plataforma web escalável, moderna e eficiente para a execução de modelos de otimização baseados em Algoritmos Genéticos (AG), denominada GADEMO (Genetic Algorithm Demo). A plataforma oferece uma interface intuitiva, flexível e interativa para os usuários, permitindo que os usuários explorem, visualizem e analisem os resultados produzidos por modelos baseados em Algoritmos Genéticos de maneira clara, acessível e com maior capacidade de customização, seguindo as recomendações de boas práticas propostas por Li e Wu (2022) [7].

Os objetivos específicos incluem:

- Criar uma aplicação web robusta, que utiliza tecnologias modernas para permitir a execução de modelos baseados em Algoritmos Genéticos, proporcionando maior acessibilidade e interação.
- **Modularidade e Separação de Componentes:** Implementar uma arquitetura baseada em microsserviços, com clara separação entre *frontend*, *backend* e comunicação via API RESTful (FELDING, 2000 [8])[1], seguindo as melhores práticas de design e implementação estabelecidas para interfaces web modernas (RODRIGUEZ et al., 2016 [9]). Segundo Newman (2021), em "*Building Microservices: Designing Fine-Grained Systems*" [10], uma arquitetura baseada em microsserviços promove maior escalabilidade e manutenibilidade do sistema.
- **Otimização e Flexibilidade dos Algoritmos Genéticos:** Permitir a customização dos parâmetros dos Algoritmos Genéticos, incluindo taxas de cruzamento, mutação, tamanho da população e número de gerações, possibilitando experimentos variados e comparações de desempenho.
- **Integração de Tecnologias de Visualização:** Desenvolver componentes de *frontend* que forneçam resposta visual clara e acurada sobre os resultados dos experimentos, com gráficos e interfaces interativas para acompanhamento da evolução dos AG.
- **Aplicação de Boas Práticas de Desenvolvimento:** Utilizar boas práticas de design de software, incluindo padrões de projeto (*Design Patterns*), princípios de *Domain-Driven Design* (DDD) e conteinerização (PAHL et al., 2019 [11]), garantindo qualidade do código, segurança e facilidade de implantação.
- **Escalabilidade e Infraestrutura em Nuvem:** Hospedar a aplicação em serviços de nuvem, como o Render (SANTOS et. al., 2020 [12]), para demonstrar o potencial de escalabilidade e disponibilidade do sistema, facilitando o acesso a usuários de diferentes locais.
- **Documentação e Demonstração de Funcionalidades:** Documentar detalhadamente o funcionamento da aplicação, a configuração de parâmetros, os experimentos realizados e os resultados obtidos, oferecendo um guia claro para novos usuários e desenvolvedores interessados.

Este trabalho busca, assim, contribuir para o estudo e aplicação de Algoritmos Genéticos em ambientes computacionais modernos, explorando e aprimorando o uso de tecnologias web para oferecer um ambiente educacional e experimental mais rico, flexível e acessível.

[1] A tese de doutorado de Fielding é particularmente muito interessante, pois é o documento que originou o conceito de REST.

3 Introdução

Os *Algoritmos Genéticos* (AG) são uma classe de algoritmos de busca e, eventualmente, otimização inspirados nos processos de seleção natural e reprodução genética presentes na natureza. Introduzidos por John Holland na década de 1970 (Holland, 1975 [13]) e popularizados por David Goldberg nos anos 1980 (Goldberg, 1989 [2]), os AG são utilizados para resolver problemas complexos, que frequentemente possuem espaços de busca extensos ou múltiplos mínimos/máximos locais. Esses algoritmos têm se mostrado especialmente úteis em problemas de difícil ou ausência de formulação matemática, onde é necessária uma abordagem flexível e adaptativa para encontrar soluções satisfatórias.

No entanto, a implementação prática dos AG enfrenta desafios significativos, especialmente quando aplicada em ferramentas de simulação e ensino. A plataforma **GADEMO**, desenvolvida inicialmente como uma aplicação desktop, oferece recursos de simulação de AG, mas sua arquitetura e interface limitam sua acessibilidade e usabilidade, dificultando sua utilização em contextos mais amplos, como ensino e pesquisa. Assim, surgiu a necessidade de modernizar e ampliar a plataforma, tornando-a mais robusta, escalável e intuitiva para os usuários. Nesse contexto, este trabalho visa o desenvolvimento de uma versão web do GADEMO e utiliza uma arquitetura de microsserviços que integra um *backend* em Python (VAN ROSSUM; DRAKE, 2011 [14]) com a biblioteca DEAP (FORTIN et al., 2012 [15]) para processamento dos algoritmos e um *frontend* em HTML5, CSS3 e Vanilla Javascript (SEVERANCE, 2012 [16]; MARCOTTE, 2017 [17]) para visualização e interação com os dados.

A modernização do GADEMO envolve a superação de alguns desafios técnicos, entre os quais destacam-se:

- **Escalabilidade:** A nova versão web deve ser capaz de lidar com múltiplos usuários simultâneos, mantendo a eficiência de processamento mesmo em cenários de alta demanda.
- **Eficiência Computacional:** Executar AGs requer um sistema robusto que permita realizar simulações complexas sem sobrecarregar os recursos de hardware.

A arquitetura proposta utiliza uma API desenvolvida em Python, que faz uso da biblioteca *DEAP* para implementar os operadores genéticos e as funções de aptidão, possibilitando uma experiência de simulação interativa e visualmente compreensível, adaptada tanto para iniciantes quanto para pesquisadores experientes.

O objetivo deste trabalho é, portanto, proporcionar uma plataforma aprimorada que viabilize o ensino e a pesquisa em Algoritmos Genéticos. A nova versão do GADEMO busca:

- Tornar a plataforma mais acessível e moderna, adequada ao ambiente educacional e de pesquisa.
- Melhorar a capacidade de simulação e análise de AGs, proporcionando uma experiência de usuário mais fluida e intuitiva.
- Permitir configurações detalhadas dos parâmetros dos AGs, como taxa de *crossover*, mutação e tamanho da população, facilitando o aprendizado sobre os efeitos de cada parâmetro.

a Organização do Trabalho

Este trabalho está estruturado em nove capítulos principais, além de um apêndice, conforme descrito a seguir:

- No **Capítulo 1**, são apresentados o **Prefácio** e os objetivos gerais do trabalho, destacando a motivação para o desenvolvimento do sistema *GADEMO* e os desafios enfrentados.
- O **Capítulo 2** aborda a *Introdução*, fornecendo uma visão geral sobre algoritmos genéticos, suas aplicações e importância no contexto de problemas de otimização. Este capítulo também contextualiza os objetivos específicos e organiza a estrutura do trabalho.
- O **Capítulo 3**, *Fundamentação Teórica*, introduz os conceitos fundamentais sobre algoritmos genéticos, incluindo representação genética, critérios de parada e métodos de seleção e reprodução. Este capítulo também apresenta a flexibilidade dos algoritmos

genéticos na personalização de características e suas aplicações em diferentes cenários de otimização.

- O **Capítulo 4**, *Metodologia*, detalha a arquitetura do sistema *GADEMO*, com ênfase nas práticas de desenvolvimento do *backend* e *frontend*, arquitetura de microserviços e integração com a infraestrutura utilizada. Este capítulo também explora os recursos tecnológicos empregados, como *Docker* e integração com Heroku, e as ferramentas utilizadas para análise de desempenho.
- No **Capítulo 5**, *Sistema GADEMO*, são descritas as etapas de fluxo do algoritmo genético na aplicação construída, desde a inicialização da população até os métodos de seleção, cruzamento e mutação. Além disso, o capítulo inclui seções dedicadas ao funcionamento detalhado do *Steady-State* e sua implementação no sistema.
- O **Capítulo 6** foca na *Interface e Funcionalidades do Sistema GADEMO*, explorando os componentes de entrada e visualização de resultados. Isso inclui gráficos de evolução de *fitness*, tabelas de resultados, e as ferramentas interativas para análise do desempenho dos experimentos.
- O **Capítulo 7**, *Resultados e Discussão*, apresenta os experimentos realizados com diferentes configurações de parâmetros do algoritmo genético, comparando métricas como desempenho, eficiência computacional e qualidade dos resultados. Este capítulo também discute os benefícios e limitações do sistema *GADEMO* no contexto avaliado.
- O **Capítulo 8**, *Conclusões e Trabalhos Futuros*, sintetiza as principais contribuições do trabalho e propõe direções para pesquisas futuras, tanto na área científica quanto na melhoria da plataforma.
- Por fim, o **Apêndice** inclui funções de *benchmark* utilizadas nos experimentos, códigos para visualização em MATLAB, e outros elementos complementares relevantes.

4 Fundamentação Teórica

Este capítulo apresenta uma introdução aos conceitos essenciais relacionados aos algoritmos genéticos: seus fundamentos e a estrutura/metodologia geral de seu desenvolvimento.

a Um Pouco de História

Para contar a legítima e completa história dos algoritmos genéticos, seria necessário voltar até o início do universo, mas esta monografia é curta demais para explorar os últimos 5 bilhões de anos[2] de forma detalhada. Até o século XIX, a ciência predominante oscilava entre duas teorias principais: o criacionismo, que defendia que Deus criou o universo como ele é, e a teoria da geração espontânea, que sugeria que a vida surgia de essências presentes no ar.

Foi em meados do século XIX que Charles Darwin embarcou em uma longa jornada a bordo do HMS Beagle. Durante suas expedições, visitou diferentes regiões do mundo, onde observou com grande atenção como os animais de uma mesma espécie apresentavam pequenas diferenças de acordo com os ecossistemas que habitavam. Essas variações demonstravam adaptações específicas às necessidades e oportunidades oferecidas por cada ambiente.

Essas observações foram fundamentais para que Darwin formulasse sua teoria da evolução das espécies, apresentada em 1859 no livro "A Origem das Espécies" [18]. Embora inicialmente recebida com muitas críticas, hoje sua obra é amplamente reconhecida e aceita pela comunidade científica global [3].

A teoria da evolução propõe que, na natureza, todos os indivíduos de um ecossistema competem por recursos limitados, como alimento e água. Aqueles que não conseguem se adaptar, tendem a deixar menos descendentes, o que reduz as chances de seus genes serem transmitidos às gerações futuras. Por outro lado, os indivíduos mais adaptados sobrevivem e se reproduzem, e a combinação de seus genes pode dar origem a descendentes ainda mais ajustados ao ambiente, incorporando as características vantajosas de seus progenitores.

No campo computacional, a história dos algoritmos genéticos começou a tomar forma na década de 1940, quando cientistas passaram a buscar inspiração na natureza para desenvolver o campo emergente da inteligência artificial. Inicialmente, o foco estava mais voltado à pesquisa cognitiva e ao entendimento dos processos de aprendizado e raciocínio. Foi somente no final dos anos 1950 que surgiu o interesse por modelos genéricos capazes de gerar soluções para problemas complexos demais para serem resolvidos com os métodos computacionais disponíveis na época.

O grande marco veio nos anos 1960, com John Holland, considerado o pai dos algoritmos genéticos. Ele estudou a evolução das espécies e propôs um modelo heurístico computacional que simulava esses processos naturais para solucionar problemas extremamente desafiadores, até então considerados insolúveis. Em 1975, Holland publicou seu livro "Adaptation in Natural and Artificial Systems" [13], que consolidou o conceito de algoritmos genéticos e impulsionou sua popularidade na comunidade científica. Desde então, esses algoritmos têm sido amplamente utilizados para abordar problemas complexos, contribuindo significativamente em áreas que antes pareciam inacessíveis [4].

b O que são Algoritmos Evolucionários?

Os algoritmos evolucionários são ferramentas computacionais inspiradas nos processos naturais de evolução (Rechenberg, 1973 [19]), utilizadas para solucionar problemas complexos. Embora existam diversos modelos computacionais baseados nesse conceito, todos compartilham a ideia central de simular a evolução de espécies por meio de mecanismos como seleção, mutação e

[2] Ou os últimos 6 mil anos, caso o criacionismo seja uma opção.

[3] Porém, nem toda a comunidade religiosa compartilha dessa aceitação. Em 2001, um grupo de religiosos nos Estados Unidos tentou barrar o ensino da teoria da evolução nas escolas públicas. Sem sucesso, felizmente.

[4] Certamente, alguém poderia desenvolver outra forma de resolvê-los, ainda que não tão eficiente.

reprodução (Holland, 1975 [13]). Esses processos dependem diretamente do desempenho de cada indivíduo dentro de um ambiente específico.

Essencialmente, o funcionamento desses algoritmos se baseia em manter uma população de estruturas que evoluem de maneira semelhante às espécies naturais, conforme descrito por John Holland (1975) [13]. Essa evolução ocorre attravés da aplicação de operadores genéticos, como recombinação e mutação, conforme detalhado por Goldberg (1989) [2], que ressalta a importância desses operadores para explorar o espaço de soluções de maneira eficiente.

Cada indivíduo na população é avaliado segundo uma função de aptidão, que mede sua qualidade como solução para o problema em questão. Esse mecanismo, inspirado no princípio de seleção natural de Charles Darwin (1859) [18], prioriza a "sobrevivência do mais apto", um processo que, de acordo com Rechenberg (1973) [19], promove melhorias iterativas na população ao longo das gerações. Dessa forma, os operadores genéticos conduzem a evolução da população, permitindo a descoberta de soluções cada vez melhores para problemas complexos, conforme demonstrado por Bäck, Fogel e Michalewicz (1997) [20] em suas pesquisas sobre computação evolutiva.

Então, algoritmos evolucionários podem ser definidos como modelos computacionais baseados nos princípios de evolução biológica, projetados para resolver problemas de otimização e busca em espaços de alta complexidade. Essa abordagem, como argumentado por Schwefel (1995) [**?**], destaca-se por sua adaptabilidade e eficácia em enfrentar desafios que requerem soluções inovadoras e não triviais.

Segundo Linden [21], comportamento básico dos algoritmos evolucionários pode ser exemplificado pelo seguinte pseudocódigo.

Algorithm 1 Pseudocódigo do Algoritmo Genético

Input: Parâmetros iniciais, população inicial $P(0)$
Output: Solução ótima ou população final
$t \leftarrow 0$ `// Inicializa-se o contador de tempo`
Inicializa_População $P(0)$ `// Inicializa-se a população randomicamente`
while *não terminar* **do**
  Avalie_População $P(t)$ `// Avalie a população neste instante`
  $P' \leftarrow$ Selecione_Pais $P(t)$ `// Seleciona-se subpopulação para nova geração`
  $P' \leftarrow$ Recombinação_e_Mutação P' `// Aplicamos operadores genéticos`
  Avalie_População P' `// Avalie esta nova população`
  $P(t+1) \leftarrow$ Selecione_sobreviventes $P(t), P'$ `// Selecione sobreviventes desta geração`
  $t \leftarrow t+1$ `// Incrementa-se o contador de tempo`
end

De acordo com Linden (2006) [21], esse modelo reflete, como esses algoritmos buscam, dentro da população atual, soluções que apresentem as melhores características, combinando-as para gerar resultados ainda mais eficientes. Esse ciclo se repete até que seja alcançado um tempo limite ou uma solução satisfatória.

Contudo, como fica evidente no funcionamento do algoritmo, ainda de acordo com Linden [21], os processos evolucionários são fortemente influenciados por fatores aleatórios. Então, desde a inicialização da população até a seleção dos pais durante a evolução, a aleatoriedade desempenha um papel crucial. Por isso, os resultados desses algoritmos raramente são totalmente reprodutíveis, e eles não garantem a obtenção da melhor solução possível em todas as execuções.

Essas características levam a uma conclusão importante: se há um algoritmo eficiente e preciso para resolver um problema, não há necessidade de utilizar um algoritmo evolucionário. Eles devem ser considerados principalmente para problemas onde os algoritmos exatos são muito lentos (como os problemas NP-completos) ou incapazes de fornecer uma solução (como problemas de maximização de funções multi-modais). Nesse contexto, os algoritmos evolucionários se mostram como uma abordagem útil para desafios particularmente difíceis.

c O que são Algoritmos Genéticos?

Os algoritmos genéticos (GAs) são uma abordagem dentro dos algoritmos evolucionários, baseando-se em uma analogia com o processo biológico de evolução natural. Em essência, eles funcionam como uma técnica de busca global e heurística [5], que difere de métodos como o *hill climbing*. Enquanto este último utiliza a derivada de uma função para encontrar o máximo, frequentemente ficando preso em máximos locais, como se observa na figura 1, os GAs buscam explorar amplamente o espaço de soluções, evitando essa limitação.

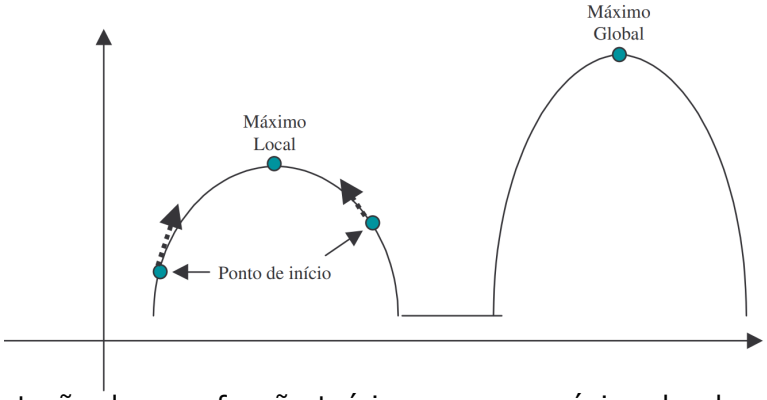


Figure 1: Representação de uma função teórica com um máximo local e outro global. Uma abordagem como o *hill climbing* que inicie em qualquer um dos pontos destacados irá seguir a direção de maior crescimento (ou gradiente) e acabará ficando presa no máximo local, onde a inclinação da função se torna zero. Os algoritmos genéticos, por sua vez, não possuem uma dependência tão acentuada em relação aos valores iniciais.

Linden (2006) [21] afirma que o funcionamento dos Algoritmos Genéticos é estruturado em torno de populações de indivíduos, que são geradas e submetidas a operadores genéticos, como seleção, recombinação (*crossover*) e mutação. Cada indivíduo é avaliado em termos de sua qualidade como solução para o problema em questão. Esse processo simula a evolução natural, resultando, ao longo do tempo, em indivíduos melhor adaptados, sendo possível que a solução ótima seja alcançada.

De maneira simplificada, Linden (2006) [21] define os AGs como algoritmos de busca que se baseiam nos mecanismos de seleção natural e genética. Eles combinam a sobrevivência dos melhores indivíduos com uma troca estruturada de informações genéticas entre dois indivíduos, criando assim uma poderosa estratégia de busca.

Diferentemente de métodos como *hill climbing*, segundo Linden (2006) [21], os AGs não se restringem ao primeiro máximo local encontrado. Linden [21] declara que esse comportamento reflete a evolução natural, onde a busca por melhores indivíduos continua, mesmo após identificar um que seja o mais adaptado em um dado momento. Na natureza, mudanças no ambiente podem alterar completamente quais características são vantajosas, como no caso de bactérias resistentes a antibióticos, que se tornam dominantes quando o ambiente muda.

Nos Algoritmos Genéticos, os operadores genéticos desempenham um papel central ao permitir uma exploração abrangente do espaço de soluções, mesmo em ambientes estáticos. Linden (2006) [21] ressalta que esse processo é inspirado na dinâmica da natureza em cenários de estabilidade ambiental, onde a evolução ocorre sem necessariamente buscar uma solução ideal. Em vez disso, como na seleção natural descrita por Darwin, a competição entre indivíduos favorece a sobrevivência dos mais aptos, levando à melhoria gradual das soluções ao longo das gerações.

Os Algoritmos Genéticos frequentemente encontram soluções ótimas em suas execuções, embora isso não seja garantido em todas as situações. No entanto, fornecem uma heurística que encontra boas soluções com consistência (Linden, 2006 [21]), ainda que não necessariamente

[5] Este termo é muito comumente usado ao se referir a Algoritmos Genéticos. Apesar do nome Algoritmos Genéticos sugerir o contrário, não encontram necessariamente a solução ótima para um problema e, quando o fazem, nem sempre repetem.

idênticas em cada execução [6]. Essa característica é útil, especialmente em problemas onde a busca exata seria inviável ou extremamente lenta.

De acordo com Linden (2006) [21], um ponto crucial dos AGs é a codificação da informação nos cromossomos, que conecta diretamente o algoritmo ao problema que se deseja resolver, pois são a representação das soluções ao problema. O autor explica que uma codificação adequada pode incluir as particularidades do problema, como restrições ou condições específicas. É importante ressaltar que, além da representação dos cromossomos, os operadores genéticos também devem ser projetados considerando essas restrições e especificidades do problema, garantindo que as soluções geradas permaneçam válidas durante todo o processo evolutivo. Desta forma, elimina-se a necessidade de validações adicionais para cada solução gerada. Conforme Linden (2006), ao final da execução, a solução codificada precisa ser decodificada para uso prático.

Linden (2006) [21] afirma que, semelhante ao que ocorre na natureza, a informação é representada nos cromossomos (ou genomas) se a representação não estiver alinhada ao espaço de solução, e a reprodução, equivalente à reprodução sexuada [7], guia a evolução da população. A mutação, por sua vez, ainda de acordo com Linden(2006) [21], promove diversidade ao alterar aleatoriamente os genes dos indivíduos, sendo aplicada com menos frequência do que a recombinação (*crossover*), que é o principal motor da reprodução.

Para Linden (2006) [21], a seleção dos indivíduos para reprodução deve favorecer os mais aptos, garantindo que suas características sejam transmitidas com maior frequência para a próxima geração. Contudo, os indivíduos menos aptos também devem ser mantidos no processo de reprodução em alguma medida, evitando uma convergência genética precoce, o que limitaria a busca por soluções mais variadas e possivelmente melhores.

Em resumo, os AGs são ferramentas poderosas para explorar espaços de soluções complexos, utilizando princípios inspirados pela evolução natural para criar e refinar soluções de forma eficaz e eficiente.

d Elementos de um Algoritmo Genético

De acordo com Barboza (2005) [22], os algoritmos genéticos utilizam uma terminologia inspirada na teoria da evolução natural e da genética. Nesse contexto, um indivíduo dentro de uma população pode ser composto por um ou mais cromossomos. Os termos ”indivíduo” e ”cromossomo” são frequentemente usados de forma intercambiável, especialmente quando cada indivíduo é representado por um único cromossomo.

O cromossomo, geralmente implementado como uma *string* ou vetor, é formado por elementos conhecidos como genes. Cada gene pode assumir valores específicos, chamados de alelos, que estão localizados em posições fixas no cromossomo, conhecidas como lócus. Juntos, genes e alelos compõem o genótipo, que determina as características observáveis do indivíduo, chamadas de fenótipo.

A relação entre a terminologia dos algoritmos genéticos e os conceitos da biologia é apresentada de forma resumida na Tabela 1.

[6] Segundo Linden [21], essa situação é semelhante àquela observada por Darwin em seu livro ”A Origem das Espécies”, ao descrever duas ilhas pertencentes a um mesmo arquipélago. Separadas por menos de cem metros de água, essas ilhas apresentavam clima semelhante e praticamente os mesmos nutrientes. No entanto, algumas espécies de animais, especialmente as terrestres, eram completamente distintas entre as ilhas. Darwin ficou impressionado com essa descoberta, mas atualmente sabemos que isso é resultado tanto da diversidade inicial das populações quanto da imprevisibilidade inerente ao processo de evolução natural.

[7] A escolha da reprodução sexuada como inspiração para os algoritmos genéticos não é por acaso. Esse tipo de reprodução, utilizado por todos os animais mais complexos, assegura uma maior diversidade biológica, pois ao combinar partes dos genomas de ambos os pais, é possível gerar descendentes mais adaptados. Com o passar das gerações, essa diversidade contribui para a evolução da população. Em contrapartida, a reprodução assexuada não promove variação, já que os descendentes são idênticos ao progenitor, herdando exatamente as mesmas características e habilidades.

Biologia	**Algoritmo genético**
Cromossomo	Indivíduo (string)
Gene	Bit (na representação binária) ou real (na representação real)
Alelo	Valor do bit
Lócus	Posição de um bit específico no indivíduo
Genótipo	Indivíduo candidato à solução – x
Fenótipo	Valor da função para um dado indivíduo – $f(x)$

Table 1: Relação da terminologia do AG com a biologia. Fonte: Barboza (2005) [22].

De acordo com Aytug. (2003) [23], os algoritmos genéticos geralmente são compostos por oito elementos principais: a forma de representação genética, a população inicial, a função de avaliação, o método de seleção para reprodução, os operadores genéticos, o critério de seleção entre gerações, as condições de parada e os parâmetros que configuram o algoritmo. Todos esses componentes serão abordados em maior profundidade ao longo do texto.

Um fluxograma genérico, capaz de abranger a maior parte dos algoritmos genéticos existentes, é apresentado na Figura 2. Nesse fluxograma, o processo começa com a criação de uma população inicial composta por possíveis soluções para o problema. Em seguida, os indivíduos dessa população são avaliados de acordo com a função de avaliação escolhida. Após essa etapa, verifica-se se o critério de parada do algoritmo foi atendido. Caso contrário, os indivíduos dessa geração são selecionados por meio de algum método de seleção para reprodução.

Os indivíduos escolhidos passam então pelos operadores genéticos, e, a partir deles, é criada uma nova geração de ”filhos”. Esse ciclo se repete até que o critério de parada seja satisfeito. Quando isso ocorre, o algoritmo finaliza sua execução apresentando a melhor solução encontrada para o problema.

1 Representação Genética

De acordo com Linden (2006) [21], a representação genética tem como objetivo traduzir as informações do problema em uma forma que os computadores possam processar. Quanto mais adequada for essa tradução, maior será a qualidade dos resultados obtidos.

Almeida (2007) [24] destaca que a representação do indivíduo em um algoritmo genético precisa descrever corretamente o espaço de busca relacionado ao problema. Concilio (2000) [25] reforça que uma representação inadequada pode levar a problemas de convergência prematura, tornando esta uma etapa crucial na definição de um Algoritmo Genético.

As formas mais comuns de representação são a **codificação binária** (também chamada de clássica) e a **codificação real**. Essas representações foram amplamente utilizadas e discutidas em trabalhos clássicos, como os de Holland (1975) [13] para codificação binária e Michalewicz (1996) [20]) para a codificação real. Uma terceira abordagem, menos utilizada, é a codificação inteira, que tem aplicação restrita.

A codificação binária é historicamente relevante, sendo introduzida nos trabalhos pioneiros de Holland (1975) [13]. Ela é simples de implementar, manipular e analisar, mas pode ser menos eficiente para problemas com parâmetros contínuos, que demandam maior precisão numérica e podem exigir um grande volume de memória.

Por outro lado, como apontado por Michalewicz (1996) [20], a codificação real oferece vantagens no que diz respeito à velocidade de processamento e à acurácia dos resultados, sendo ideal para problemas inseridos no contexto de números reais ou arranjos mais complexos.

Segundo Soares (1997) [26], a representação de uma população na codificação binária é feita como uma sequência de símbolos, normalmente utilizando o alfabeto binário (0 e 1). O tamanho da população é definido pelo contexto do problema e ajustado conforme necessário. O tamanho da população e o tamanho do indivíduo de uma população, em representação binária, pode ser representada na figura 3.

Por fim, a escolha da representação genética e do tamanho da população deve ser cuidadosamente alinhada ao problema, garantindo que o algoritmo seja eficiente e bem adaptado ao desafio proposto.

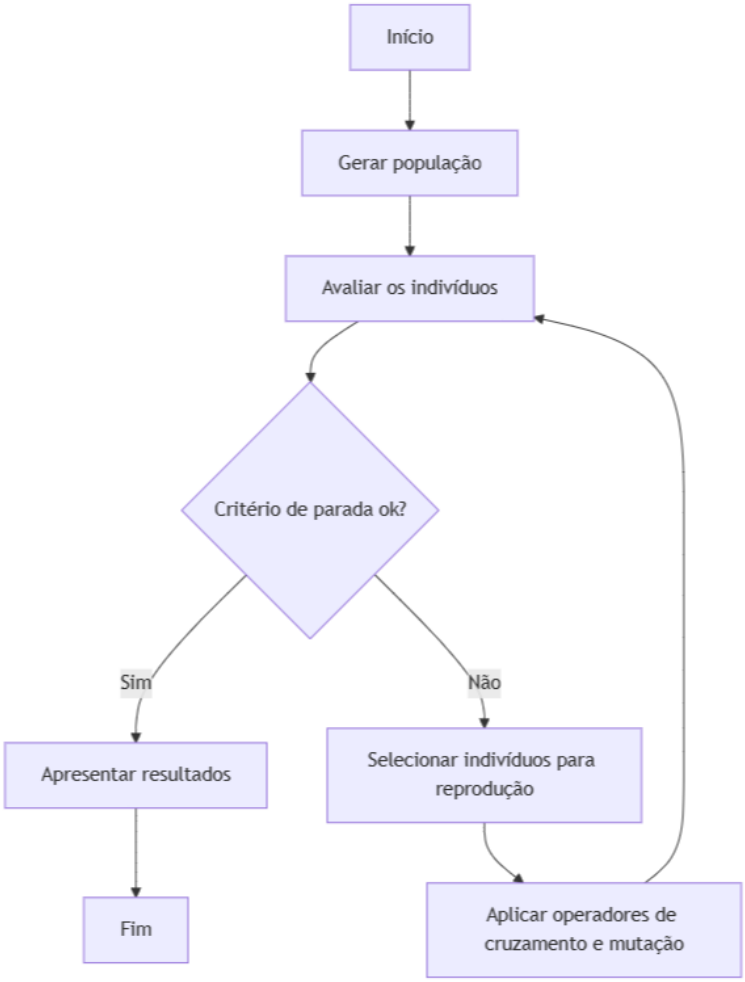


Figure 2: Fluxograma Para o AG Genérico.

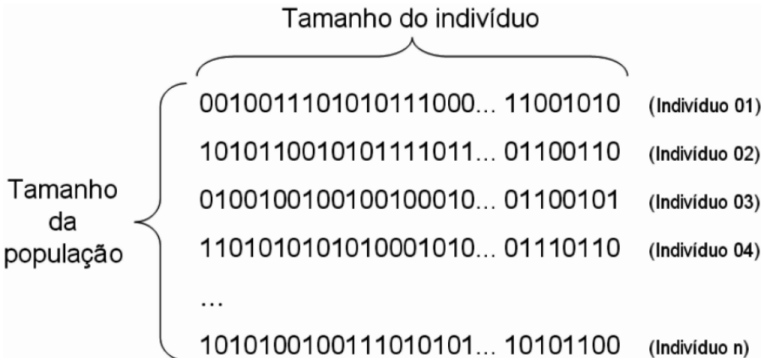


Figure 3: Tamanho da População vs. Tamanho do Indivíduo

Para Davis (1991) [27], o tamanho de um indivíduo em algoritmos genéticos pode ser representado pela quantidade de bits. Por exemplo, um indivíduo com 5 bits pode assumir 2^5, ou seja, 32 combinações diferentes.

Linden (2006) [21] complementa, destacando que a quantidade de bits necessária deve ser definida com base na variação entre os limites inferior e superior do problema, além do nível de precisão desejado.

Mitchell (1996) [28] propôs uma equação (Equação 1) que estabelece a relação entre esses fatores, facilitando a escolha do tamanho ideal para representar os indivíduos em função do problema abordado.

$$P = \frac{sup_i - inf_i}{2^k - 1} \tag{1}$$

Onde:

- P: Precisão;
- inf_i e sup_i: limites inferior e superior da faixa de operação, respectivamente;
- k: número de bits do indivíduo.

Na equação 2, é demonstrado um cálculo onde considera-se, por exemplo, uma variação no intervalo de $[-100, 100]$ e uma precisão de 0.0001. Neste caso, a quantidade de bits necessária para o indivíduo será de 21.

$$0,0001 = \frac{100 - (-100)}{2^k - 1} \longrightarrow k = 20,93 \longrightarrow k = 21 \tag{2}$$

É importante mencionar que é possível trabalhar tanto com valores reais quanto inteiros utilizando a codificação binária. Para converter para um valor real, primeiro é necessário identificar o número inteiro correspondente à codificação binária e, posteriormente, aplicar a operação descrita na Equação 3, conforme apresentada por Linden (2006) [21].

$$V_R = inf_i + \frac{sup_i - inf_i}{2^k - 1} * V_I \tag{3}$$

Onde:

- V_R: valor real;
- V_I: valor inteiro;
- inf_i e sup_i: limites inferior e superior da faixa de operação, respectivamente;
- k: número de bits do indivíduo.

2 População Inicial

De acordo com Soares (2006) [29], a população inicial em algoritmos genéticos pode ser formada de duas maneiras principais. A primeira opção é gerar os indivíduos de forma totalmente aleatória, sem seguir nenhum critério específico. A segunda opção envolve criar a população com base em uma abordagem heurística, ou seja, usando conhecimentos prévios sobre o problema para gerar indivíduos mais próximos de boas soluções ou soluções consideradas adequadas ou adotadas em problemas.

Reeves (1995) [30] explica que, ao usar heurísticas, é possível obter soluções melhores mais rapidamente em comparação com populações aleatórias. Por exemplo, ao se planejar uma rota de entregas em uma cidade. Ao se usar uma heurística, pode-se começar com trajetos que priorizem vias principais, evitando becos ou atalhos. Isso pode gerar soluções iniciais mais eficazes, alinhadas ao conceito de *exploitation* (Bellman, 1957) [31]. No entanto, essa abordagem tem um risco: o algoritmo pode se concentrar demais em um tipo de solução e acabar ”preso” em trajetos que parecem bons, mas não são os melhores possíveis (os chamados máximos/mínimos locais). Isso ocorre quando há excesso de *exploitation* sem considerar novas possibilidades.

Por outro lado, a geração aleatória inicial, que corresponde ao conceito de *exploration* (Bellman, 1957) [31], cria uma diversidade maior ao espalhar por todo o espaço de busca. Nesse mesmo exemplo de rotas, a população inicial incluiria trajetos variando de vias principais a atalhos estranhos. Apesar de muitas dessas opções serem ruins, a diversidade aumenta a chance de encontrar soluções realmente inovadoras ao longo das gerações.

O equilíbrio entre *exploration* e *exploitation* (Russel et. al, 2010 [32]) [8] é essencial para o sucesso de um algoritmo genético. Enquanto a exploration permite escapar de máximos ou mínimos locais, garantindo diversidade, a exploitation assegura que as regiões promissoras sejam exploradas ao máximo, otimizando os resultados.

Um ponto importante destacado por Reeves [30] é que a população inicial deve ser ampla o suficiente para cobrir boa parte do espaço de busca do problema, aumentando as chances de encontrar a melhor solução.

Tamanho da População: Torabi (2006) [33] ressalta que o tamanho da população inicial também desempenha um papel crucial. Uma população pequena pode limitar a diversidade e levar à convergência precoce, ou seja, o algoritmo pode chegar rapidamente a uma solução boa, mas longe da ideal. Por exemplo, ao se iniciar o planejamento de rotas com apenas três opções, todas podem seguir o mesmo padrão, dificultando encontrar algo realmente bom.

Por outro lado, uma população muito grande aumenta a diversidade, mas exige mais tempo e recursos computacionais. Ao se imaginar 10.000 rotas diferentes, sobrecarrega-se o processamento, tornando o algoritmo lento e menos eficiente.

Para equilibrar esses fatores, uma população inicial de tamanho moderado é ideal: grande o suficiente para garantir diversidade, mas não tão grande que torne o processamento inviável.

Portanto,

1. **Geração aleatória**: É como jogar vários dados e deixar que as combinações caiam ao acaso. As opções podem variar muito, desde boas soluções até soluções terríveis.
    - Exemplo: Gerar horários de aulas sem considerar restrições, como disponibilidade de professores ou salas.
2. **Geração heurística**: É como ter uma "regra básica" para começar. Por exemplo, garantir que professores só sejam alocados quando estão disponíveis.
    - Risco: Todas as soluções podem seguir o mesmo padrão e ignorar combinações alternativas melhores.

3 Função de Avaliação da População

De acordo com Koza (2003) [34], a função de avaliação desempenha um papel crucial nos algoritmos genéticos, pois reflete os objetivos que queremos alcançar na solução de um problema. Essa função é diretamente derivada das condições impostas pelo problema e define como medir a qualidade de cada solução (ou indivíduo) dentro da população.

Linden (2006) [21] reforça que a função de avaliação é o método utilizado pelo algoritmo genético para determinar o quão bem um indivíduo atende ao problema em questão, ou seja, sua aptidão. É ela que orienta o algoritmo sobre quais indivíduos devem ser preservados, reproduzidos ou descartados ao longo das gerações.

Barboza (2005) [22] acrescenta que a avaliação pode variar conforme o objetivo do problema. Se o objetivo for **maximizar** algo (como lucro ou eficiência), os indivíduos com valores maiores terão melhores avaliações. Por outro lado, se o objetivo for **minimizar** (como reduzir custos ou erros), os indivíduos com valores menores serão considerados melhores.

Por exemplo:

1. **Função Matemática Simples:**

   Imagine que se tenha a função $f(x) = \sin(5x) + \cos(3x)$ no intervalo $[0, 2\pi]$, e se quer encontrar o **máximo global**.

[8] Bellman [31] formulou o dilema de *exploration* (explorar novas opções para obter mais conhecimento) versus *exploitation* (usar as melhores opções já conhecidas) no contexto de processos de decisão sequenciais e o dilema é central na teoria do controle ótimo, aparecendo em problemas como o "multi-armed bandit problem" (problema dos bandidos de vários braços), onde um agente deve decidir entre explorar novos braços de uma máquina caça-níqueis ou continuar jogando em um braço que já deu boas recompensas. No entanto, foi no campo da IA e da otimização que esses termos ganharam tração.

Nesse caso, a função de avaliação pode ser diretamente $f(x)$, ou seja, cada indivíduo x é avaliado pelo valor da própria função. Indivíduos com valores maiores de $f(x)$ terão mais chances de "sobreviver".

Porém, no mundo real, esse método pode ser simplista. Imagine que se quer restringir a busca para $x \in [1, 5]$. Nesse caso, a função de avaliação poderia incluir uma penalidade para valores fora desse intervalo:

$$\text{Avaliação}(x) = f(x) - \text{Penalidade}(x)$$

Aqui, a penalidade seria zero dentro do intervalo e aumentaria para valores fora dele.

2. **Problema Real: Planejamento de Horários**

   No planejamento de horários para uma escola, a função de avaliação poderia considerar:

   - Minimizar conflitos entre professores;
   - Maximizar o uso eficiente das salas;
   - Organizar a distribuição das disciplinas considerando fatores pedagógicos e operacionais, como o nível de concentração dos alunos em diferentes horários do dia.

   Uma possível função de avaliação seria:

$$\text{Avaliação} = 1000 - (\text{Conflitos} \times 50) - (\text{Salas insuficientes} \times 100)$$

   Nesse caso, soluções com menos conflitos e maior ocupação das salas teriam melhores avaliações.

4 Métodos de Seleção

De acordo com Bäck (2000) [35], os métodos de seleção têm como objetivo direcionar o processo para explorar as regiões mais promissoras do espaço de busca. Hicks (2006) [36] explica que esses métodos são responsáveis por escolher os indivíduos que passarão pelos operadores genéticos, como recombinação e mutação. A seguir, serão apresentados alguns métodos de seleção que são utilizados em Algoritmos Genéticos.

4.1 Seleção por Roleta Simples

Este método, introduzido por Holland (1975) [13], utiliza um mecanismo inspirado em uma roleta para selecionar indivíduos com base em sua aptidão. Como explicado por Falcone (2004) [37], cada indivíduo na população tem sua chance de seleção proporcional ao seu valor de aptidão em relação ao total acumulado das aptidões. Em outras palavras, quanto maior a aptidão de um indivíduo, maior será o espaço que ele ocupa na roleta, e, consequentemente, maior será sua chance de ser escolhido. A roleta é "girada" repetidamente, e a cada giro um novo indivíduo é selecionado de forma aleatória, mas proporcional à sua contribuição na população.

No entanto, o método da roleta apresenta algumas limitações, como observado por Barboza (2005) [22]. Uma delas é a alta variância, que pode resultar na seleção repetida de indivíduos com aptidão muito elevada, diminuindo a diversidade da população. Essa falta de diversidade pode levar a uma convergência prematura, onde o algoritmo fica preso em uma solução local sem explorar outras possibilidades. Por outro lado, quando o processo evolutivo já está avançado, o método pode causar uma estagnação no algoritmo, dificultando a busca por soluções mais refinadas.

Exemplo: Para uma população com três indivíduos A, B e C, com aptidões 60, 30 e 10, respectivamente. O total de aptidão é $60 + 30 + 10 = 100$.

- A ocupa 60%, B ocupa 30% e C ocupa 10% e a imagem 4 nos garante um estímulo visual deste exemplo;
- Se a roleta for girada, A terá maior probabilidade de ser selecionado, mas B e C ainda têm chances, embora menores.

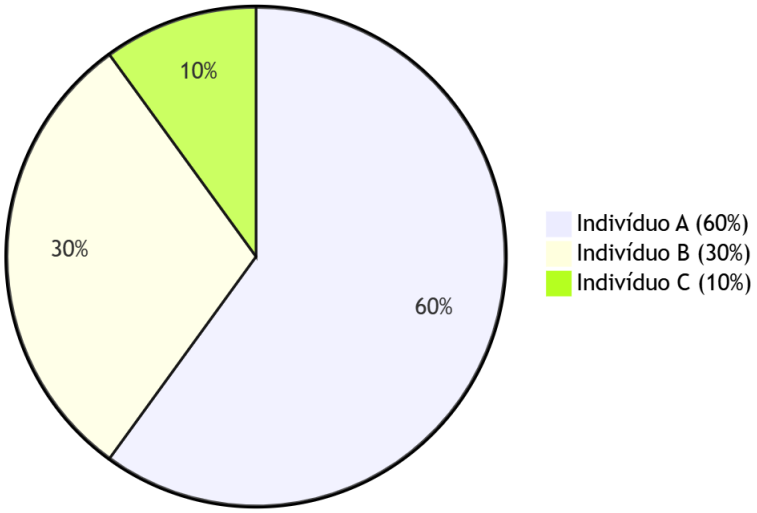


Figure 4: Exemplo de Roleta Simples

Essa mecânica garante que indivíduos mais aptos sejam privilegiados, mas ainda permite uma pequena oportunidade para aqueles com menor aptidão contribuírem para a próxima geração, promovendo diversidade até certo ponto.

4.2 Seleção por Torneio

Segundo Teles e Gomes (2010) [38], a seleção por torneio é uma técnica amplamente utilizada em algoritmos genéticos, especialmente por sua eficiência e facilidade de implementação. Este método contribui para manter a diversidade da população, pois a seleção é realizada comparando o valor de **aptidão** (calculado com base na função objetivo) entre um subconjunto aleatório de indivíduos, chamados de competidores.

Linden (2006) [21] destaca que esse método utiliza um parâmetro chamado tamanho do torneio (K), que determina quantos indivíduos serão escolhidos aleatoriamente da população para competir. Dentro deste grupo, o indivíduo com a melhor aptidão é selecionado para reprodução. Contudo, um aumento no tamanho do torneio pode levar a uma redução na diversidade, já que indivíduos mais aptos tendem a dominar.

Uma das principais vantagens desse método é que os competidores são selecionados de forma aleatória, sem priorizar diretamente os indivíduos mais aptos da população geral, como ocorre na seleção por roleta. Isso promove um equilíbrio entre exploração e diversidade, ajudando a evitar a convergência precoce para soluções locais (Linden, 2006) [21]. A imagem 5 ilustra a lógica da seleção por torneio, onde temos o seguinte fluxo.

1. **População Geral**: Todos os indivíduos disponíveis no algoritmo genético.
2. **Selecionar K Indivíduos Aleatórios**: Um subconjunto da população é escolhido de forma aleatória. O tamanho deste grupo é definido pelo parâmetro K.
3. **Comparar Funções de Avaliação**: Avaliam-se os indivíduos do grupo com base na função de avaliação.
4. **Selecionar o Mais Apto**: O indivíduo com melhor desempenho é escolhido.
5. **Indivíduo Escolhido**: Este indivíduo é enviado para a próxima etapa do algoritmo (como reprodução ou mutação).

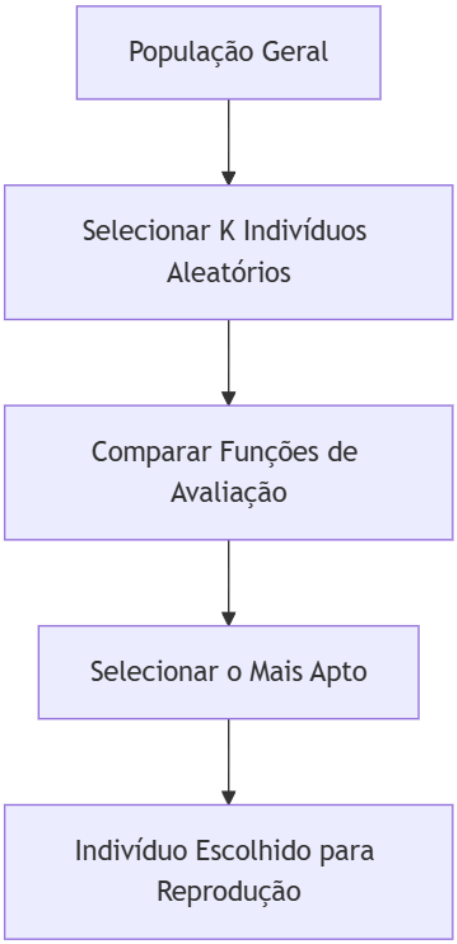


Figure 5: Fluxograma da Lógica da Seleção por Torneio

4.3 Seleção por Elitismo

O elitismo, embora não seja um método de seleção por si só, é frequentemente usada em conjunto com outros métodos para melhorar a eficiência dos algoritmos genéticos. Segundo Barboza (2005) [22], essa técnica garante que os indivíduos mais bem avaliados de uma geração sejam preservados para a próxima, evitando a perda de soluções de alta qualidade ao longo do processo evolutivo.

A maioria dos métodos tradicionais de seleção substitui completamente a geração anterior pelos descendentes, de acordo com Bento e Kagan (2008) [39], o que pode resultar na perda de informações valiosas de indivíduos altamente aptos. A seleção elitista resolve esse problema ao reintroduzir na nova geração os melhores indivíduos da anterior, assegurando que características vantajosas sejam mantidas.

Além disso, a técnica pode armazenar o melhor resultado encontrado durante toda a evolução, permitindo que, mesmo que esse indivíduo não esteja presente na última geração, ele seja considerado como a melhor solução final do processo (Barboza, 2005) [22]. A imagem 6 representa o fluxograma com a lógica elitista de seleção.

1. **População Atual**: Todos os indivíduos da geração atual.
2. **Selecionar os Melhores Indivíduos**: Identificam-se os indivíduos mais aptos da geração.
3. **Reintroduzir na Nova Geração**: Esses indivíduos são diretamente adicionados à próxima geração.

4. **Gerar Novos Descendentes**: Os operadores genéticos (como cruzamento e mutação) criam novos indivíduos.
5. **Combinar com Indivíduos Elitistas**: A nova geração é composta pelos indivíduos elitistas e pelos novos descendentes.
6. **Nova Geração Pronta**: A população está pronta para a próxima iteração do algoritmo genético.

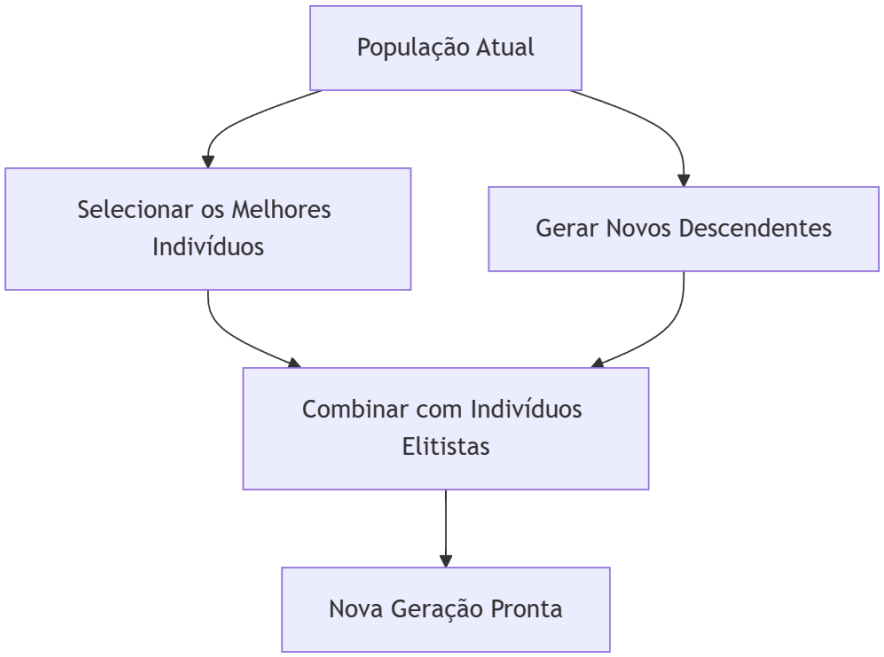


Figure 6: Fluxograma representativo da lógica de seleção elitista

Portanto, o elitismo garante que as melhores soluções sejam preservadas, mesmo que os operadores genéticos não as recriem. Este tipo de seleção também promove mais desempenho, pois aumenta as chances de manter um progresso consistente em direção a soluções melhores. A seleção elitista proporciona chance de maior convergência, pois evita que boas soluções sejam descartadas, ajudando o algoritmo a atingir um resultado final mais consistente.

4.4 Normalização Linear

De acordo com Linden [21], a normalização linear é considerada um mecanismo auxiliar da função de avaliação, uma vez que seu objetivo principal, segundo o autor, é ajustar os valores de aptidão (*fitness*) para que sejam utilizados de forma mais eficiente pelo algoritmo genético. Esse método organiza os valores de aptidão em uma escala predefinida (como K e t), facilitando a priorização de certos indivíduos durante o processo de avaliação. Assim, na visão de Linden [21], a normalização linear está intimamente relacionada ao ajuste dos valores calculados pela função de avaliação, e não ao processo de seleção em si.

Por outro lado, Pacheco (2004) [40] entende a normalização linear como parte integrante do processo de seleção, pois ela desempenha um papel central na definição de como os indivíduos serão escolhidos para reprodução. Então, a normalização linear determina a chamada ”pressão seletiva”, ou seja, o grau de influência da aptidão de um indivíduo na sua probabilidade de ser selecionado. Sob essa perspectiva, a normalização linear atua como um mecanismo para equilibrar as chances de seleção, evitando tanto a dominância de superindivíduos quanto a concentração seletiva que poderia reduzir a diversidade da população.

Ambas as interpretações têm mérito, e a categorização da normalização linear depende do foco de análise no contexto do estudo:

- Quando o objetivo é ajustar os valores antes de qualquer operação de seleção, faz sentido classificá-la como parte da função de avaliação.

- Já se o interesse está em como os indivíduos são efetivamente escolhidos para reprodução, ela pode ser considerada um método de seleção.

Embora a normalização linear possa ser vista como uma ferramenta ambígua, seu papel fundamental é servir como uma ponte entre a avaliação de aptidão e o processo de seleção. No entanto, para este projeto, escolheu-se por classificá-la como um **método de seleção**, dado que seu impacto direto está na forma como os indivíduos são escolhidos para continuar no processo evolutivo. Essa decisão reflete a ênfase do projeto no impacto da normalização sobre a reprodução e o desempenho global do algoritmo genético. De acordo com Pacheco (2004) [40], a normalização linear é determinada por alguns passos:

1. **Ordenação dos indivíduos por aptidão:**
   Inicialmente, os indivíduos são organizados em ordem decrescente de aptidão. O indivíduo com maior aptidão fica no topo da lista, enquanto o de menor aptidão ocupa a última posição.
2. **Definição de valores extremos:**
   O indivíduo mais apto recebe o valor de aptidão máximo, denominado máx, enquanto o indivíduo menos apto é atribuído ao valor mínimo, chamado mín. Esses valores podem ser ajustados pelo usuário. Segundo Pacheco, recomenda-se que máx ≥ 2 e mín ≥ 0 para garantir uma distribuição eficiente dos valores.
3. **Cálculo da aptidão dos indivíduos intermediários:**
   Os valores de aptidão dos demais indivíduos são distribuídos linearmente entre mín e máx, com base na posição relativa de cada indivíduo na ordenação. A fórmula utilizada é dada pela equação 4.

$$A_i = \text{mín} + \frac{(\text{máx} - \text{mín})}{n-1} \times (i-1) \tag{4}$$

   onde:

   - A_i é o valor de aptidão do indivíduo i.
   - n é o número total de indivíduos na população.
   - i é a posição do indivíduo na ordenação (1 para o pior indivíduo e n para o melhor).

A intensidade da seleção, ainda de acordo com Pacheco (2004) [40] é uma métrica que avalia o impacto da normalização linear na pressão seletiva sobre a população. É definida pela equação 5.

$$I = (1 - \text{mín}) \cdot \frac{1}{\sqrt{\pi}} \tag{5}$$

Essa expressão mostra que, à medida que o valor de mín aumenta, a intensidade da seleção diminui. Isso significa que, ao elevar mín, reduz-se as diferenças entre as aptidões dos indivíduos, promovendo uma competição mais equilibrada.

5 Operadores Genéticos

Existe uma variedade de operadores genéticos, de acordo com Goldberg (1989) [2], no entanto, *crossover* (cruzamento) e mutação são os mais comuns de serem utilizados.

5.1 *Crossover* (Cruzamento)

Conforme Konak (2006) [34], o *crossover* é uma das partes mais importantes dos algoritmos genéticos. Nesse processo, dois indivíduos da população são combinados para formar novos indivíduos. Ao aplicar repetidamente esse operador, espera-se que as melhores características dos indivíduos sejam transmitidas com maior frequência, permitindo que a população evolua em direção a soluções melhores.

Na operação de cruzamento, um ponto aleatório no indivíduo é escolhido, onde ele é "quebrado". Os segmentos resultantes são então recombinados com os fragmentos de outro indivíduo, criando dois novos indivíduos. Em variações desse método, o cruzamento pode ocorrer em mais de um ponto ou até ser distribuído de forma mais elaborada, combinando múltiplas partes de diferentes indivíduos (Azadivar Tompkins, 1999) [41].

Existem diferentes tipos de cruzamento, que variam conforme a escolha do ponto de troca (ou *locus*) e a maneira como os genes são misturados entre os pais. Abaixo estão alguns dos métodos mais utilizados:

- ***Crossover* Uniforme**: Este método realiza o pareamento entre dois indivíduos, onde cada ponto (*locus*) tem uma chance de 50% de ser trocado. Por exemplo, imagine um indivíduo com 8 pontos genéticos (ou *loci*). Em um cruzamento uniforme, pontos como o primeiro, quarto e quinto podem ser trocados aleatoriamente entre os pais, resultando em uma combinação única de características dos dois indivíduos (Linden, 2006) [21]. Pode-se observar um exemplo na imagem 130, onde um indivíduo possui 8 (oito) lócus e sofreu cruzamento no primeiro, quarto e quinto lócus.

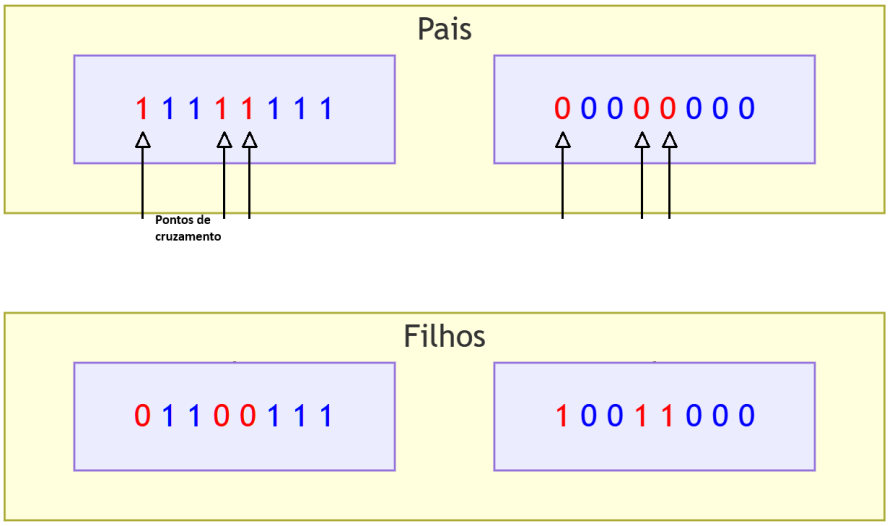


Figure 7: Exemplo de Crossover Uniforme

- ***Crossover* de Um Ponto**: Nesse caso, um único ponto de corte é selecionado de maneira aleatória em ambos os pais previamente selecionados. A partir desse ponto, os genes do primeiro pai são transferidos para o filho até o ponto de corte, enquanto os genes após o corte vêm do segundo pai. Este método é mais simples e bastante utilizado. Por exemplo, ao cruzar dois indivíduos, o filho pode herdar o início do código genético de um pai e o final do outro (Barboza, 2005) [22]. A figura 8 exemplifica este tipo de *crossover*.

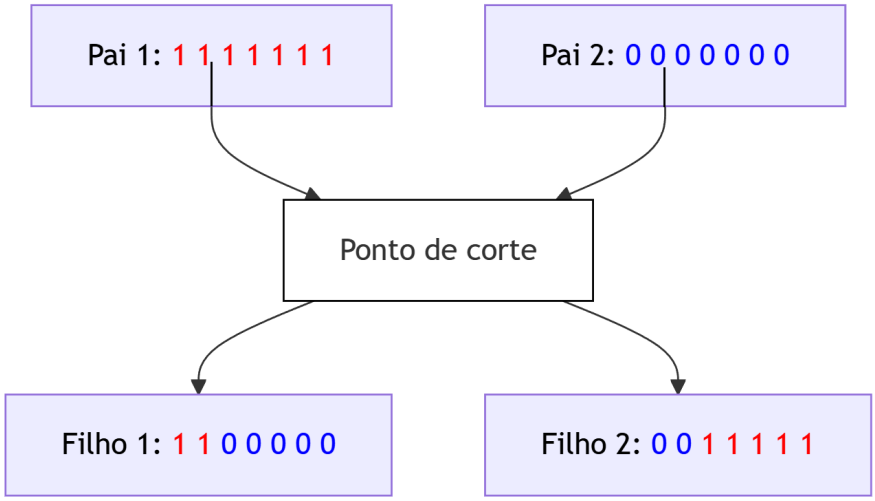


Figure 8: Exemplo de Crossover de Um Ponto.

- ***Crossover* de dois pontos**: Nesse método, dois pontos são escolhidos de forma aleatória no material genético dos pais. Entre esses dois pontos, as informações genéticas são trocadas entre os pais, gerando os novos filhos. Já as partes do material genético que estão fora desses limites permanecem inalteradas, garantindo que apenas uma porção específica seja recombinada. Esse processo possibilita maior diversidade na geração de descendentes, ao mesmo tempo que preserva características importantes dos pais. A Figura 5 ilustra um exemplo prático de como esse cruzamento ocorre na prática (Linden, 2006) [21]. A figura 9 exemplifica este tipo de *crossover*.

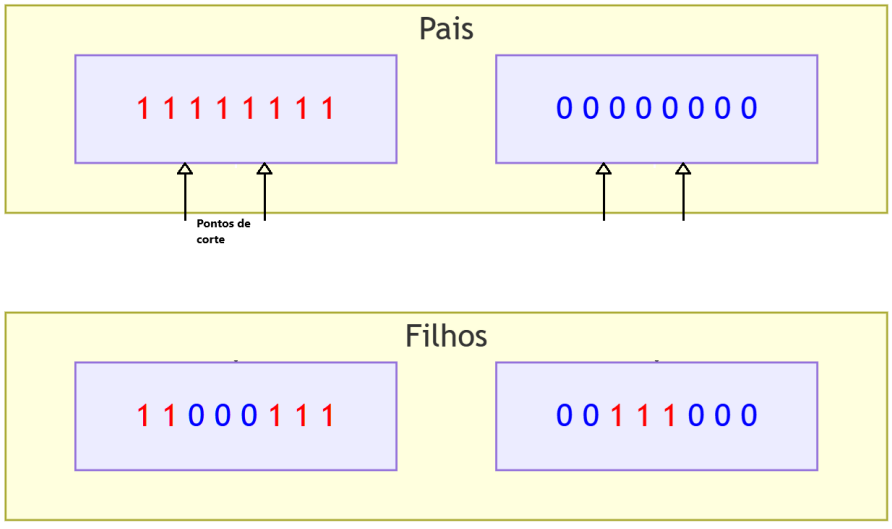


Figure 9: Exemplo de Crossover de Dois Pontos.

5.2 Mutação

O operador de mutação realiza mudanças aleatórias em características específicas dos indivíduos. Essas alterações acontecem a nível de genes (que pode ser binário ou real), e os genes alterados por esse operador costumam alterar pouco o filho produzido por crossover a partir dos pais. Contudo, a mutação desempenha um papel essencial ao reintroduzir diversidade genética na população, ajudando o algoritmo a escapar de máximos ou mínimos locais (Konak, 2006) [42].

No caso da codificação binária, o tipo de mutação mais comum é a mutação binária simples. Nesse processo, algumas posições dos indivíduos são selecionadas com base na *taxa de mu-*

tação, que define a probabilidade de cada gene ser alterado. Os valores dos genes nas posições escolhidas são então invertidos. Por exemplo, se um gene possui o valor 1, ele será alterado para 0, e vice-versa (Mitchell, 1996). A Figura 10 apresenta um exemplo prático desse tipo de mutação em ação.

Figure 10: Exemplo mutação binária simples.

6 Técnicas de Reprodução

Para Pacheco (2004) [40], As técnicas, os parâmetros e os tipos de operadores genéticos desempenham um papel crucial no desempenho de um algoritmo genético. Em termos gerais, esses algoritmos são altamente sensíveis às escolhas feitas em relação à combinação e aplicação dessas técnicas. A definição de quais parâmetros e operadores serão utilizados é, em sua essência, um processo empírico, mas deve estar alinhada com as características específicas do problema a ser resolvido, garantindo uma abordagem otimizada e eficaz.

As técnicas de reprodução definem os critérios para a substituição dos indivíduos de uma população na próxima geração. Basicamente, podem ser classificadas nos seguintes métodos: troca de toda população, troca de toda população com elitismo, *steady state* e *steady state* sem duplicados.

6.1 Troca de Toda População

Neste método, de acordo com Pacheco (2004) [40] a cada ciclo evolutivo, a população inteira é substituída por novos indivíduos. Para isso, $N/2$ pares de indivíduos da geração atual são selecionados para o cruzamento, gerando N descendentes que compõem a nova geração. Essa abordagem é amplamente utilizada devido à sua simplicidade, mas apresenta um risco: a perda de informações genéticas importantes, já que todos os indivíduos da geração anterior são descartados. Isso pode levar a uma diminuição na diversidade genética e a uma possível estagnação do processo evolutivo em certos cenários.

6.2 Troca de Toda População Com Elitismo

Essa variação busca contornar o problema da perda completa de informações genéticas, segundo Pacheco (2004) [40]. Enquanto todos os cromossomos são substituídos, os indivíduos mais aptos da geração atual (preservar 2 indivíduos é uma escolha prática para manter a paridade) são preservados e diretamente copiados para a nova geração. Esse processo, conhecido como elitismo, garante que as melhores soluções encontradas até o momento sejam mantidas no processo evolutivo.

Como resultado, o método equilibra a introdução de novos indivíduos com a preservação das características mais vantajosas, aumentando a eficiência do algoritmo genético e diminuindo a probabilidade de perda de soluções promissoras.

6.3 Steady State

Linden (200) [21] descreve o steady state como uma técnica inspirada no comportamento biológico natural, onde gerações coexistem e se sobrepõem, permitindo que indivíduos melhores da geração k sejam preservados e reproduzam com novos indivíduos da geração $k+1$. Segundo Linden [21], essa abordagem evita a substituição completa de uma geração por outra,

característica dos métodos de renovação total, e foca na continuidade evolutiva, além de enfatizar que há riscos associados, como a convergência prematura devido à falta de diversidade genética.

Goldberg [2], por outro lado, introduz o conceito de GAP no *steady state*, que define o número de indivíduos substituídos por ciclo ($M < N$), o que torna a técnica mais flexível. Portanto, é uma técnica elitista que permite maior estabilidade na população e a utilização de operadores de cruzamento menos conservadores, como o *crossover* uniforme.

6.4 Steady State Sem Duplicados

O conceito de *steady state sem duplicados* foi originalmente introduzido por Kenneth De Jong em sua tese de doutorado intitulada *"An Analysis of the Behavior of a Class of Genetic Adaptive Systems"* [43], onde propôs evitar duplicações na população para garantir maior diversidade genética e evitar convergência prematura.

O steady state sem duplicatas apresenta uma variação importante, onde duplicatas são evitadas, garantindo maior diversidade genética e aproveitamento do paralelismo intrínseco dos GAs. Pacheco (2004) [40] destaca uma aplicação prática dessa variação, apontando que, embora seja eficiente para manter a diversidade, pode gerar um custo adicional de processamento para detectar e excluir duplicatas.

7 Critérios de Parada

Os critérios de parada são estratégias que determinam quando o processo de busca de um Algoritmo Genético (AG) deve ser encerrado. De acordo com Hicks (2006) [36], isso geralmente ocorre após um número predeterminado de gerações ou pode ser baseado em condições específicas. Por exemplo, o algoritmo pode ser interrompido ao atingir um tempo máximo definido, quando houver uma redução significativa na diversidade da população ou quando não houver melhorias nos indivíduos da população ao longo de várias gerações consecutivas. Essas abordagens garantem que o algoritmo não continue executando desnecessariamente, economizando tempo e recursos.

8 Parâmetros dos AGs

Conforme apontado por Yang. (2007) [44], os Algoritmos Genéticos são ferramentas poderosas para resolver problemas de otimização, mas sua eficácia depende significativamente de como seus principais parâmetros são configurados. Entre os parâmetros mais críticos estão o tamanho da população, o número de gerações, a taxa de cruzamento e a taxa de mutação. Esses elementos influenciam diretamente a capacidade do algoritmo de explorar e refinar soluções ao longo do tempo.

Neppalli (1996) [45] destaca que a definição desses parâmetros é um processo desafiador, pois envolve um equilíbrio delicado entre exploração e explotação. Geralmente, os valores utilizados são ajustados com base em estudos anteriores ou por meio de experimentação, levando em conta a natureza específica do problema.

Este tópico é, sem dúvida, um dos mais relevantes quando se discute a execução e a performance de Algoritmos Genéticos. Configurações inadequadas podem levar a resultados insatisfatórios, como convergência prematura ou alto custo computacional, enquanto uma configuração otimizada pode aumentar significativamente a eficácia do algoritmo em encontrar soluções de alta qualidade.

8.1 Tamanho da População

O número de indivíduos na população tem impacto direto no desempenho e na eficiência de um Algoritmo Genético, como destaca Barboza (2005) [22]. Um tamanho de população muito pequeno pode limitar a diversidade genética, reduzindo a capacidade do algoritmo de explorar diferentes soluções. Por outro lado, populações excessivamente grandes podem aumentar o custo computacional, tornando o processo de otimização mais lento.

De acordo com experimentos conduzidos por De Jong (1975), conforme citado por Mitchell (1996) [28], uma faixa ideal para o tamanho da população está entre 50 e 100 indivíduos. Essa faixa é amplamente aceita e frequentemente utilizada em aplicações práticas de AG, pois tende a equilibrar diversidade e custo computacional de forma eficiente. Assim, definir um tamanho de população adequado é essencial para garantir o equilíbrio entre a exploração do espaço de soluções e a velocidade de convergência do algoritmo.

8.2 Número de Gerações

O número de gerações em um Algoritmo Genético depende diretamente da complexidade do problema em questão e geralmente precisa ser ajustado por experimentação. Segundo Yun e Gen (2003) [46], como o objetivo principal do AG é resolver problemas de otimização, o ideal seria que o algoritmo fosse executado até que a melhor solução possível fosse encontrada.

Entretanto, há divergências entre os pesquisadores sobre qual seria o número ideal de gerações. Goldberg (1989) [2] sugere uma abordagem interessante baseada no desvio padrão dos valores de aptidão da população. Nesse método, compara-se o desempenho da geração atual com o da anterior. Se o desvio padrão for igual ou menor que um valor predefinido (indicando pouca variação ou progresso), o processo é encerrado. Essa estratégia implica que o número total de gerações não é pré-determinado, mas descoberto durante a execução do algoritmo, quando um critério de parada é atingido.

8.3 Taxa de *Crossover* (Cruzamento)

A taxa de cruzamento define a probabilidade de ocorrer a troca de material genético entre dois indivíduos dentro de uma população. Esse processo é determinado gerando um número aleatório entre 0 e 1, e, caso esse número seja inferior à taxa estabelecida, o cruzamento é realizado.

De forma prática, observa-se que uma taxa de cruzamento mais elevada facilita a introdução de novos indivíduos na população, acelerando a diversidade genética. No entanto, taxas muito altas podem levar à substituição rápida de grande parte da população, resultando na perda de indivíduos com alta aptidão e comprometendo o progresso do algoritmo. Por outro lado, se a taxa for muito baixa, a evolução do processo pode se tornar extremamente lenta, dificultando a busca por soluções melhores.

De Jong (1975), citado por Mitchell (1996) [20], sugere uma taxa de cruzamento ideal em torno de 0,6. Já Tanomaru (1995) [47] recomenda taxas um pouco mais altas, superiores a 0,7, dependendo da aplicação.

8.4 Taxa de Mutação

A taxa de mutação define a probabilidade de um gene sofrer alteração em cada indivíduo. Esse processo é realizado gerando um número aleatório entre 0 e 1 para cada gene, e, caso esse número seja inferior à taxa de mutação estabelecida, a mutação é aplicada.

No que diz respeito aos valores dessa taxa, percebe-se que uma taxa de mutação muito baixa pode fazer com que a busca fique presa em um valor, como um ótimo local, sem explorar adequadamente outras possibilidades. Por outro lado, uma taxa muito alta torna o processo essencialmente aleatório, dificultando o progresso consistente do algoritmo.

De Jong (1975) [43], citado por Mitchell (1996) [28], sugere um valor ideal de taxa de mutação de 0,001, enquanto Tanomaru (1995) recomenda taxas inferiores a 0,01, dependendo do contexto.

e Personalização dos Algoritmos Genéticos: A Flexibilidade na Definição das Características

Com base nos conteúdos que foram profundamente estudados e abordados, de forma a harmonizar o ponto de vista de diversos cientistas que são referência neste campo, pode-se inferir que os Algoritmos Genéticos se destacam pela sua flexibilidade e capacidade de adaptação a uma ampla gama de problemas. Cada uma das etapas dentro do AG — desde a geração da

população inicial até a seleção e substituição de indivíduos — pode ser personalizada de acordo com os objetivos e características do problema em questão. Isso permite ao pesquisador ou designer do algoritmo uma grande liberdade criativa para ajustar o comportamento do AG, equilibrando diferentes forças como exploração e explotação (Goldberg, 19089 [2]) no espaço de soluções.

1 Seleção e Reprodução: O Equilíbrio Essencial

1. Métodos de Seleção:
   - A seleção é o processo pelo qual os indivíduos mais aptos são escolhidos para gerar a próxima geração. O designer do Algoritmo Genético escolhe a técnica de seleção com base no equilíbrio desejado entre exploração (buscar novas soluções) e explotação (refinar soluções já conhecidas). Como sabemos, os métodos comuns são:
     - **Roleta Simples**, onde os indivíduos têm uma chance de ser selecionados proporcional à sua aptidão.
     - **Seleção por Torneio**, que permite selecionar o melhor de um grupo de indivíduos aleatórios.
     - **Normalização Linear**, que ajusta as aptidões para equilibrar a pressão seletiva e impedir que indivíduos extremamente aptos dominem a seleção. Não é exatamente uma técnica de seleção, mas sim uma técnica de pré-processamento que antecede a o processo de seleção.
2. Métodos de Crossover e Mutação:
   - Aqui, há as operações de **reprodução** que garantem a criação de novos indivíduos a partir dos existentes. O designer deve decidir as taxas e os tipos de *crossover* e mutação que equilibram a diversificação e a convergência do Algoritmo Genético.
     - Crossover pode ser de 1 ponto, 2 pontos ou uniforme, com a escolha impactando a quantidade de exploração realizada no espaço de soluções.
     - Mutação deve ser aplicada de forma controlada. Uma taxa de mutação muito alta pode desestabilizar o processo, enquanto uma taxa muito baixa pode fazer o algoritmo se estagnar rapidamente em ótimos locais.

2 Elitismo: Seleção ou Reprodução?

O elitismo, ao que se percebe, é uma das técnicas mais debatidas no contexto dos Algoritmos Genéticos. Sua natureza dual — como parte da seleção e da reprodução — exige um olhar atento sobre como ele é implementado.

- **Elitismo como Seleção**: Como um componente de seleção, o elitismo preserva os melhores indivíduos de cada geração. Ele garante que as melhores soluções, baseadas na avaliação de aptidão, sejam preservadas e copiada para a próxima geração, sem modificação. Assim, ele assegura que boas soluções encontradas ao longo do processo de evolução não sejam perdidas. O elitismo, nesse sentido, age como uma técnica de seleção ao determinar quais indivíduos da geração atual devem ser preservados.
- **Elitismo como Reprodução**: Quando considerado no contexto de reprodução, o elitismo garante que certos indivíduos passem para a próxima geração sem sofrerem alteração (sem recombinação ou mutação). Isso implica que, em vez de seguir o processo de recombinação, esses indivíduos mais aptos são reproduzidos sem alteração. Portanto, embora o elitismo envolva uma seleção, sua aplicação direta à nova geração o torna um mecanismo importante na reprodução.

O elitismo, embora esteja mais alinhado às definições de reprodução, desempenha o papel de um complemento à seleção. Sua função principal é atuar na transição entre gerações, assegurando que os indivíduos mais qualificados sejam preservados diretamente na nova população.

3 *Steady State*: Seleção ou Reprodução?

A técnica de *steady state* também pode ser vista tanto como uma técnica de seleção quanto de reprodução, dependendo de sua implementação.

- ***Steady State* como Seleção**: No contexto de *steady state*, a seleção determina quais indivíduos serão mantidos e quais serão substituídos, mas o foco principal está em substituir parcialmente a população ao longo das gerações. Isso significa que a seleção é usada para escolher quais indivíduos serão preservados e quais serão substituídos pelos novos descendentes. Contudo, o *steady state* não se limita a essa etapa de seleção, pois a substituição de indivíduos é o principal foco.
- ***Steady State* como Reprodução**: A substituição parcial da população — em vez de uma substituição completa, como acontece na abordagem geracional — é uma característica intrínseca do *steady state*. O *steady state* é, portanto, uma técnica de reprodução, pois ele define como os indivíduos da geração anterior são substituídos por descendentes. Isso acontece de forma gradual, o que ajuda a manter o equilíbrio entre exploração e explotação (Goldberg,1989 [2]) ao longo do tempo.

4 A Flexibilidade do Designer do Algoritmo Genético

No final, a grande vantagem dos Algoritmos Genéticos está em sua flexibilidade. O pesquisador ou desenvolvedor tem a liberdade de ajustar os parâmetros e definir como os mecanismos de seleção, reprodução e substituição se aplicam ao problema específico que está sendo abordado. As decisões que envolvem o elitismo, o *steady state*, a taxa de *crossover* e mutação, entre outros parâmetros, são feitas com base em:

- **Objetivo do Problema**: Dependendo da natureza do problema, o designer pode optar por técnicas de seleção mais exploratórias (como torneio ou roleta) ou técnicas mais focadas na exploração de soluções de alta aptidão (como normalização linear), podendo complementar o processo com estratégias de elitismo quando necessário.
- **Diversidade e Estabilidade**: O designer também deve considerar a diversidade da população ao longo das gerações. Se for importante evitar a convergência prematura, ele pode optar por métodos que incentivem mais diversidade (como steady state sem duplicados ou usar taxas de mutação mais altas).
- **Eficiência Computacional**: Para problemas com grandes espaços de busca ou restrições de tempo, o steady state pode ser preferido, pois ele evita o custo de recalcular a aptidão de toda a população a cada geração.

f Aplicações dos Algoritmos Genéticos

Os Algoritmos Genéticos têm sido amplamente aplicados em diversas áreas de pesquisa ao longo dos anos, oferecendo resultados promissores e, muitas vezes, inovadores. A seguir, são apresentadas algumas dessas aplicações.

1 Otimização de Funções Matemáticas

Este item tem como objetivo principal demonstrar a aplicação de um Algoritmo Genético (AG) na otimização de funções matemáticas. Além disso, busca ilustrar, de maneira prática, os procedimentos adotados pelo AG por meio do exemplo apresentado.

A função matemática escolhida para otimização (ou maximização) é definida como $f(x) = x\,\mathrm{sin}(10\pi x) + 1$, conforme retirada do trabalho de [48]. Essa função é analisada no intervalo $-1 \leq x \leq 2$, e possui um ponto máximo global conhecido, localizado em $x = 1,85055$, onde $f(x) = 2,85027$.

Embora, à primeira vista, essa função possa parecer simples, sua solução não é trivial. Isso ocorre porque ela apresenta diversos máximos locais, além de apenas um máximo global. Essa característica torna o problema desafiador, como pode ser visualizado na Figura 11, que ilustra os comportamentos da função.

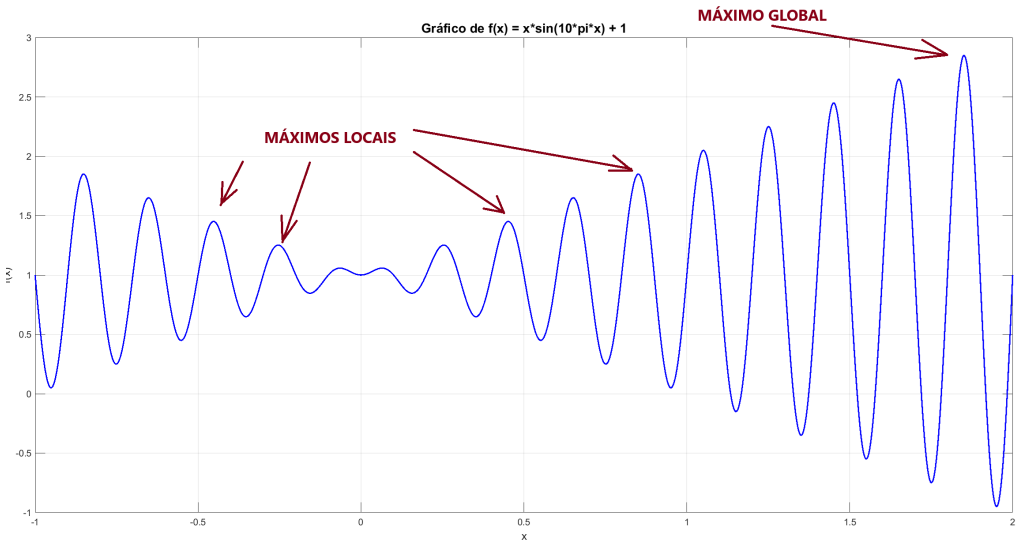


Figure 11: Gráfico da função $f(x) = x\,\mathrm{sin}(10\pi x) + 1$, mostrando os máximos locais e o máximo global.

Para iniciar um modelo de algoritmo genético, primeiro precisa-se definir como representar a variável x. Como se quer uma precisão bem refinada de três casas decimais, precisa-se usar indivíduos com 12 bits, como pode-se observar por meio da Equação 6.

$$0,001 = \frac{2-(-1)}{2^k - 1} \Rightarrow k = 11,55 \Rightarrow k = 12 \tag{6}$$

No caso de representações baseadas em variáveis reais, o espaço de representação é automaticamente gerenciado pelo computador, utilizando a precisão da arquitetura computacional, como ponto flutuante. Por outro lado, em representações binárias, é necessário calcular manualmente o número de *bits* necessários para garantir a precisão desejada.

Depois que define-se como a variável x será representada, pode-se dar início a ciclo do algoritmo genético! O fluxograma da Figura 2, indica que o primeiro passo é criar a população inicial e, em seguida, avaliar cada indivíduo dessa população.

1. **Inicialização dos Parâmetros**:
   - Função: $f(x) = x\,\mathrm{sin}(10\pi x) + 1$;
   - Objetivo: maximizar;
   - Tamanho da População: 3 indivíduos;
   - Número de Gerações: 2;
   - Intervalo de Busca: $[-1, 2]$;
   - Técnica de reprodução: elitismo (preservar 1 indivíduo por geração);
   - Tipo de crossover: 1 ponto;
   - Taxa de Crossover: 60%;
   - Taxa de Mutação: 65%.

2. **Primeira Geração**:
   (a) **Avaliação da População**: Os indivíduos gerados inicialmente, seus valores de x e avaliações ($f(x)$) estão apresentados na Tabela 2. Cada indivíduo é avaliado calculando $f(x)$ com base no valor de x representado pelos cromossomos binários.

Indivíduo	**Valor de x**	**Avaliação**
001001110101	0,46081	1,11520
100011110010	1,67766	2,33463
110011010011	2,40513	3,32914

Table 2: Indivíduos gerados e suas respectivas avaliações.

(b) **Elitismo**: O melhor indivíduo da geração é identificado com base na maior avaliação. Nesse caso, o indivíduo 110011010011 (com avaliação $f(x) = 3,32914$) é preservado e será automaticamente incluído na próxima geração.

(c) **Seleção (Método da Roleta)**: A seleção é realizada proporcionalmente às avaliações. O total acumulado das avaliações é calculado:

$$\text{Soma Total} = 1,11520 + 2,33463 + 3,32914 = 6,77997.$$

As probabilidades de seleção são então determinadas:

$$\text{Probabilidades:} \begin{cases} \text{Indivíduo 1: } \frac{1,11520}{6,77997} \approx 0,1645, \\ \text{Indivíduo 2: } \frac{2,33463}{6,77997} \approx 0,3444, \\ \text{Indivíduo 3: } \frac{3,32914}{6,77997} \approx 0,4911. \end{cases}$$

Dois indivíduos são selecionados para crossover com base nessas probabilidades (suponhamos que os indivíduos 2 e 3 sejam selecionados neste exemplo).

(d) **Crossover (1 ponto)**: O crossover é realizado entre os indivíduos selecionados (2 e 3). Suponhamos que o ponto de crossover seja o 7º bit:

$$\text{Indivíduo 2: } 100011\mathbf{110010}, \quad \text{Indivíduo 3: } 110011\mathbf{010011}.$$

Após o crossover:

$$\text{Filho 1: } 100011\mathbf{010011}, \quad \text{Filho 2: } 110011\mathbf{110010}.$$

(e) **Mutação**: Com taxa de mutação de 65%, cada bit dos filhos tem 65% de chance de ser alterado. Suponhamos que o 3º bit do Filho 1 e o 8º bit do Filho 2 sofram mutação:

$$\text{Filho 1 mutado: } 101011010011, \quad \text{Filho 2 mutado: } 110011100010.$$

(f) **População Pronta para a Próxima Geração**: A nova população será composta pelo indivíduo elitizado e os dois filhos resultantes:

Indivíduo	**Descrição**
110011010011	Elite preservado
101011010011	Filho 1 mutado
110011100010	Filho 2 mutado

Table 3: População após a primeira geração.

3. **Segunda Geração**:

(a) **Avaliação da População**: A população da geração anterior, composta pelo indivíduo elitizado e os dois filhos mutados, é reavaliada. A Tabela 4 apresenta os indivíduos e suas respectivas avaliações.

Indivíduo	**Valor de x**	**Avaliação**
110011010011	2,40513	3,32914
101011010011	1,54632	2,19856
110011100010	1,78342	2,53421

Table 4: População avaliada na segunda geração.

(b) **Elitismo**: O melhor indivíduo da população, 110011010011, com $f(x) = 3,32914$, é preservado para a próxima geração.

(c) **Seleção (Roleta)**: As probabilidades de seleção são calculadas com base na avaliação de cada indivíduo. A soma total das avaliações é:

$$\text{Soma Total das Avaliações} = 3,32914 + 2,19856 + 2,53421 = 8,06191.$$

As probabilidades de seleção são:

- Indivíduo 1 (Elite): $\frac{3,32914}{8,06191} \approx 0,413$;
- Indivíduo 2 (Filho 1): $\frac{2,19856}{8,06191} \approx 0,273$;
- Indivíduo 3 (Filho 2): $\frac{2,53421}{8,06191} \approx 0,314$.

Supondo que, pela roleta, os indivíduos **Filho 2 (110011100010)** e **Filho 1 (101011010011)** sejam selecionados.

(d) **Crossover (1 ponto)**: O crossover será realizado entre os dois indivíduos selecionados. O ponto de crossover é sorteado, por exemplo, no 6º bit:

Indivíduo 1 (Filho 2): $110011\mathbf{100010}$, Indivíduo 2 (Filho 1): $101011\mathbf{010011}$.

Após o crossover:

Filho 1: $110011\mathbf{010011}$, Filho 2: $101011\mathbf{100010}$.

(e) **Mutação**: Com uma taxa de mutação de 65%, cada bit tem essa probabilidade de ser alterado. Supondo que:

- No Filho 1, o 4º bit seja mutado: $110011010011 \rightarrow 110111010011$.
- No Filho 2, o 2º bit seja mutado: $101011100010 \rightarrow 111011100010$.

Após a mutação, os novos cromossomos são:

- Filho 1: 110111010011;
- Filho 2: 111011100010.

(f) **População Final da Segunda Geração**: A população final desta geração é composta pelo indivíduo elitizado e os dois filhos resultantes. A Tabela 5 apresenta os resultados finais desta geração.

Indivíduo	**Valor de x**	**Avaliação**
110011010011	2,40513	3,32914
110111010011	1,73245	2,48763
111011100010	1,95231	2,65217

Table 5: População final da segunda geração.

4. **Resultado Final**: Após duas gerações, o melhor indivíduo encontrado é o que possui a maior avaliação final, neste caso: $x = 2,40513$ com $f(x) = 3,32914$.

2 Aplicações Variadas

Os Algoritmos Genéticos têm revolucionado diversas áreas do conhecimento. Na área de otimização computacional, pesquisadores como Hwang e He [49] desenvolveram, em 2006, uma abordagem inovadora que combina AG com técnicas de meta-heurística e *Simulated Annealing*, criando soluções mais robustas para problemas complexos.

No campo industrial, Pendharkar (2007) [50] trouxe uma contribuição notável ao desenvolver um sistema inteligente baseado em AG para otimizar a programação da produção. Seu trabalho conseguiu equilibrar múltiplas variáveis do ambiente fabril, como tempo, recursos e custos, através de simulações computacionais sofisticadas.

Uma aplicação particularmente interessante foi desenvolvida por James e colaboradores (2006) [51], que criaram um sistema capaz de lidar com múltiplas funções objetivas simultaneamente, aproveitando o poder de processamento paralelo - um verdadeiro avanço para a época.

Recentemente, Wen et al. (2022) [52] apresentaram um uso inovador dos AGs em *Machine Learning*, aplicando-os na busca evolutiva de arquiteturas de redes neurais para classificação de imagens. Esse avanço demonstra a relevância contínua dos AGs em áreas de fronteira, como inteligência artificial e *deep learning*.

Outras aplicações impressionantes incluem:

- **Mitigação de Epidemias em Redes Sociais**: Em 2017, Concatto et al. [53] propuseram um algoritmo genético para minimizar a propagação de infecções em redes, focando na remoção estratégica de relacionamentos. O estudo apresentou resultados promissores na redução do número de indivíduos infectados durante a simulação de epidemias, destacando a eficácia dos AGs em problemas de saúde pública.
- **Otimização de Redes Neurais Artificiais**: Silva e Ludermir (2021) [54] apresentaram uma metodologia que utiliza Algoritmos Genéticos Celulares para a busca automática de arquiteturas de Redes Neurais Artificiais. O objetivo foi encontrar redes compactas com bom desempenho em problemas de classificação, evidenciando a capacidade dos AGs em configurar modelos eficientes sem intervenção manual intensiva.
- **Aplicações em Engenharia Logística**: Moreira (2021) [55] desenvolveu um Algoritmo Genético aplicado à resolução de problemas na Engenharia Logística, demonstrando a eficácia dessa abordagem na otimização de processos complexos e na melhoria da eficiência operacional.

Cada uma dessas aplicações demonstra como os Algoritmos Genéticos podem ser adaptados para resolver problemas complexos do mundo real, imitando o processo natural de evolução para encontrar soluções cada vez melhores.

5 Metodologia

A proposta do GADEMO busca oferecer uma experiência mais escalável, flexível e performática, proporcionando aos usuários um ambiente propício para explorar e experimentar com AGs de maneira dinâmica (De Jong, 2016 [56]). Com uma arquitetura baseada na separação entre *backend* e *frontend*, conectados por meio de uma API RESTful (FELDING, 2000 [8]), e seguindo padrões modernos de arquitetura de software (Richards, 2015 [57]), o sistema assegura que cada parte desempenhe funções específicas e complementares: o *backend* lida com a lógica computacional e execução dos Algoritmos Genéticos, enquanto o *frontend* promove uma interação intuitiva e amigável com o usuário (Garret, 2010 [58]).

A aplicação visa fornecer uma ferramenta educacional robusta e de fácil uso (Whitley, 1994 [59]). Para isso, a metodologia de desenvolvimento foi orientada pela seleção de tecnologias modernas e *frameworks* que maximizam a eficiência, a escalabilidade e a usabilidade (Newman, 2021 [60]). A parametrização detalhada dos algoritmos genéticos, aliada à visualização gráfica em tempo real dos resultados dos experimentos, busca melhorar a experiência do usuário e tornar o sistema mais adaptado às necessidades contemporâneas de estudo, análise e aplicação de AGs em ambientes computacionais diversos (Back et. al, 2007 [61]).

O GADEMO foi concebido para oferecer uma arquitetura escalável, modular e aberta, atendendo às demandas contemporâneas de experimentação com Algoritmos Genéticos (Coello, 2007 [62]). Ou seja, uma nova proposta deve oferecer uma abordagem que permita ao usuário avaliar o desempenho com métricas mais universalmente reconhecidas em otimização, possibilitando a análise de diferentes funções, utilizando gráficos que mostrem evolução de aptidões, distribuição de valores de *fitness*, variação por geração, e flexibilidade na configuração de experimentos, tudo isso através de uma interface web mais moderna e interativa.

a Arquitetura do Sistema

Para garantir a flexibilidade e a consistência desejadas, o GADEMO foi projetado com uma arquitetura cuja abordagem é modular. O *frontend* foi pensado para proporcionar uma interação intuitiva com o usuário, permitindo a configuração de parâmetros, visualização de resultados e execução de experimentos. Já o *backend* lida com a execução dos Algoritmos Genéticos, oferecendo um processamento eficiente e escalável por meio de padrões arquiteturais como o *Domain-Driven Design* (DDD) (Freeman, 2007 [63]).

Para abordar a arquitetura do sistema, é fundamental considerar os princípios de escalabilidade, boas práticas de programação, *design patterns* e a estruturação orientada para microsserviços. O objetivo é fornecer uma solução robusta, flexível e modular, que suporte a expansão de funcionalidades e ofereça um desempenho eficiente.

1 Escalabilidade e Boas Práticas de Programação

Escalabilidade é essencial para garantir que o sistema possa atender a um número crescente de usuários ou requisitos sem perda significativa de desempenho. Ao separar as responsabilidades do *frontend* e *backend*, permite-se que cada parte seja escalada independentemente conforme necessário. As boas Práticas de Programação garantem a manutenção e a legibilidade do código, facilitando a colaboração em equipe e a adição de novas funcionalidades. Isso inclui:

- *Clean Code* (Código Limpo);
- Adoção de boas práticas de versionamento com repositórios no GitHub para cada serviço.
- Utilização de ferramentas para *linting* e formatação automática de código.

2 *Design Patterns* e Estruturação de Projetos

O uso de *Design Patterns* (Freeman, 20007 [63]) nos garante uma estrutura organizada e modular do código, facilitando a reutilização de componentes e a manutenção. No *backend*, padrões como ”Service Layer” e ”Repository Pattern” são usados para organizar os serviços de lógica de negócio e a interação com dados. No *frontend*, a separação de componentes

é fundamental para facilitar a reutilização e atualização dos elementos de interface, como o teclado matemático e os gráficos.

As aplicações estão hospedadas em serviços como o Heroku, permitindo a escalabilidade automática, e estão versionadas em repositórios GitHub para maior controle e colaboração.

3 Diagrama da Arquitetura do Sistema

Abaixo há uma simples representação do diagrama arquitetural do sistema Gademo, exibindo as principais conexões entre o *Frontend* e *Backend*.

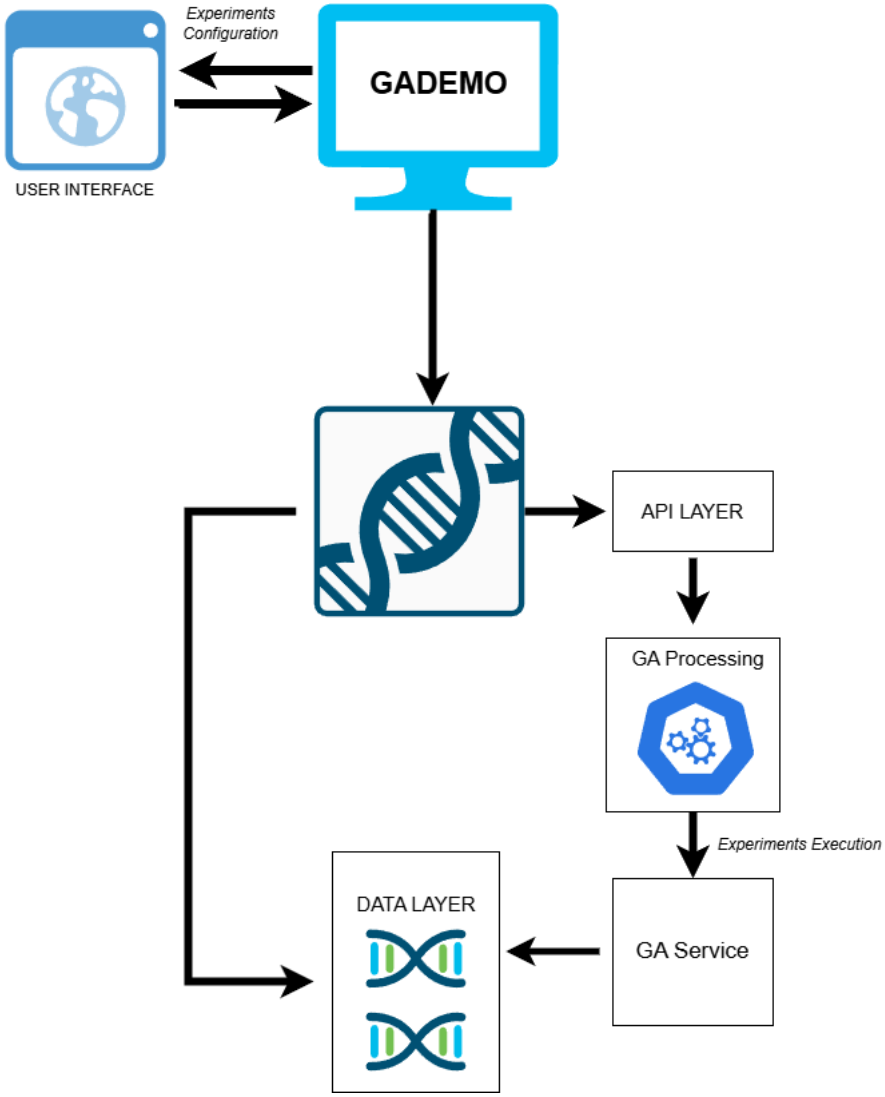


Figure 12: Arquitetura do Sistema

1. **User Interface (Interface do Usuário)**
   - **Descrição**: Esta camada representa o *frontend* da aplicação, que é a interface visível e interativa para os usuários. A interface do usuário desempenha um papel crítico ao permitir a interação direta com o sistema. Através dela, os usuários podem configurar experimentos, fornecer parâmetros de entrada e visualizar os resultados dos AGs de maneira intuitiva e amigável. A escolha de uma interface web moderna possibilita a acessibilidade em múltiplas plataformas e dispositivos, sem a necessidade de instalação local de softwares específicos, maximizando a experiência do usuário. Os dados e solicitações do usuário são enviados para o

backend através da API.

2. **GADEMO (Central de Gerenciamento)**:
   - **Descrição**: Gerencia a interação com as camadas de API e processamento de Algoritmos Genéticos. Recebe as solicitações e os parâmetros do usuário e repassa ao serviço de processamento.
3. ***API Layer* (Camada de API)**:
   - **Descrição**: A camada de API serve como ponto de conexão entre o *frontend* e o *backend*, garantindo que a comunicação entre as partes seja feita de forma desacoplada e padronizada. Utilizando princípios RESTful (FELDING, 2000 [8]), a API fornece *endpoints* bem definidos para enviar e receber dados, facilitando a integração e manutenção do sistema. Além disso, esta camada promove a escalabilidade horizontal, permitindo que múltiplas instâncias da API possam ser executadas em paralelo para lidar com um volume maior de requisições.
4. ***Genetic Algorithm Processing* (Processamento de Algoritmos Genéticos)**:
   - **Descrição**: Esta camada lida especificamente com a execução dos experimentos dos AG. É o núcleo do sistema, onde a lógica dos Algoritmos Genéticos é executada. Ela é responsável pela aplicação de operadores genéticos como seleção, cruzamento (*crossover*) e mutação, além de avaliar as funções de *fitness* para cada geração. A separação dessa lógica em uma camada distinta permite modularidade e facilita a evolução do sistema, possibilitando a introdução de novas estratégias de AG sem impactar outras partes da arquitetura.
5. **GA *Service* (Serviço de Algoritmos Genéticos)**:
   - **Descrição**: Este serviço encapsula as operações realizadas pelos AG, permitindo que o processamento de múltiplos experimentos seja organizado e gerenciado. O *GA Service* atua como um intermediário que orquestra a execução dos experimentos. Ele gerencia a configuração dos parâmetros, realiza ajustes necessários e garante que os resultados sejam formatados de maneira adequada para retorno à camada de API. Essa separação de responsabilidades melhora a organização do código, permitindo que a lógica de negócios seja isolada e de fácil manutenção.
6. ***Data Layer* (Camada de Dados)**:
   - **Descrição**: A camada de dados tem a função de armazenar informações relacionadas aos experimentos, resultados e métricas de desempenho dos AGs. A persistência de dados permite análises futuras e comparações, possibilitando uma avaliação detalhada da evolução das populações e do comportamento dos AGs ao longo dos experimentos. O armazenamento estruturado facilita a consulta e recuperação de informações, promovendo eficiência e integridade dos dados.
7. **Fluxo de Dados e Processamento**:
   - **Fluxo Geral**: O fluxo de dados na arquitetura é projetado para ser linear e modular, com cada camada interagindo com a seguinte de maneira bem definida. A interface do usuário envia requisições para a camada de API, que repassa os parâmetros para o *GA Service* e, em seguida, para a camada de processamento de AGs. Os resultados são então retornados, armazenados na camada de dados, e enviados de volta para a interface do usuário. Este fluxo garante que a lógica de apresentação, processamento e armazenamento seja isolada, permitindo escalabilidade e modularidade.

4 Microsserviços e Integração com Heroku

A arquitetura orientada a microsserviços permite a decomposição das funcionalidades em serviços menores, que podem ser implementados, escalados e mantidos de forma independente. Isso é especialmente relevante para o GADEMO, considerando a comunicação entre o *frontend* e *backend* por meio de chamadas RESTful (FELDING, 2000 [8]).

A seguir, é exibida a ilustração da infraestrutura da aplicação:

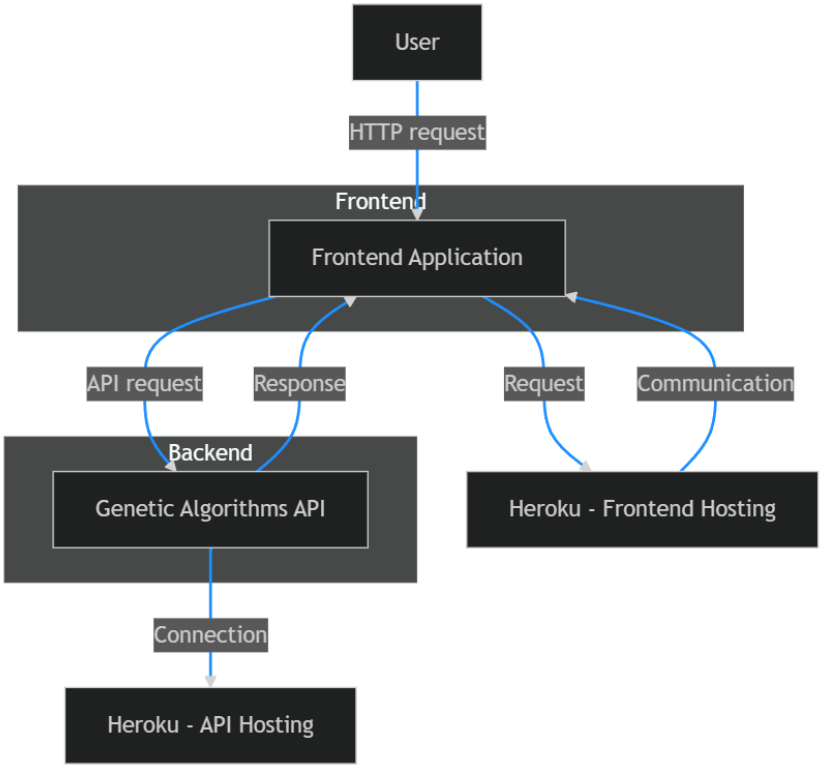


Figure 13: Diagrama de Infraestrutra

1. ***User* (Usuário)**:
   Representa o usuário final que interage com o sistema GADEMO por meio da interface. O usuário envia solicitações (HTTP requests) ao sistema por meio de ações como submeter uma função para execução dos Algoritmos Genéticos, configurar parâmetros ou visualizar resultados.
2. **HTTP *Request***:
   Esse componente mostra a comunicação inicial entre o usuário e o sistema. O usuário faz uma solicitação HTTP, que pode ser uma requisição de página ou de interação com o sistema por meio de *inputs* na interface, onde as requisições HTTP são tratadas.
3. ***Frontend Application:***
   Esta é a camada de *frontend*, onde está a interface gráfica do usuário. Neste caso, ela inclui a aplicação desenvolvida para interação com o GADEMO. O Frontend da aplicação recebe e processa as solicitações do usuário; exibe resultados, gráficos e informações com base na execução dos Algoritmos Genéticos e Comunica-se com o *backend* através de chamadas de API para enviar e receber dados.
4. **Heroku - *Frontend Hosting:***
   Representa o serviço de hospedagem onde a aplicação *frontend* é implantada. No contexto atual, foi utilizado o Heroku como plataforma para disponibilizar a interface frontend do GADEMO. O Heroku oferece escalabilidade e facilidade de uso, permitindo que a aplicação esteja acessível pela web sem a necessidade de configuração complexa de servidores.

5. **API *Request / Response:***
   Mostra o fluxo de comunicação entre o *frontend* e o *backend* por meio de solicitações e respostas de API. Quando o usuário faz uma ação que requer processamento (por exemplo, executar um Algoritmo Genético), o frontend envia uma requisição de API para o *backend*. A resposta do *backend* retorna para o *frontend*, que apresenta os resultados ao usuário.
6. ***Backend (Genetic Algorithms API):***
   Camada de backend que contém a lógica da aplicação. A API de Algoritmos Genéticos é responsável pelo processamento das solicitações enviadas pelo *frontend*. Isso inclui a execução de funções e cálculos relacionados aos Algoritmos Genéticos e o retorno de resultados processados para o *frontend*. A lógica de negócios e algoritmos estão encapsulados nesta camada, garantindo modularidade e separação de responsabilidades.
7. ***Connection to Heroku - API Hosting:***
   A API também está hospedado no Heroku, representando a infraestrutura que permite acesso ao *backend* pela web. A hospedagem no Heroku facilita a escalabilidade e disponibilidade da API, permitindo que o *frontend* e outros serviços interajam com ela de forma confiável.

b *Backend*

1 Padrão DDD (*Domain-Driven Design*)

O *backend* do projeto GADEMO foi projetado seguindo o padrão arquitetural de *Domain-Driven Design* (Freeman, 20007 [63]), que divide a aplicação em camadas distintas com responsabilidades específicas. A estrutura apresentada no backend reflete essa divisão em quatro camadas principais: ***Domain***, ***Infrastructure***, ***Interface*** e ***Services***. O DDD proporciona maior modularidade, escalabilidade e facilita a manutenção do código.

- ***Domain***
  - **Arquivos principais:**
    - `crossover_type.py`
      Este módulo define os diferentes tipos de *crossover* disponíveis para o Algoritmo Genético (AG). Ele contém uma classe que permite a configuração de opções como *one-point*, *two-point* e *uniform*.
    - `execution_characteristics.py`
      Este módulo define as características de execução para os experimentos do Algoritmo Genético, como número de gerações, tamanho da população, taxas de mutação e *crossover*, e opções para normalização, elitismo, e outros parâmetros ajustáveis.
- ***Infrastructure***
  - **Arquivos principais:**
    - `genetic_algorithm_executor.py`
      Este módulo contém a lógica central para a execução dos Algoritmos Genéticos. Ele implementa as funções necessárias para inicialização de populações, avaliação de indivíduos, aplicação de operadores genéticos (crossover, mutação) e seleção de indivíduos. As principais funções incluem:
      - `__init__`: Inicializa o executor do Algoritmo Genético.
      - `get_function`: Converte a função fornecida em uma expressão simbólica, aceitando funções de uma ou duas variáveis.
      - `run_genetic_algorithm`: Executa o Algoritmo Genético para uma população definida, aplicando operadores genéticos e calculando *fitness*.
      - `run_multiple_experiments`: Executa múltiplos experimentos de maneira assíncrona utilizando `concurrent.futures` para paralelismo.
      - `evaluate_func`: Avalia a função de *fitness* para um indivíduo.

        - `mutate_within_bounds`: Aplica mutação a um indivíduo, garantindo que os valores estejam dentro dos limites especificados.

- ***Interface***
  - **Arquivos principais:**
    - `api.py`
      Define os *endpoints* da API, permitindo a comunicação entre o *frontend* e o *backend*. Utiliza a biblioteca FastAPI para expor *endpoints* que recebem os parâmetros de configuração do Algoritmo Genético, executam os experimentos e retornam os resultados. Os principais *endpoints* incluem:
      - `/run-experiments`: Recebe parâmetros de configuração via POST, executa os experimentos e retorna os resultados em formato JSON.

- ***Services***
  - **Arquivos principais:**
    - `genetic_algorithm_service.py`
      Esta camada lida com a lógica de serviço para a execução de Algoritmos Genéticos. Ela atua como um intermediário entre a interface da API e a execução dos experimentos, garantindo que os parâmetros sejam adequadamente transmitidos e os resultados sejam processados e formatados.

O uso do padrão DDD (Freeman, 20007 [63]) organiza a aplicação em camadas bem definidas, cada uma com sua responsabilidade. A camada de ***Domain*** define as regras de negócio e entidades centrais. ***Infrastructure*** fornece suporte técnico e acesso a recursos externos (por exemplo, a execução de algoritmos). A **Interface** lida com a comunicação externa, e **Services** intermedia a lógica de aplicação, promovendo uma separação de responsabilidades e facilitando a escalabilidade e manutenção do sistema.

2 *UML Diagram*

Para proporcionar uma visão mais detalhada da arquitetura do sistema, foi desenvolvido o diagrama UML do *backend*. Assim, é possível ponderarmos seus componentes principais, e as relações e interações entre eles. O diagrama de classes UML, em particular, descreve as classes de um sistema, seus atributos e métodos, além das relações entre elas, como associações, dependências e herança.

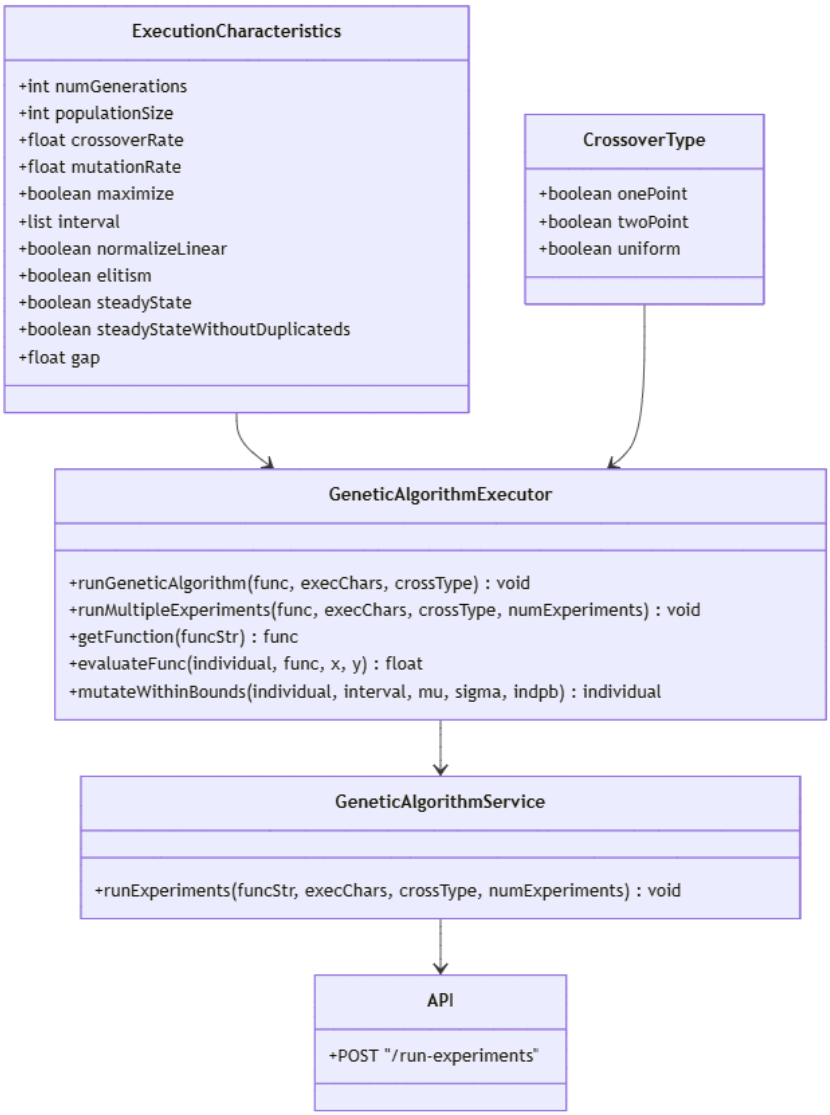


Figure 14: Diagrama UML do Backend

- **ExecutionCharacteristics**: Esta classe representa as características de execução dos algoritmos genéticos. Os atributos incluem:
  - `numGenerations`: Número de gerações a serem processadas.
  - `populationSize`: Tamanho da população a ser utilizada.
  - `crossoverRate`: Taxa de crossover, representada como um valor float.
  - `mutationRate`: Taxa de mutação, também representada como um valor float.
  - `maximize`: Booleano que indica se a função deve ser maximizada.
  - `interval`: Lista de valores que define os limites do intervalo de busca.
  - `normalizeLinear`: Booleano que determina se a normalização linear será utilizada.

  - `elitism`: Booleano que indica se a técnica de elitismo será aplicada.
  - `steadyState`: Booleano que define se o estado estacionário será usado.
  - `steadyStateWithoutDuplicateds`: *Booleano* que determina se o estado estacionário sem duplicados será utilizado.
  - `gap`: *Gap* percentual para o uso em técnicas específicas.
- **CrossoverType**: Classe que define os tipos de cruzamento disponíveis para os algoritmos genéticos. Os atributos são:
  - `onePoint`: *Booleano* que indica se o cruzamento de um ponto será utilizado.
  - `twoPoint`: *Booleano* que indica se o cruzamento de dois pontos será utilizado.
  - `uniform`: *Booleano* que indica se o cruzamento uniforme será utilizado.
- **GeneticAlgorithmExecutor**: Esta classe é responsável por executar os algoritmos genéticos. Os métodos incluem:
  - `runGeneticAlgorithm(func, execChars, crossType) : void`: Executa o algoritmo genético com a função, características de execução e tipo de cruzamento especificados.
  - `runMultipleExperiments(func, execChars, crossType, numExperiments) : void`: Executa múltiplos experimentos com os parâmetros fornecidos.
  - `getFunction(funcStr) : func`: Retorna a função a ser otimizada, baseada em uma string.
  - `evaluateFunc(individual, func, x, y) : float`: Avalia a função fornecida para um indivíduo específico.
  - `mutateWithinBounds(individual, interval, mu, sigma, indpb) : individual`: Aplica a mutação a um indivíduo dentro dos limites especificados.
- **GeneticAlgorithmService**: Esta classe atua como uma camada de serviço para facilitar a execução dos experimentos com algoritmos genéticos. O método principal é:
  - `runExperiments(funcStr, execChars, crossType, numExperiments) : void`: Executa os experimentos usando os parâmetros fornecidos.
- **API**: Representa o ponto de entrada para a comunicação com o sistema através de uma API. Possui o seguinte endpoint:
  - `POST "/run-experiments"`: Endpoint que permite a execução dos experimentos através de uma chamada POST.

3 Bibliotecas Utilizadas no *Backend*

Para o desenvolvimento do *backend* do sistema GADEMO, foram utilizadas diversas bibliotecas que oferecem funcionalidades específicas e essenciais para a execução dos algoritmos genéticos, manipulação de dados e comunicação eficiente com o *frontend*. Abaixo, descrevemos cada uma dessas bibliotecas de maneira detalhada:

- **NumPy**
  A biblioteca NumPy (*Numerical* Python) foi desenvolvida inicialmente por Travis Oliphant em 2006 [64]. Ela é amplamente utilizada em ciência de dados e computação científica devido ao suporte a arrays multidimensionais (ndarray) e a uma ampla gama de funções matemáticas de alto desempenho. NumPy é projetada para operações vetorizadas, proporcionando eficiência em cálculos numéricos de larga escala e manipulação de matrizes. No contexto dos algoritmos genéticos, é utilizada para operações que envolvem manipulação de vetores e matrizes representando populações, além de cálculos matemáticos em avaliações de funções de *fitness*, mutações e cruzamentos.
- **Pandas**
  A biblioteca Pandas foi criada por Wes McKinney em 2008 [65]. Ela é uma ferramenta poderosa para manipulação e análise de dados, oferecendo estruturas de dados rápidas, flexíveis e expressivas, como *DataFrames* e *Series*. Essas estruturas permitem operações como filtragem, agregação e transformação de dados tabulares, além de suporte para operações de entrada e saída (I/O). No GADEMO, Pandas pode ser usada para registrar, analisar e manter um histórico das execuções dos algoritmos genéticos, incluindo as medições de *fitness* e evolução das populações.
- **DEAP (Distributed Evolutionary Algorithms in Python)**
  DEAP foi desenvolvida por Fortin et al. em 2012 [66] e é focada em algoritmos evolucionários e genéticos, oferecendo uma estrutura flexível para definição de problemas de otimização e soluções usando operadores genéticos, como seleção, cruzamento (*crossover*) e mutação. DEAP facilita a implementação de algoritmos genéticos personalizados com classes e funções para populações, funções de *fitness* e operadores genéticos. No GADEMO, ela fornece a base para a execução da lógica evolucionária, tornando possível a manipulação e evolução dos indivíduos.
- **SymPy**
  SymPy, criada por Ondřej Čertík e outros colaboradores em 2006 [67], é uma biblioteca de álgebra simbólica que oferece recursos para manipulação simbólica de expressões matemáticas, incluindo diferenciação, integração, resolução de equações algébricas e operações de simplificação. No contexto do GADEMO, SymPy pode ser usada para definir e manipular funções matemáticas de maneira simbólica, proporcionando flexibilidade para avaliações complexas e cálculos precisos.
- **Uvicorn**
  Uvicorn foi criado por Tom Christie e é um servidor ASGI (*Asynchronous Server Gateway Interface*) leve e rápido [68]. Ele é utilizado para executar aplicações web assíncronas em Python. Ele é compatível com *frameworks* como FastAPI, permitindo a criação de APIs de alto desempenho e suporte para operações assíncronas. No *backend* do GADEMO, Uvicorn é utilizado para hospedar a API construída com FastAPI, garantindo serviços eficientes para lidar com requisições de usuários.
- **FastAPI**
  FastAPI foi desenvolvido por Sebastián Ramírez em 2018 [69]. É um *framework* moderno e rápido para construção de APIs com Python, baseado em padrões como OpenAPI e JSON Schema. Ele oferece suporte para operações assíncronas, validação automática de dados, documentação interativa e facilidade de uso. É conhecido por ser um dos *frameworks* mais rápidos para construção de APIs, mantendo flexibilidade e desempenho. No GADEMO, FastAPI é utilizado para criar *endpoints* que permitem a comunicação entre o *frontend* e os algoritmos genéticos, oferecendo uma experiência escalável e de alta performance.
- **Flask**
  Flask, criado por Armin Ronacher em 2010 [70], é um *framework* micro de desenvolvimento web utilizado para criar aplicações leves e APIs RESTful (FELDING, 2000 [8]). Ele é flexível, permitindo a definição de rotas e gerenciamento de requisições de maneira

simples. Embora FastAPI seja a escolha predominante devido ao suporte para operações assíncronas e documentação automática, Flask pode ser uma alternativa para construção rápida de APIs simples.

4 Análise da API de Processamento de Algoritmos Genéticos

A API desenvolvida para o sistema GADEMO tem como principal finalidade expor uma interface para a execução e monitoramento de Algoritmos Genéticos de maneira flexível, permitindo múltiplas configurações de parâmetros e a execução de experimentos em diferentes cenários. Utilizando tecnologias modernas e arquiteturas RESTful (FELDING, 2000 [8]), a API foi projetada para ser eficiente, escalável e modular, proporcionando uma integração adequada com o *frontend* e o processamento dos algoritmos de otimização. A seguir, detalhamos seu funcionamento:

- ***Parameters*:**
  A API permite que parâmetros específicos sejam passados para configurar a execução dos Algoritmos Genéticos. No exemplo adiante (`response(200)`), temos:
  - `func_str`: Representa a função a ser otimizada, inserida como uma string. Suponhamos a função F6, cuja definição é:

$$f(x,y) = 0.5 - \frac{(sin^2(\sqrt{x^2+y^2}) - 0.5)}{(1+0.001*(x^2+y^2))^2}$$

    Esta função é comumente utilizada em testes de Algoritmos Genéticos para avaliar sua performance em encontrar valores ótimos.
  - `num_experiments`: Número de experimentos a serem realizados com as configurações fornecidas. No exemplo, foram executados 2 experimentos.
- ***Endpoint*:**
  O ponto de acesso principal da API é definido como um *endpoint* POST, disponível em: `/run-experiments`. Este *endpoint* recebe uma solicitação contendo os parâmetros necessários para executar os experimentos com Algoritmos Genéticos, como funções, características de execução e tipos de cruzamento.
- ***Request Body*:**
  O corpo da solicitação enviado ao *endpoint* deve conter os seguintes campos (representados em formato JSON), que determinam os parâmetros de execução dos algoritmos genéticos:

```
{
  "num_generations": 3,
  "population_size": 5,
  "crossover_rate": 65,
  "mutation_rate": 0.8,
  "maximize": true,
  "interval": [-100, 100],
  "crossover_type": {
    "one_point": true,
    "two_point": false,
    "uniform": false
  },
  "normalize_linear": false,
  "normalize_min": 0,
  "normalize_max": 100,
  "elitism": false,
  "steady_state": false,
  "steady_state_without_duplicateds": false,
  "gap": 0
  "execution_time": 15
}
```

- ***Response* (200):**
  A resposta retornada pela API em uma requisição bem-sucedida fornece informações detalhadas sobre os resultados do experimento realizado, incluindo os melhores valores encontrados em cada geração, os melhores indivíduos, e outros dados de análise. Exemplo de resposta:


```
 {
"best_experiment_values": [
  0.6870479695091991,
  0.5197181697952571
],
"best_individuals_per_generation": [
  [
    [-1.5487334097288254, -21.614351295001683],
    [-1.5487334097288254,25.02198759075202],
    [-1.5487334097288254,25.02198759075202]
  ],
  [
    [35.04715535334745,-11.94081518734265],
    [35.04715535334745,-11.94081518734265],
    [35.04715535334745,-11.94081518734265]
  ]
],
"mean_best_individuals_per_generation":
[
  0.6025172672675478,
  0.6033830696522281,
  0.6033830696522281
],
"best_values_per_generation":
[
  [
    0.6853163647398386,
    0.6870479695091991,
    0.6870479695091991
  ],
  [
    0.5197181697952571,
    0.5197181697952571,
    0.5197181697952571
  ]
],
"last_generation_values": [
  [
    0.6537096812273153,
    0.6853163647398386,
    0.32152644106071643,
    0.6870479695091991,
    0.6714355324571999
  ],
  [
    0.5197181697952571,
    0.5197181697952571,
    0.4943729653321432,
    0.5197181697952571,
    0.5197181697952571
  ]
]
}
```

Explicação Detalhada da Estrutura de Resposta:
Para melhor compreensão, detalhamos os elementos da resposta com base nos parâmetros fornecidos:

- `"best_experiment_values"`: Contém os melhores valores de *fitness* encontrados ao longo de cada experimento realizado. No exemplo acima, são fornecidos dois valores, um para cada experimento.
- `"best_individuals_per_generation"`: Lista de listas representando os melhores indivíduos (em termos de coordenadas) encontrados em cada geração para cada experimento. Cada sublista contém listas correspondentes a cada geração.
- `"mean_best_individuals_per_generation"`: Apresenta a média dos melhores valores de *fitness* para cada geração, considerando todos os experimentos realizados.
- `"best_values_per_generation"`: Lista com os melhores valores de *fitness* encontrados em cada geração de cada experimento. Cada sublista representa os valores para uma geração específica em um experimento.
- `"last_generation_values"`: Apresenta os valores de *fitness* de todos os indivíduos na última geração de cada experimento, fornecendo um panorama do estado final da população.

Reflexão sobre o Design e Implementação:
A API foi construída com foco em modularidade e eficiência. Ao receber os parâmetros de configuração, ela valida e processa os dados por meio de funções que executam os Algoritmos Genéticos em diferentes cenários experimentais. A resposta inclui métricas relevantes, como o valor médio dos melhores indivíduos, garantindo uma análise robusta dos resultados de otimização. Ressalta-se o uso de endpoints RESTful (FELDING, 2000 [8]) para facilitar a comunicação e integração com outras camadas do sistema, permitindo escalabilidade e flexibilidade no processamento das requisições.

c *Frontend*

1 Arquitetura do *Frontend*

O *frontend* do GADEMO é projetado para oferecer uma interface de usuário amigável e intuitiva, permitindo interação direta com a API de processamento de Algoritmos Genéticos. A arquitetura segue princípios de modularidade e escalabilidade, com arquivos bem organizados para facilitar a manutenção e a extensão da aplicação.

- **Organização do projeto:**
  A estrutura do projeto é organizada em diferentes diretórios, cada um responsável por uma funcionalidade específica. O `src/` contém os principais componentes de interface, scripts e arquivos de configuração necessários para a execução.
- **Principais componentes e arquivos:**
  Os principais arquivos incluem `index.html` para a estrutura da página, `index.js` para a lógica de interação e manipulação de eventos, `style.css` para o design visual, e outros scripts específicos como `keyboard.js`, `modal.js` e `spinner.js`, que implementam funcionalidades complementares.

2 Configuração e Dependências

- `package.json` e gerenciamento de dependências:
  O arquivo `package.json` gerencia as dependências da aplicação, especificando os pacotes necessários, como bibliotecas de frontend e scripts utilitários. Ele também define scripts de inicialização e execução.
- Arquivos de configuração, como `docker-compose.yml` e `Procfile`:
  A configuração do `docker-compose.yml` permite a execução da aplicação em ambientes conteinerizados, enquanto o `Procfile` é utilizado para definir os processos executados na plataforma de hospedagem.

3 Estrutura dos Arquivos HTML e CSS

- **Estrutura básica em `index.html`:**
  O `index.html` fornece a estrutura fundamental da interface web, contendo elementos de layout, links para estilos e scripts, e pontos de interação com o usuário. Ele serve como ponto de entrada para a aplicação.
- **Estilo com `style.css` e arquivos específicos:**
  O `style.css` define as regras de estilo para o layout geral da página, enquanto arquivos específicos como `keyboard.css`, `modal.css` e `spinner.css` são responsáveis por estilizar componentes individuais. Isso garante uma separação clara de responsabilidades no design.

4 Componentes de Interface e Funcionalidades

- `keyboard.js` para entrada de dados do usuário:
  Este arquivo implementa um teclado virtual que facilita a entrada de funções e parâmetros específicos para os Algoritmos Genéticos. Ele captura eventos de clique e teclado, enviando os valores para processamento.
- `modal.js` para manipulação de diálogos modais:
  Os modais são utilizados para fornecer informações ao usuário ou capturar entradas específicas. O `modal.js` gerencia a abertura, fechamento e interação dentro desses diálogos.
- `spinner.js` para carregamento e feedback de execução:
  Este componente fornece feedback visual ao usuário durante operações assíncronas, como requisições à API. Ele exibe um ícone de carregamento até que a operação seja concluída.

5 Integração e Comunicação com o Backend

- **Conexão com a API:**
  A comunicação com o backend ocorre através de requisições HTTP feitas pelo `index.js`, que envia dados de entrada para a API e processa as respostas recebidas.
- **Funções de chamada e resposta no `index.js`:**
  As funções neste arquivo capturam eventos de interface, constroem requisições adequadas para a API e manipulam os resultados, atualizando a interface de forma dinâmica com os valores retornados.

6 Dockerização e Execução

- **Arquivo `Dockerfile` e configuração de execução:**
  O `Dockerfile` define como a aplicação frontend é construída e executada em um contêiner, especificando as dependências e os comandos necessários para inicialização.
- `start.sh` para inicialização:
  Este script auxilia na inicialização do ambiente de execução, configurando variáveis e executando os processos necessários para a aplicação.

7 Diagrama UML (*Unified Modeling Language*)

A UML (*Unified Modeling Language*) foi inicialmente proposta pelos engenheiros de *software* Grady Booch, James Rumbaugh e Ivar Jacobson enquanto trabalhavam na *Rational Software Corporation*. Eles são frequentemente referidos como os ”Três Amigos” no contexto do desenvolvimento de UML. A versão inicial foi padronizada pela *Object Management Group* (OMG) em 1997 [71].

O diagrama UML apresentado ilustra a estrutura e a interação entre os principais componentes do *frontend* do sistema GADEMO. Os elementos são organizados para demonstrar a relação entre HTML, CSS, *scripts JavaScript* e configurações para contêineres. Cada elemento diagrama é detalhado adiante.

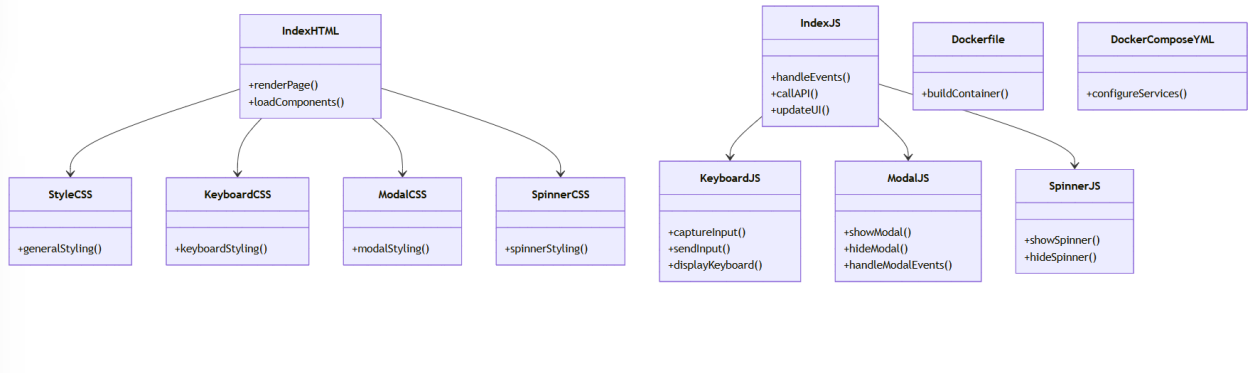


Figure 15: Diagrama UML do *Frontend*

- `IndexHTML`: Representa o ponto central de interação com o usuário. Ele carrega a página inicial do sistema e organiza a estrutura básica dos elementos visuais. Principais métodos:
  - `renderPage()`: Responsável por renderizar os elementos da interface.
  - `loadComponents()`: Carrega os componentes e scripts necessários para o funcionamento da página.
- `IndexJS`: Centraliza a lógica de interação e comunicação com o backend da aplicação. Principais métodos:
  - `handleEvents()`: Lida com eventos gerados pelo usuário.
  - `callAPI()`: Realiza chamadas para a API do backend.
  - `updateUI()`: Atualiza a interface do usuário com base nas respostas recebidas.
- `KeyboardJS`: Gerencia a entrada de dados do usuário por meio de um teclado virtual. Principais métodos:
  - `captureInput()`: Captura a entrada fornecida pelo usuário.
  - `sendInput()`: Envia os dados capturados para outros componentes.
  - `displayKeyboard()`: Exibe o teclado na interface.
- `ModalJS`: Manipula diálogos modais para exibição de informações, alertas ou interações específicas. Principais métodos:
  - `showModal()`: Exibe o modal na tela.
  - `hideModal()`: Oculta o modal.
  - `handleModalEvents()`: Gerencia eventos de interação dentro do modal.
- `SpinnerJS`: Lida com o feedback visual de carregamento e execução. Principais métodos:
  - `showSpinner()`: Mostra o indicador de carregamento.
  - `hideSpinner()`: Oculta o indicador de carregamento.
- `StyleCSS`, `KeyboardCSS`, `ModalCSS` e `SpinnerCSS`: São responsáveis pela estilização da interface do usuário, fornecendo regras de design para cada componente.
- `Dockerfile`: Define as etapas para construir e empacotar o frontend como um contêiner.
  - `buildContainer()`: Função responsável pela configuração de execução do contêiner.
- `DockerComposeYML`: Gerencia e configura os serviços em contêineres do frontend.
  - `configureServices()`: Configura os serviços necessários para execução do frontend.

Fluxo Geral: O fluxo do *frontend* inicia-se com a renderização do `IndexHTML`, que carrega os *scripts* principais como `IndexJS` para lidar com eventos e comunicações com o *backend*. Componentes como `KeyboardJS`, `ModalJS` e `SpinnerJS` fornecem funcionalidades específicas de interação. As regras de estilo aplicadas são definidas pelos arquivos CSS correspondentes. A configuração do ambiente em contêiner é feita com base no `Dockerfile` e `DockerComposeYML`, garantindo a escalabilidade e modularidade da aplicação.

6 Sistema GADEMO

O sistema GADEMO foi projetado com base no fluxo convencional de Algoritmos Genéticos (AG), fundamentado nos estudos aprofundados abordados na Seção de Fundamentação Teórica. Este projeto teve como objetivo aprimorar o sistema original do GADEMO *desktop*, incorporando melhorias substanciais para torná-lo mais robusto, escalável e alinhado às melhores práticas em algoritmos evolutivos.

Partindo de uma análise detalhada dos desafios e limitações do sistema *desktop*, a API foi desenvolvida para fornecer uma solução mais consistente e flexível, permitindo a execução eficiente de AGs com suporte a configurações personalizáveis, como diferentes métodos de seleção, tipos de *crossover* e técnicas de reprodução, incluindo o elitismo e *steady-state*.

A nova implementação destaca-se por sua coerência no fluxo interno, garantindo que cada etapa do AG seja conduzida de maneira estruturada, com espaço para adaptações e aperfeiçoamentos futuros. Além disso, o design modular da API promove escalabilidade e integração com sistemas externos, tornando-a uma plataforma versátil para aplicações científicas e acadêmicas.

a Visão Geral do Fluxo

O fluxo do AG no *backend* é composto por cinco etapas principais:

1. Inicialização: definição dos parâmetros do AG, geração inicial da população e normalização dos indivíduos.
2. Avaliação: cálculo das aptidões (*fitness*) de cada indivíduo com base na função objetivo fornecida pelo usuário.
3. Seleção: escolha dos indivíduos para reprodução, utilizando métodos configuráveis.
4. *Crossover*: combinação dos genes dos pais selecionados para gerar novos indivíduos.
5. Mutação: aplicação de variações aleatórias nos indivíduos gerados para aumentar a diversidade genética.

b Inicialização

Na etapa de inicialização, os seguintes parâmetros são definidos:

- Tamanho da população.
- Número de gerações.
- Taxa de *crossover* e mutação.
- Método de seleção e reprodução.
- Intervalo de busca dos valores (*lower bound* e *upper bound*).

A geração inicial é criada aleatoriamente, respeitando os limites estabelecidos, e os indivíduos são representados como sequências binárias de comprimento definido pela precisão especificada.

c Avaliação dos Indivíduos

O cálculo de aptidão (*fitness*) é realizado utilizando a própria função objetivo fornecida pelo usuário [9]. Cada indivíduo é decodificado (convertido do formato binário para valores reais) e avaliado. O backend implementa essa funcionalidade na função `evaluate()`, que recebe a população atual e a função objetivo, retornando as aptidões correspondentes. O valor de aptidão é armazenado para cada indivíduo, sendo utilizado nas etapas subsequentes.

[9] Isto pode ser modificado e refinado posteriormente. Estaremos abertos sempre a sugestões.

d Seleção

A seleção determina quais indivíduos irão participar do processo de reprodução. A biblioteca `DEAP` oferece suporte a diversos métodos de seleção configuráveis [10].

1 Torneio

No GADEMO, método de seleção padrão utilizado é o **Torneio** (`tools.selTournament`), cujo parâmetro relativo ao tamanho do torneio é (`tournsize`) e, por padrão, é configurado como sendo um número aleatório dentro do intervalo $[1, size(pop)]$, onde $size(pop)$ refere-se ao tamanho da população (número de indivíduos).

e Crossover

O *crossover* é a etapa responsável pela recombinação dos genes dos indivíduos selecionados. O GADEMO implementa o *crossover* de forma configurável. A taxa de *crossover* `crossover_rate` define a probabilidade de ocorrência dessa operação em cada indivíduo.

- ***one-point*** **(`tools.cxOnePoint`):** O ponto de corte é sorteado dentro do comprimento dos genes binários, e os filhos são gerados a partir dessa troca;
- ***two-point*** **(`tools.cxTwoPoint`):** Troca segmentos entre dois pontos de divisão.
- ***uniform*** **(`tools.cxUniform`):** Troca bits individuais com probabilidade `indpb=0.5`

O tipo de *crossover* utilizado é definido pelo parâmetro `cross_type`.

f Mutação

Após o *crossover*, é realizada a mutação, que consiste em alterações aleatórias nos genes dos indivíduos gerados. A mutação ocorre de forma independente para cada gene, com probabilidade definida, para cada indivíduo, pela taxa de mutação. Isso garante a introdução de variabilidade genética, prevenindo a convergência para soluções locais.

- A mutação ocorre com probabilidade configurada (`mutation_rate`);
- Cada gene (bit ou valor contínuo) é alterado com probabilidade `indpb=0.1`;
- É usada uma distribuição normal (`random.gauss`), e os valores são limitados ao intervalo permitido.

Método relacionado: `mutate_within_bounds`.

g Elitismo

No GADEMO, o *elitismo* foi implementado com uma regra adaptativa, baseada no tamanho da população. Essa abordagem permite preservar os melhores indivíduos de cada geração, garantindo que as melhores soluções encontradas até o momento sejam mantidas para a próxima geração. O número de indivíduos preservados é calculado dinamicamente, conforme descrito na tabela 6:

- **Regra de Elitismo por Faixas**: O número de indivíduos preservados (*elitismo*) é determinado pela função `calculate_elite_count`, que segue as seguintes faixas:
  - Para populações com tamanho entre 1 e 5 indivíduos: nenhum indivíduo é preservado (*elitismo* = 0);
  - Para populações com tamanho entre 6 e 10 indivíduos: preserva-se 1 indivíduo;
  - Para populações com tamanho superior a 10 indivíduos: preservam-se 2 indivíduos;

[10] Mais detalhes disponíveis na documentação: https://deap.readthedocs.io/en/master/

Tamanho da População	Indivíduos Preservados
1–5	0
6–10	1
11+	2

Table 6: Regra de Elitismo baseada no Tamanho da População

Então, o número de indivíduos elitizados é adaptado ao tamanho da população, garantindo equilíbrio entre diversidade e preservação das melhores soluções. A seleção e substituição ocorrem de forma direta e otimizada, sem a necessidade de alterar significativamente o restante da população. Além disso, a preservação de um número proporcional de indivíduos evita a convergência precoce em populações relativamente pequenas. Este sistema de elitismo dinâmico pode equilibrar a exploração do espaço de busca com a preservação das melhores soluções, sendo ideal para problemas que exigem tanto uma certa consistência quanto diversidade populacional.

h Steady-State

No GADEMO, o *Steady-State* é configurado como um método de reprodução que substitui apenas uma fração dos indivíduos da população a cada geração, enquanto o restante é mantido inalterado, criando uma *geração estável*. Esse método é amplamente utilizado por sua capacidade de equilibrar *exploração* e *exploração*, ao mesmo tempo em que evita mudanças drásticas na população, promovendo uma convergência mais controlada.

No sistema, o `gap` é o parâmetro que define a proporção da população que será substituída em cada geração. A configuração sem gap (ou seja, `gap=0`) resulta em uma proporção de substituição definida pelas operações genéticas de *crossover* e *mutação*. Já o *Steady-State sem duplicados* (`steady_state_without_duplicateds=True`) remove indivíduos redundantes da população, promovendo maior diversidade.

1 Funcionamento do Steady-State Sem Gap

Quando configurado sem `gap`, o número de indivíduos substituídos a cada geração é determinado pela lógica das operações genéticas (*crossover* e *mutação*). Em uma população de tamanho N, o processo pode ser descrito da seguinte forma:

- A **taxa de *crossover*** define a probabilidade de pares de indivíduos sofrerem recombinação. Para uma taxa típica de 65%, aproximadamente $0.65 \times \frac{N}{2}$ novos indivíduos são gerados por *crossover*.
- A **taxa de mutação** define a probabilidade de um indivíduo sofrer mutação. Com uma taxa de 0,8%, apenas $0.008 \times N$ indivíduos, em média, são alterados por mutação.
- Esses novos indivíduos substituem diretamente alguns da população atual, geralmente de forma aleatória ou por seleção baseada em aptidão (como torneio).

A proporção de substituição, neste caso, não é fixa, mas varia a cada geração dependendo da combinação das taxas de *crossover* e mutação e do resultado das operações genéticas. Em geral, apenas uma fração pequena da população (geralmente inferior a 30%) é substituída por geração.

2 Funcionamento do Steady-State com Gap

Quando o `gap` é definido, ele especifica explicitamente a fração da população que será substituída a cada geração. Por exemplo:

- Para `gap=0.10`, 10% da população será selecionada para reprodução, enquanto os outros 90% serão mantidos inalterados na próxima geração.

Este controle explícito da substituição permite um equilíbrio mais preciso entre *exploração* e *explotação*, sendo particularmente útil para evitar convergência prematura ou estagnação.

3 Steady-State sem Duplicados

A funcionalidade de **Steady-State sem duplicados** adiciona uma verificação para evitar que indivíduos redundantes sejam adicionados à população. Isso é particularmente relevante para evitar estagnação em problemas multimodais, onde a diversidade é essencial para escapar de máximos locais.

Quando `steady_state_without_duplicateds=True`:

- Cada novo indivíduo gerado por *crossover* ou *mutação* é comparado com a população existente.
- Indivíduos duplicados são descartados, garantindo que apenas soluções únicas sejam adicionadas.
- Caso todos os novos indivíduos sejam duplicados, a população permanece inalterada.

4 Resumo da Configuração

- **Sem gap:** A proporção de substituição é determinada pelas taxas de *crossover* e mutação. Geralmente, é baixa e varia a cada geração.
- **Com gap:** Uma fração fixa da população é substituída em cada geração, definida pelo parâmetro `gap`.
- **Sem duplicados:** Garante diversidade ao eliminar redundâncias, especialmente útil em problemas multimodais.

Essas configurações tornam o *Steady-State* uma abordagem flexível e adaptável às necessidades de diferentes problemas de otimização, permitindo um controle preciso sobre o balanço entre *exploração* e *explotação* (Goldberg, 1989 [2]).

i Normalização Linear

A normalização linear é usada como um método de reprodução, ajustando os valores de aptidão para um intervalo comum. Isso é particularmente útil para funções com escalas muito diferentes, garantindo que as aptidões estejam dentro de limites que favoreçam o processo de seleção.

- Ajusta os valores de aptidão para um intervalo específico (`[normalize_min, normalize_max]`).
- Implementado no método `normalize_fitness`.

j Fluxo Final do Algoritmo

O fluxo completo do AG no backend segue as etapas descritas acima de forma iterativa, até que o critério de parada (número de gerações ou limite de aptidão) seja atingido. O resultado final inclui o melhor indivíduo encontrado, sua avaliação e as estatísticas de aptidão ao longo das gerações.

7 GADEMO Web: Interface e Funcionalidades

Após a elaboração dos algoritmos e definições metodológicas no desenvolvimento do sistema *GADEMO*, esta seção apresenta a interface web construída para permitir a interação direta do usuário com os processos de otimização via Algoritmos Genéticos. O foco é descrever com precisão os elementos de interface, funcionalidades e como o sistema foi projetado para ser acessível, eficiente e funcional.

a Componentes de Entrada

A interface do *GADEMO Web* foi projetada para garantir uma experiência de usuário intuitiva, com foco em flexibilidade na definição de experimentos de otimização e visualização clara dos resultados obtidos. Os principais componentes da interface são detalhados a seguir:

1 Navbar e Configuração de Parâmetros

A barra de navegação (*sidebar*), como é exibida na imagem 16, é acessível por meio de um ícone de "*Burguer Button*". A *sidebar* desempenha um papel central na configuração de experimentos. Quando acionada, ela exibe um painel de configuração que permite a entrada de parâmetros essenciais para o funcionamento dos Algoritmos Genéticos, incluindo:

- **Número de Experimentos:** Define quantos experimentos independentes serão realizados. Esta configuração é crucial para permitir a análise estatística dos resultados, considerando diferentes execuções com a mesma configuração de parâmetros.
- **Número de Gerações:** Estabelece quantas gerações serão produzidas em cada experimento, impactando a convergência do algoritmo.
- **Tamanho da População:** Determina o número de indivíduos em cada geração, influenciando diretamente a diversidade genética e o espaço de busca.
- **Taxa de Crossover e Taxa de Mutação:** Controlam as operações de recombinação e mutação aplicadas aos indivíduos, parâmetros que são fundamentais para a exploração e exploração do espaço de soluções.
- **Objetivo de Maximização ou Minimização:** Permite ao usuário especificar se o objetivo do experimento é maximizar ou minimizar o *fitness*.
- **Intervalo de Valores:** Especifica os limites mínimos e máximos para os valores dos genes dos indivíduos, definindo o domínio da busca.
- **Tipo de Crossover e Opções Avançadas:** Oferece diferentes opções de crossover (como ponto único, dois pontos, e uniforme) e funcionalidades adicionais como normalização linear, elitismo e estado estacionário, permitindo maior controle sobre o comportamento evolutivo.

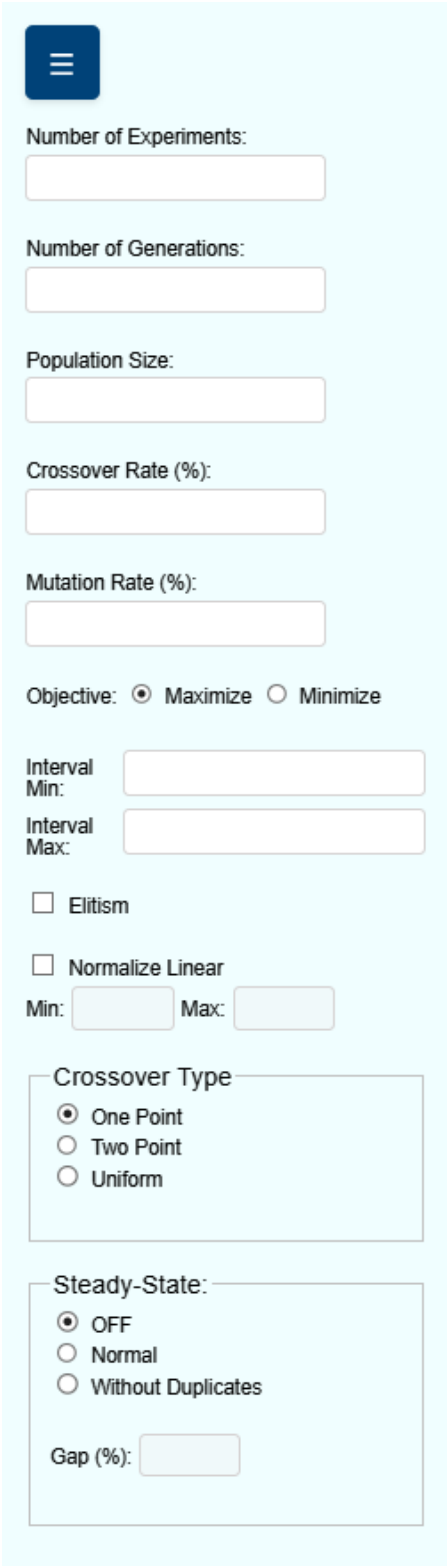


Figure 16: Sidebar da interface GADEMO

2 Teclado Numérico para Inserção de Funções

A interface conta com um teclado numérico (imagem 17) interativo que permite ao usuário definir a função de otimização desejada. Este componente, centralizado na interface, oferece

botões para operadores matemáticos básicos, funções trigonométricas, constantes matemáticas (π, e) e caracteres especiais para garantir flexibilidade na formulação de funções. A função é inserida diretamente em um campo de texto adjacente ao teclado e, ao ser concluída, o usuário deve acionar o botão *Run* para iniciar o experimento de otimização. Vale ressaltar que a utilização deste não é obrigatória, i.e., o usuário pode simplesmente optar por digitar as funções via teclado físico. Este recurso foi pensado apenas visando uma melhor experiência do usuário quando uma versão *mobile* for ajustada para utilização.

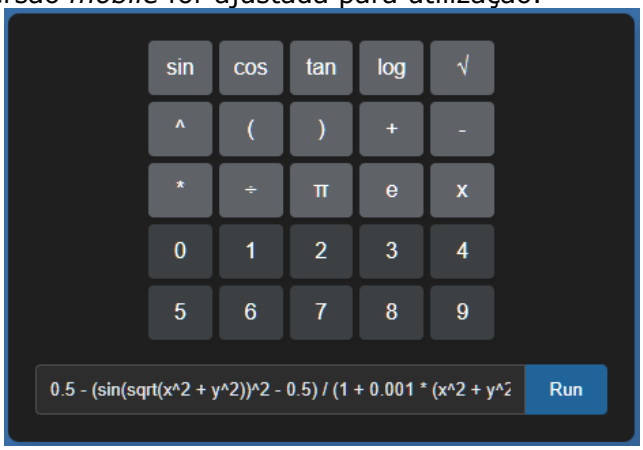


Figure 17: Teclado númerico para inserção de funções

b Componentes de Visualização de Resultados

Após a configuração dos parâmetros e a definição da função a ser otimizada, a interface exibe os resultados dos experimentos em uma série de contêineres visuais. A interface desenvolvida apresenta uma série de componentes gráficos e tabelares que exibem os resultados obtidos durante a execução da aplicação. Abaixo estão detalhados os principais componentes de visualização, ilustrados com exemplos e explicações detalhadas.

1 Gráfico de Média de Melhor *Fitness* por Geração

Este gráfico de linha apresenta a média das melhores *fitness* por geração entre as rodadas[11] realizadas.

A Figura 18 apresenta a evolução do *fitness* médio por geração para as 3 rodadas realizadas:

- **Eixo X:** Gerações.
- **Eixo Y:** *Fitness* médio.
- **Linhas:** Cada linha representa uma rodada e é diferenciada por uma cor.

[11] Considerando rodada como o ato de, simplesmente, clicar em *Run* e processar uma execução.

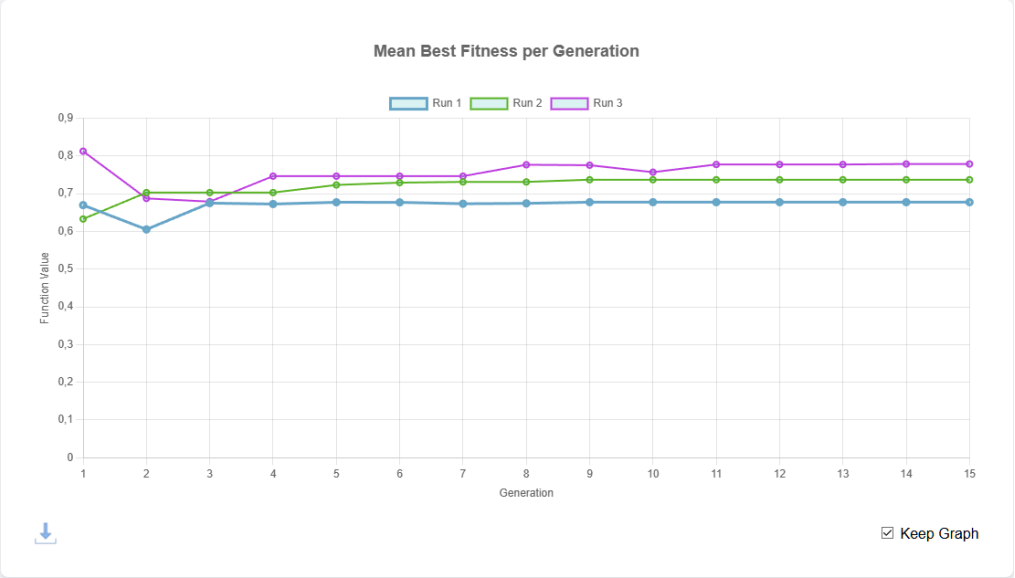


Figure 18: Gráfico de Média do Melhor Fitness por Geração para as três rodadas realizadas.

2 Tabela de Resultados

A tabela exibe os valores de melhor *fitness* para cada geração de todos os experimentos realizados, juntamente com a média entre os experimentos. Ela permite uma análise quantitativa detalhada.

A Figura 19 apresenta um exemplo da tabela com os melhores valores de *fitness* por geração para os 3 experimentos realizados na última rodada. Inclui:

- **Coluna "Generations":** Lista as gerações.
- **Colunas dos Experimentos:** Exibem os valores de *fitness* de cada experimento.
- **Coluna "Average":** Calcula a média dos valores fitness para cada geração.

Generations	Experiments			Average
	1º	2º	3º	
gen 1	0.6528	0.9398	0.8459	0.8128
gen 2	0.6528	0.5640	0.8459	0.6876
gen 3	0.6098	0.5815	0.8459	0.6791
gen 4	0.6342	0.6838	0.9218	0.7466
gen 5	0.6342	0.6838	0.9218	0.7466
gen 6	0.6342	0.6838	0.9218	0.7466
gen 7	0.6342	0.6838	0.9218	0.7466
gen 8	0.7248	0.6838	0.9218	0.7768
gen 9	0.7214	0.6838	0.9218	0.7757
gen 10	0.6658	0.6838	0.9218	0.7571
gen 11	0.7271	0.6838	0.9218	0.7776
gen 12	0.7271	0.6838	0.9218	0.7776
gen 13	0.7271	0.6838	0.9218	0.7776
gen 14	0.7272	0.6878	0.9218	0.7789
gen 15	0.7272	0.6878	0.9218	0.7789

Figure 19: Tabela de *Fitness*: Resumo por geração para os três experimentos.

3 Parâmetros Utilizados

Os parâmetros escolhidos e utilizados na execução da aplicação são apresentados em uma tabela para referência, melhorando a experiência o usuário. Isso inclui número de experimentos, gerações, tamanho da população, taxa de mutação, tipo de cruzamento, entre outros.

A Figura 20 exibe os parâmetros utilizados na execução da última rodada:

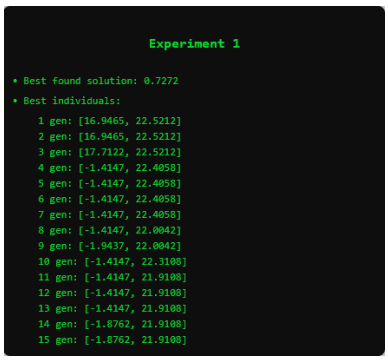
- **Número de Experimentos:** 3.
- **Número de Gerações:** 15.
- **Tamanho da População:** 30.
- Outros parâmetros como taxa de crossover, taxa de mutação, tipo de crossover, entre outros.

Best Fitness Per Generation & Experiments (Run 5)	
Number of Experiments:	5
Number of Generations:	40
Population Size:	100
Crossover Rate:	65%
Mutation Rate:	0.8%
Intent:	Minimize
Interval:	[-100,100]
Crossover Type:	One Point
Normalize Linear:	No
Elitism:	No
Steady State:	No
Steady State Without Duplicates:	No
Gap:	No
Execution Time:	25.92 seconds

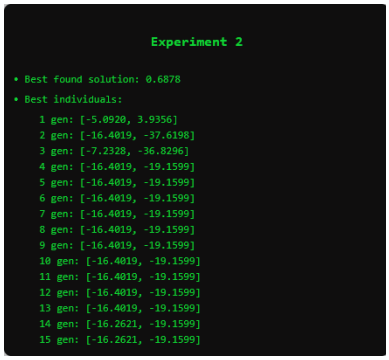
Figure 20: Parâmetros utilizados na execução da última rodada.

4 Visualização de Melhor Solução e Indivíduos por Experimento

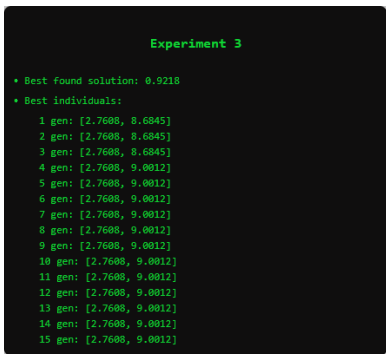
Cada experimento realizado exibe as informações sobre a melhor solução encontrada e os indivíduos correspondentes a cada geração. Isso permite analisar como os indivíduos evoluíram ao longo das gerações.



(a) Experiment 1 - Melhor solução: 0.7272.



(b) Experiment 2 - Melhor solução: 0.6878.



(c) Experiment 3 - Melhor solução: 0.9218.

Figure 21: Resultados dos indivíduos e melhores soluções para os três experimentos da última rodada.

5 Box Plot de Melhor *Fitness* por Geração

O gráfico de boxplot exibe a distribuição da melhor *fitness* por geração para todos os experimentos realizados na rodada. Esse gráfico ilustra como os valores evoluíram ao longo das gerações e permite identificar padrões e *outliers* (Hawkings, 1980 [72]). [12].

[12]Segundo Hawkings (1980) [72], Outliers são elementos de dados que se desviam significativamente dos demais e podem influenciar negativamente a análise de dados.

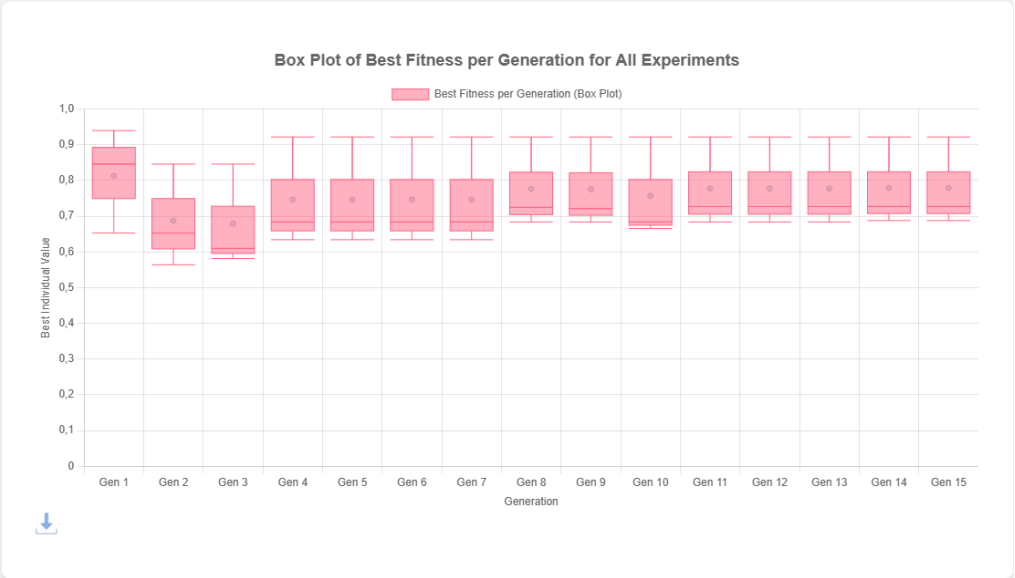
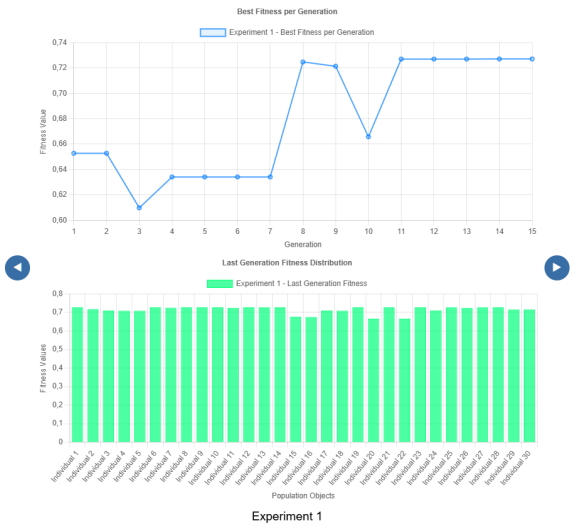


Figure 22: Box Plot: Melhor *Fitness* por Geração para todos os experimentos da última rodada.

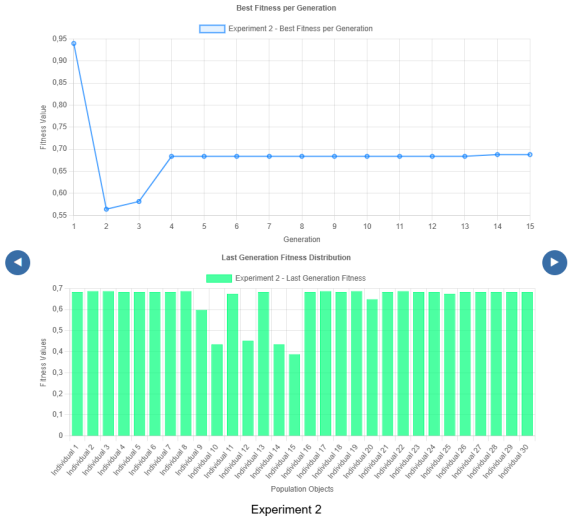
6 Gráficos de Evolução de *Fitness* por Experimento

Para cada experimento da rodada, são apresentados dois gráficos:

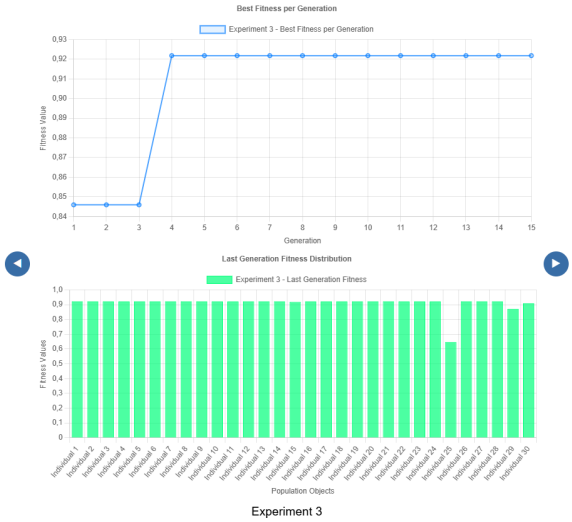
- **Gráfico de linha**: Mostra a melhor *fitness* encontrada em cada geração.
- **Gráfico de Barras**: Exibe a distribuição dos *fitness* dos indivíduos na última geração.



(a) Experiment 1.



(b) Experiment 2.



(c) Experiment 3.

Figure 23: Gráficos de linha e gráficos de barras para os três experimentos da última rodada.

8 Resultados e Discussão

a Limitações do GADEMO

O Gademo foi desenvolvido para otimizar funções de até duas variáveis (x_1 e x_2), ou seja, funções que podem ser representadas como $f(x)$ e funções que podem ser representadas como $f(x_1, x_2)$. Essa limitação não é exatamente uma questão de implementação, mas está intrinsecamente ligada à nossa capacidade de visualizar tais funções no espaço geométrico.

1. Funções de uma variável ($f(x)$):
   - Quando uma função depende de uma única variável, sua representação gráfica é bidimensional: um eixo representa a variável independente (x) e o outro, a variável dependente ($f(x)$).
   - Exemplo: o gráfico de uma parábola $f(x) = x^2$.
2. Funções de duas variáveis $f(x, y)$:
   - Quando a função depende de duas variáveis, a representação gráfica requer um espaço tridimensional: dois eixos representam as variáveis independentes (x e y) e o terceiro eixo, a variável dependente $f(x, y)$.
   - Exemplo: o paraboloide $f(x, y) = x^2 + y^2$
3. Funções de três ou mais variáveis ($f(x_1, x_2, ..., x_n)$):
   - Representar funções de três ou mais variáveis exige um número maior de dimensões, o que torna a visualização direta impossível[13]. Nessas situações, recorremos a representações parciais (projeções ou fatias do espaço), ou nos limitamos a análises numéricas e estatísticas para entender o comportamento da função.

Portanto, restringir o GADEMO a funções de até duas variáveis tem como principal vantagem a visualização intuitiva e tangível. Em funções de uma ou duas variáveis, é possível observar diretamente a superfície gerada por $f(x, y)$, o que facilita tanto a compreensão do comportamento da função quanto a análise de seu mínimo ou máximo. Essa abordagem é fundamental em estudos introdutórios e em aplicações práticas que requerem uma análise gráfica do processo de otimização.

b Análise dos Resultados

O presente trabalho gerou um conjunto significativo de dados relacionados ao desempenho dos algoritmos genéticos implementados na plataforma GADEMO. Nesta subseção, serão apresentados os principais resultados obtidos, com o objetivo de destacar as contribuições da aplicação desenvolvida e validar a eficácia das configurações realizadas. A análise gráfica inclui o comportamento do *fitness* médio ao longo das gerações, as distribuições dos valores de *fitness* por experimento, e os resultados individuais por geração. As tabelas sumarizam os dados de maneira a facilitar a identificação de padrões e tendências entre as execuções. Essas representações visuais e quantitativas são fundamentais para compreender o impacto dos parâmetros configurados na eficiência dos algoritmos.

c Configuração dos Testes

Para a realização dos experimentos, foram definidas configurações padronizadas para garantir consistência e comparabilidade dos resultados, levando em consideração práticas comuns na literatura de algoritmos genéticos. Abaixo, são descritos os parâmetros fixos adotados, juntamente com as justificativas para suas escolhas:

- **Número de rodadas**: As configuração de algoritmo genético foram testadas com números de rodadas variadas. Os números de rodadas foram selecionados visando a redução dos

[13] Do ponto de vista da física moderna, vivemos em quatro dimensões, onde o tempo é tão real quanto o espaço. Porém, enquanto as três dimensões espaciais são tangíveis e visualizáveis, o tempo é a "dimensão rebelde" que flui de forma invisível e relativa. Carl Sagan talvez dissesse: "Estamos presos à superfície de uma bolha quadridimensional, tentando entender o cosmos com uma régua de três dimensões".

efeitos de aleatoriedade inerentes aos algoritmos genéticos, permitindo a obtenção de métricas estatísticas relativamente confiáveis, como média e desvio padrão dos resultados (Montgomery, 2014 [73]).

- **Método de seleção - Torneio (*Tournament Selection*)**: Utilizou-se o método de *seleção por torneio*, descrito em 1, com tamanho do torneio (*tournament size*) igual a 3.
- **Elitismo**: A estratégia de elitismo foi implementada com base no cálculo da quantidade de elites preservadas, conforme a função `calculate_elite_count` descrita em g. A preservação de indivíduos com melhor *fitness* por geração é uma prática amplamente utilizada para evitar a perda de soluções promissoras ao longo das iterações. O número de elites varia dinamicamente com o tamanho da população, seguindo os critérios:
  - População ≤ 5: Nenhum indivíduo elite é preservado.
  - $6 \leq$ População ≤ 10: 1 elite.
  - População ≥ 11: 2 elites.

  Essa abordagem balanceada permite ajustar o nível de elitismo proporcionalmente ao tamanho da população, evitando tanto a convergência prematura quanto a ausência de pressão seletiva.
- **Avaliação inicial da população**: Toda a população inicial é avaliada com base na função objetivo fornecida. Esta etapa é essencial para determinar o *fitness* inicial de cada indivíduo, permitindo ao algoritmo genético identificar as regiões promissoras do espaço de busca desde as primeiras gerações. Esta abordagem, embora não seja a mais ideal, é comum em Algoritmos Genéticos, garantindo que a seleção inicial se baseie em informações reais da função objetivo.
- **Critério de parada**: O critério de parada adotado foi o número fixo de gerações para cada experimento, conforme especificado na Tabela 7. Este número foi definido com base em experimentos preliminares, nos quais foi observado que 40 gerações foram suficientes para alcançar a estabilização dos valores de *fitness* na maioria dos cenários testados.
- **Intervalo de busca**: Foi fixado em $[-100, 100]$, sendo válido tanto para x quanto para y (ao tratar-se de funções de duas variáveis).

A Tabela 7 apresenta as principais configurações utilizadas nos testes, que serão variados de forma seccionada.

GA	*nº Exp.*	*Generations*	*Population*	*Crossover (%)*	*Mutation (%)*	*Linear Norm.*	*Elitism*	*Steady State*
1	4	40	100	65.0	0.8	*n.a.*	*n.a.*	*n.a.*
2	4	40	100	65.0	0.8	Max=100/Min=1	*n.a.*	*n.a.*
3	4	40	100	65.0	0.8	*n.a.*	*Yes*	*n.a.*
4	4	40	100	65.0	0.8	*n.a.*	*n.a.*	*with duplicates*
5	4	40	100	65.0	0.8	*n.a.*	*n.a.*	*No duplicateds*

Table 7: Configurações dos Parâmetros no GADEMO

d Experimentos

Para avaliar o desempenho do GADEMO na otimização de diferentes tipos de funções de *benchmarck*[14], foram conduzidos múltiplos experimentos em diferentes configurações de parâmetros, conforme descrito em c. O objetivo principal é verificar:

1. Avaliar se o GADEMO consegue consistentemente localizar o ponto ótimo $x = (0, 0)$ em diferentes execuções, para diferentes tipos de funções.
2. Analisar a velocidade de convergência para o ponto ótimo ao longo das gerações.
3. Identificar a variabilidade nos resultados entre múltiplas execuções, dada a natureza estocástica doa algoritmos genéticos.

[14] No apêndice A é possível encontrar mais detalhes sobre algumas funções que serão mencionadas neste trabalho.

1 Taxas de Crossover e Mutação

Para os testes voltados a enfatizar o *crossover* e mutação, foi selecionada a função de Rastrigin[15], cujo mínimo global é localizado em

$$f(0,0) = 0 \tag{7}$$

1. **Objetivo**: Deseja-se avaliar como a redução significativa da taxa de crossover (10%) e o aumento expressivo da taxa de mutação (80%) influenciam o desempenho do algoritmo genético.
2. **Configuração de Parâmetros**:
   (a) **Cenário Base** $\longrightarrow$ **GA1** (ver tabela 7)
   - Taxa de Crossover: 65%.
   - Taxa de Mutação: 0.8%.
   - Rodadas: 3 rodadas independentes para obter médias confiáveis.
   - Gerações: 40.
   - Experimentos: 4.
   - Tamanho da População: 100 indivíduos.
   - Critério de parada: atingir o número determinado de gerações por experimento.

   (b) **Cenário Experimental** $\longrightarrow$ **GA1 modificado**:
   - Taxa de crossover: 10%.
   - Taxa de mutação: 80%.
   - Mantidos os outros mesmos parâmetros do cenário base.

Rodadas para o Cenário Base

1. **Resumo Geral de Todas as Rodadas**: De acordo com o resultado consolidado apresentado na figura 24, todas as rodadas exibem uma tendência clara de redução no valor do melhor *fitness* ao longo das gerações. Isso indica que o algoritmo está convergindo eficientemente para soluções mais próximas do ótimo global.

   Observa-se uma queda acentuada nos valores de *fitness* durante as primeiras 10 gerações. Isso sugere que as soluções iniciais estão longe do ótimo, mas o *crossover* e a mutação estão desempenhando bem suas funções para explorar o espaço de busca rapidamente.

   Após cerca da 15ª geração, os valores começam a se estabilizar, com reduções menos significativas. Isso reflete o momento em que a exploração diminui e a população começa a se concentrar em áreas próximas ao ótimo.

   Apesar de a tendência ser semelhante entre as rodadas, existem pequenas diferenças nos valores finais de *fitness*. Isso pode ser atribuído à natureza estocástica dos algoritmos genéticos, que dependem das condições iniciais e das probabilidades envolvidas nos operadores genéticos.

   No início, as diferenças entre os valores de *fitness* de cada rodada são mais evidentes, mas, com o avanço das gerações, essas diferenças se reduzem. Isso indica uma convergência consistente do algoritmo.

[15] A leitura sobre a função de Rastrigin disponível no apêndice 1 é fortemente recomendada.

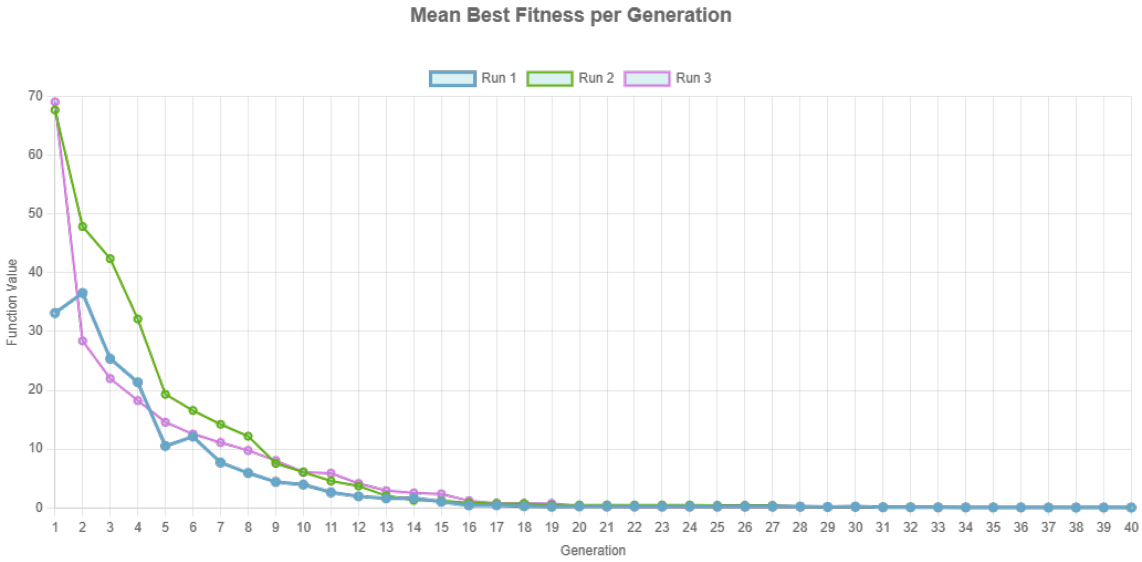


Figure 24: Gráfico consolidado de todas as rodadas - média de melhor *fitness* por geração em cada experimento

2. **Detalhamento Por Rodada**:
   - **Rodada 1**:

Tempo de Execução: O tempo de execução de aproximadamente 21,77 segundos, destacado na figura 25, observado na Rodada 1 para o cenário base da função Rastrigin, é coerente com a configuração dos parâmetros do algoritmo genético. A taxa de *crossover* de 65% e a mutação de 0,8%, combinados com o tamanho da população de 100 indivíduos ao longo de 40 gerações, indicam uma eficiência computacional interessante. Esse tempo sugere que a implementação está balanceada, considerando o esforço computacional para uma função complexa como a Rastrigin.

Vale adiantar que o tempo foi consistente entre as demais rodadas, o que reforça a confiabilidade da execução. Pequenas discrepâncias podem surgir devido a fatores externos ao algoritmo, como o ambiente computacional (processador ocupado com outras tarefas, por exemplo). Esse dado pode ser relevante para avaliar o desempenho global do sistema.

Best Fitness Per Generation & Experiments (Run 1)

Number of Experiments:	4
Number of Generations:	40
Population Size:	100
Crossover Rate:	65%
Mutation Rate:	0.8%
Intent:	Minimize
Interval:	[-100,100]
Crossover Type:	One Point
Normalize Linear:	No
Elitism:	No
Steady State:	No
Steady State Without Duplicates:	No
Gap:	No
Execution Time:	21.77 seconds

Figure 25: Tempo de execução da Primeira rodada - Otimização da função de Rastrigin

Box-Plot:

O box-plot da rodada 1 para o cenário base fornece *insights* importantes sobre o comportamento do AG ao longo das gerações, como observado na figura 26.

Primeiras Gerações (Gen 1 a 4): Há uma alta variabilidade nos valores de *fitness*, evidenciada pelos amplos intervalos interquartis e pela presença de *outliers* (Hawkings, 1980 [72]). A mediana está significativamente mais alta, indicando que, inicialmente, as soluções estão bem distantes do ótimo global. Isso reflete a diversidade inicial da população, algo esperado nas primeiras gerações devido à busca exploratória do algoritmo.

Transição e Convergência (Gen 5 a 10): A partir da geração 5, a dispersão começa a diminuir, com os quartis se estreitando gradualmente. A mediana e os quartis inferiores indicam uma melhoria consistente na qualidade das soluções. Este é o período em que o algoritmo passa de uma fase de exploração para uma fase mais refinada de exploração e início de convergência.

Convergência (Gen 11 em diante): A partir da geração 11, observa-se uma estabilização significativa nos valores de *fitness*. O intervalo interquartil é mínimo, e as medianas permanecem baixas, mostrando que as soluções estão muito próximas entre si e próximas do valor ótimo esperado. Praticamente não há *outliers* (Hawkings, 1980 [72]), o que sugere que o algoritmo encontrou e mantém uma solução ótima ou próxima do ótimo global.

A diminuição da variabilidade ao longo das gerações indica que o algoritmo conseguiu explorar bem o espaço de soluções no início e convergir adequadamente para a região de interesse. A presença de *outliers* nas primeiras gerações reforça a consistência do processo de seleção e *crossover*, já que o algoritmo descartou essas soluções menos aptas à medida que evoluiu.

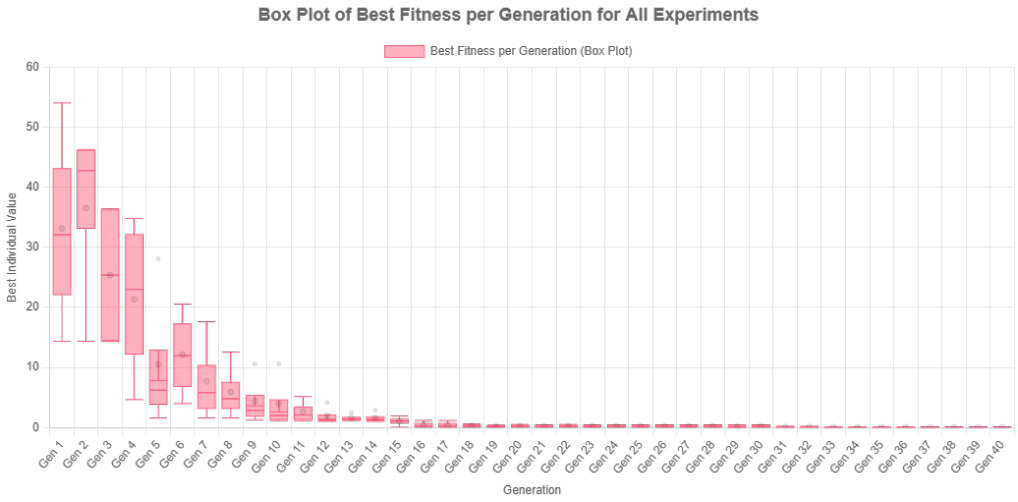


Figure 26: Box-Plot da primeira rodada - Otimização da função de Rastrigin

Melhor *Fitness* por Geração em Cada Experimento:

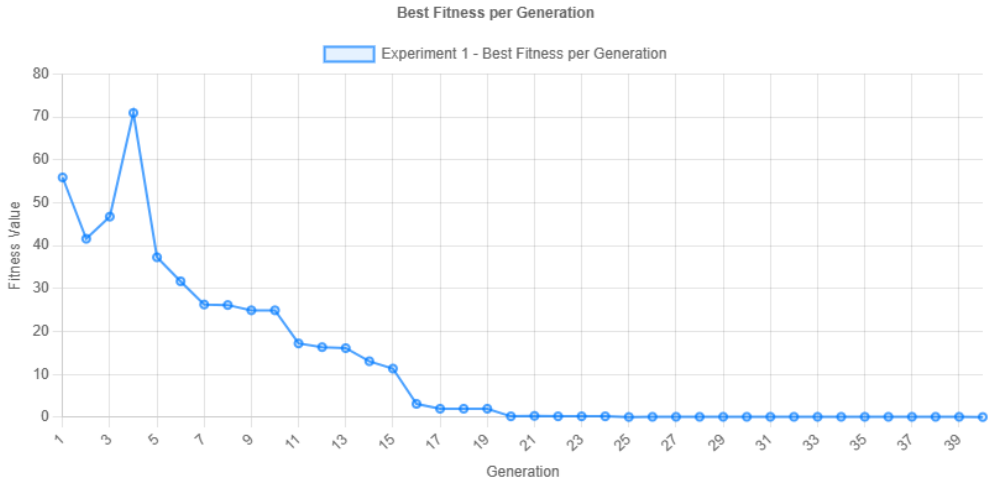


Figure 27: Gráfico de melhor *fitness* por geração no primeiro experimento da primeira rodada.

O gráfico do primeiro experimento apresenta um *fitness* inicial de aproximadamente 70. Nas primeiras gerações, há uma redução significativa dos valores de *fitness*, atingindo estabilização por volta da 10ª geração. Após a 15ª geração, os valores se mantiveram praticamente constantes, indicando que o algoritmo convergiu para uma solução ótima.

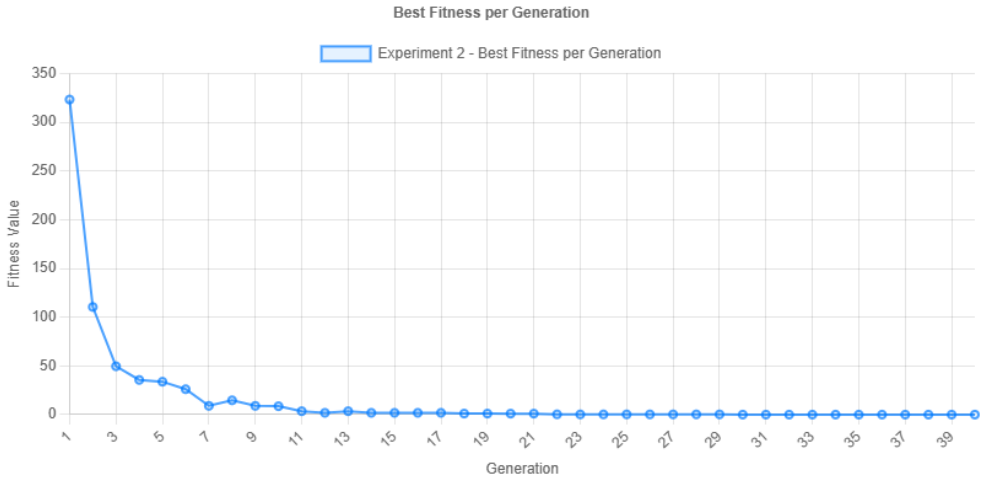


Figure 28: Gráfico de melhor *fitness* por geração no segundo experimento da primeira rodada.

No segundo experimento, o *fitness* inicial foi consideravelmente alto (≈ 350), mas apresentou uma rápida redução nas primeiras 10 gerações. A partir desse ponto, os valores estabilizaram próximos de zero, demonstrando uma boa eficiência do algoritmo em encontrar soluções, apesar do ponto de partida elevado.

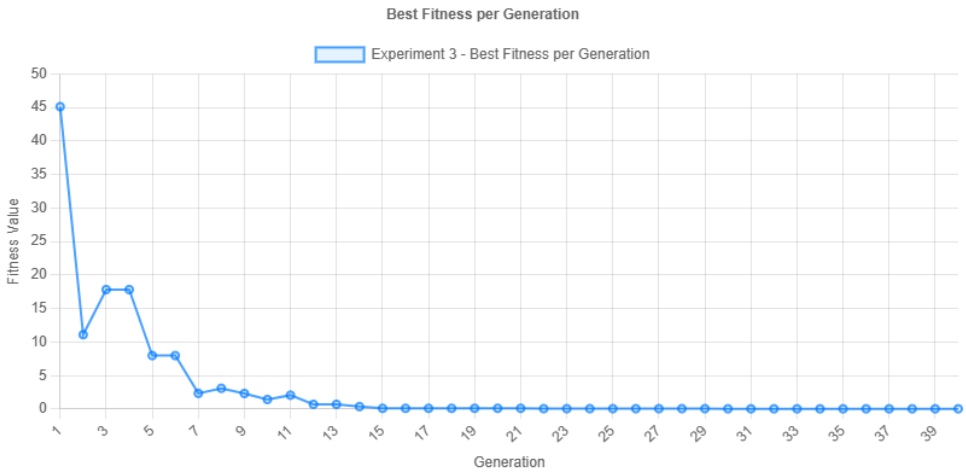


Figure 29: Gráfico de melhor *fitness* por geração no terceiro experimento da primeira rodada.

No terceiro experimento, o *fitness* inicial de cerca de 50 apresentou uma redução rápida e significativa. O gráfico mostra estabilização próxima de zero antes da 10ª geração, indicando que este experimento teve uma convergência mais rápida e eficiente em comparação aos outros.

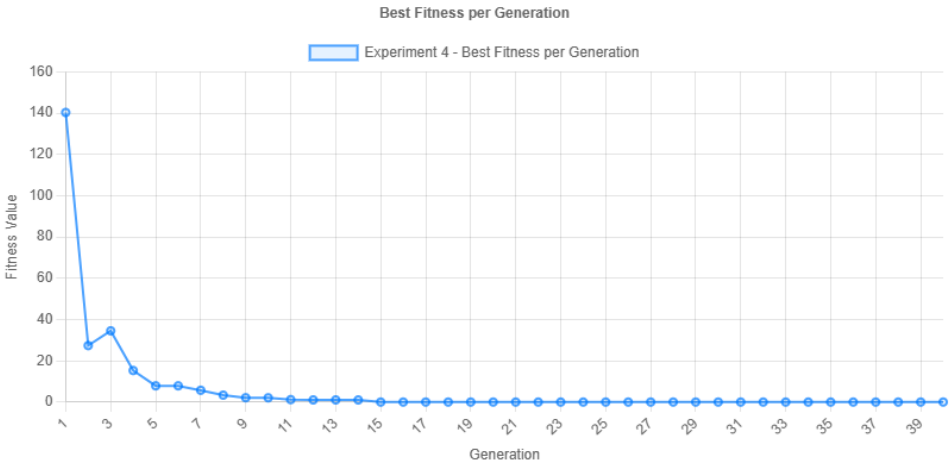


Figure 30: Gráfico de melhor *fitness* por geração no quarto experimento da primeira rodada.

No quarto experimento, o *fitness* inicial foi de aproximadamente 160, com uma queda acentuada nas primeiras gerações. A estabilização ocorreu por volta da 8ª geração, o que demonstra um ajuste eficiente inicial e convergência satisfatória nas gerações finais.

Gráficos de barras da última geração de cada experimento:

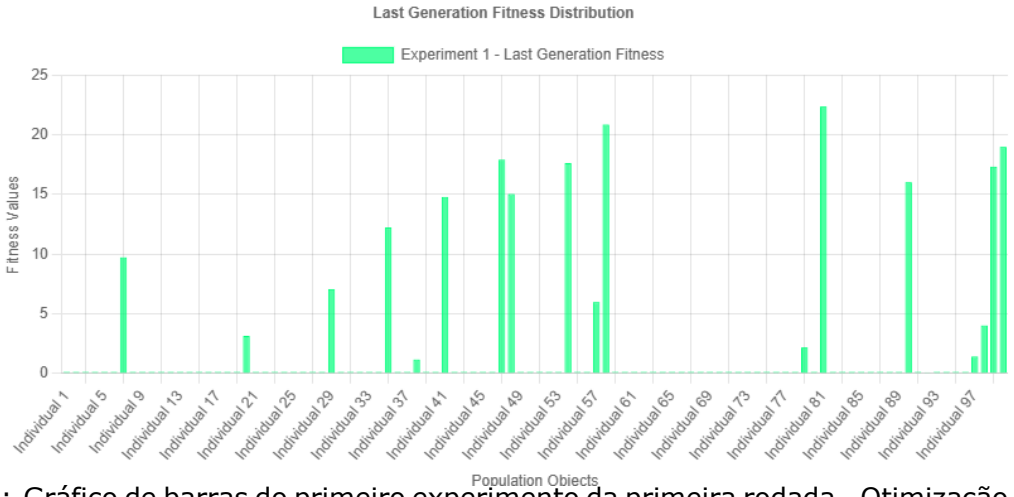


Figure 31: Gráfico de barras do primeiro experimento da primeira rodada - Otimização da função de Rastrigin

O gráfico de barras do primeiro experimento (31) mostra que a maioria dos indivíduos nas últimas gerações apresenta valores de *fitness* baixos, com algumas exceções que alcançam valores acima de 20. Isso indica que a pressão seletiva do algoritmo genético foi eficaz em concentrar a população em torno de soluções com menor valor de *fitness*. No entanto, há uma pequena dispersão que sugere a presença de indivíduos menos adaptados, o que pode ser útil para evitar a estagnação em ótimos locais.

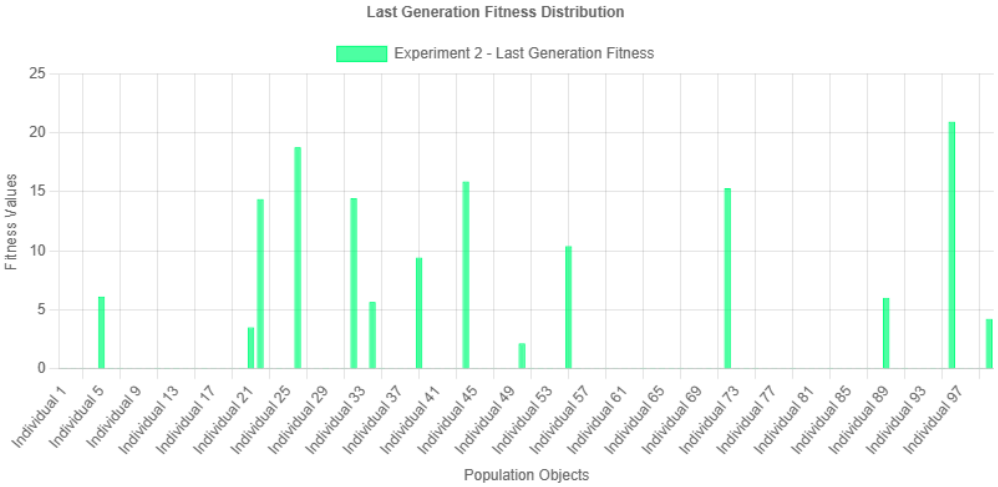


Figure 32: Gráfico de barras do segundo experimento da primeira rodada - Otimização da função de Rastrigin

O gráfico do segundo experimento (32) apresenta um comportamento similar ao primeiro, com a maioria dos indivíduos concentrados em *fitness* próximos a zero. Neste caso, há uma dispersão um pouco maior em relação aos valores intermediários, o que pode sugerir um ajuste diferente na busca por soluções ótimas, talvez devido à diversidade inicial da população ou mutações mais exploratórias.

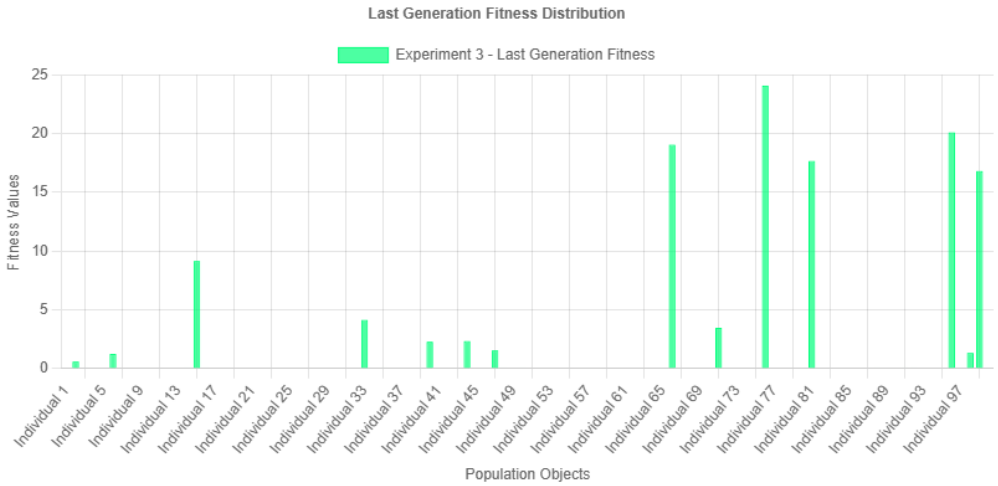


Figure 33: Gráfico de barras do terceiro experimento da primeira rodada - Otimização da função de Rastrigin

O terceiro experimento (33) destaca uma maior concentração de indivíduos com *fitness* próximos a zero, indicando que a convergência foi mais eficiente aqui do que nos experimentos anteriores. Os poucos indivíduos com *fitness* acima de 15 mostram que a diversidade foi mantida em menor escala, mas ainda está presente, o que evita problemas de convergência prematura.

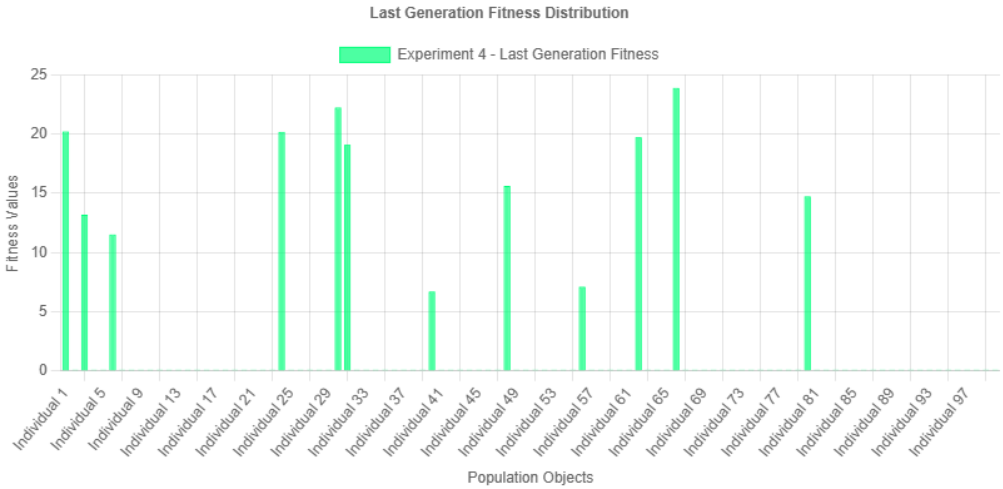


Figure 34: Gráfico de barras do quarto experimento da primeira rodada - Otimização da função de Rastrigin

No quarto experimento (34), os valores de *fitness* são majoritariamente baixos, com alguns picos em torno de 20. Isso reflete uma boa exploração do espaço de busca, mas com manutenção de indivíduos que não convergiram completamente para as melhores soluções. A distribuição neste experimento parece balancear melhor a exploração e a exploração em comparação aos demais.

Média do melhor *Fitness* por geração: 0.0310

Performance do AG longo de 40 gerações:

*** Média dos *fitnesses* com o passar das gerações ***

A análise da média dos *fitness* ao longo das 40 gerações permite compreender a convergência do Algoritmo Genético aplicado à função Rastrigin e fornecen informações sobre sua eficiência e comportamento.

Fase Inicial (Gerações 1 a 5): As médias dos *fitnesses* iniciam com valores elevados (em torno de 33.1532 na primeira geração), conforme é evidenciado na figura 35. Observa-se uma redução significativa já nas primeiras gerações, o que indica que o AG encontrou soluções melhores rapidamente, caracterizando uma forte pressão seletiva inicial.

Fase de Ajuste (Gerações 6 a 15): Nesta fase, os valores médios de *fitness* continuam a diminuir, mas a taxa de redução desacelera. Por exemplo, na geração 10, a média é de 3.9443, comparada aos 12.1241 na geração 6. Isso reflete o esgotamento gradual das soluções de alta *fitness* na população inicial, com o AG se concentrando em explorar soluções próximas ao ótimo.

Fase de Convergência (Gerações 16 a 40): A partir da geração 16, os valores médios estabilizam progressivamente em torno de 0.0310. Esse comportamento sugere que o AG convergiu para soluções próximas ao ótimo global da função Rastrigin, com pequenas variações nas últimas gerações.

Análise das Médias Finais: A média estabilizada em 0.0310 nas últimas gerações indica que o AG teve sucesso em minimizar a função Rastrigin. Essa estabilidade reflete o equilíbrio entre exploração e exploração alcançado pelas configurações do AG.

Generations	Experiments				Average
	1°	2°	3°	4°	
gen 1	39.4791	54.0561	24.7413	14.3363	33.1532
gen 2	39.4791	46.1068	46.2916	14.3363	36.5535
gen 3	36.1682	14.5951	36.4243	14.3363	25.3810
gen 4	31.2324	14.7435	34.8191	4.6482	21.3608
gen 5	1.5689	7.7873	28.0963	4.6482	10.5252
gen 6	16.1467	7.7873	20.5821	3.9802	12.1241
gen 7	1.5689	7.8895	17.6256	3.7250	7.7022
gen 8	1.5689	5.8482	12.5336	3.7250	5.9189
gen 9	1.3013	3.6183	10.6115	2.0895	4.4052
gen 10	1.4017	2.5976	10.6115	1.1663	3.9443
gen 11	1.4017	2.7653	5.1592	1.1663	2.6231
gen 12	1.4017	1.0367	4.1732	1.1663	1.9445
gen 13	1.4017	1.3666	2.4847	1.1663	1.6048
gen 14	1.4017	1.0367	2.8782	1.1663	1.6207
gen 15	0.0259	1.0367	1.8849	1.1663	1.0284
gen 16	0.0259	0.0151	0.4866	1.1663	0.4235
gen 17	0.0259	0.0151	0.4866	1.1624	0.4225

Figure 35: Média dos *fitnesses* da primeira à décima sétima geração em todos os experimentos na primeira rodada

gen 18	0.0259	0.0151	0.5774	0.3971	0.2539
gen 19	0.0259	0.0151	0.2361	0.3971	0.1685
gen 20	0.0259	0.0151	0.4866	0.3971	0.2312
gen 21	0.0259	0.0151	0.4137	0.3971	0.2129
gen 22	0.0259	0.0151	0.4866	0.3931	0.2302
gen 23	0.0259	0.0151	0.4137	0.3931	0.2120
gen 24	0.0259	0.0151	0.4137	0.3931	0.2120
gen 25	0.0259	0.0151	0.4137	0.3931	0.2120
gen 26	0.0259	0.0151	0.4137	0.3931	0.2120
gen 27	0.0259	0.0151	0.4137	0.3931	0.2120
gen 28	0.0259	0.0151	0.4137	0.3931	0.2120
gen 29	0.0147	0.0151	0.4137	0.1718	0.1538
gen 30	0.0147	0.0151	0.4137	0.3931	0.2092
gen 31	0.0147	0.0151	0.1146	0.1845	0.0822
gen 32	0.0147	0.0138	0.1146	0.1845	0.0819
gen 33	0.0147	0.0138	0.1146	0.1160	0.0648
gen 34	0.0147	0.0138	0.1146	0.0708	0.0535
gen 35	0.0147	0.0138	0.1146	0.0708	0.0535
gen 36	0.0147	0.0138	0.1146	0.0708	0.0535

Figure 36: Média dos *fitnesses* da décima oitava à trigésima sexta geração em todos os experimentos na primeira rodada

gen 37	0.0147	0.0138	0.0247	0.0708	0.0310
gen 38	0.0147	0.0138	0.0247	0.0708	0.0310
gen 39	0.0147	0.0138	0.0247	0.0708	0.0310
gen 40	0.0147	0.0138	0.0247	0.0708	0.0310

Figure 37: Média dos *fitnesses* das últimas quatro gerações em todos os experimentos na primeira rodada

Pode-se constatar que o desempenho do AG foi consistente, com uma rápida convergência inicial e estabilização em valores baixos de *fitness*, evidenciando a eficácia do cruzamento e da mutação nas configurações estabelecidas. O intervalo de *fitness* observado ao longo das gerações demonstra que o AG conseguiu explorar o espaço de busca adequadamente, reduzindo progressivamente a diversidade da população até alcançar soluções quase ótimas.

*** Melhores soluções em cada experimento*** :

Analisando as melhores soluções encontradas para cada experimento na primeira rodada do cenário base, observa-se que, no **experimento 1** (imagem 38), A melhor solução encontrada foi 0.0147. Isso indica que o algoritmo conseguiu convergir para uma solução de alta qualidade em termos de minimização. Esta solução está bem próxima do ideal, evidenciando que o algoritmo está ajustado adequadamente para explorar e refinar candidatos promissores.

No **experimento 2**, a melhor solução encontrada foi 0.0138 (fugura 39), um resultado ainda mais próximo do valor ideal. Esse desempenho reforça a robustez do algoritmo, especialmente em um cenário onde há variações no espaço de busca. A menor solução em relação ao Experimento 1 demonstra que o algoritmo conseguiu explorar melhor as regiões promissoras do espaço de soluções.

No **experimento 3** (figura 40), A melhor solução foi 0.0247. Embora seja ligeiramente

superior às encontradas nos dois primeiros experimentos, ainda representa um desempenho satisfatório dentro do contexto de um algoritmo genético. A maior variação pode estar relacionada a características específicas da população inicial ou a pequenas diferenças nas combinações geradas.

No **experimento 4** (figura 41), a solução encontrada foi 0.0708, a mais elevada entre os quatro experimentos. Este resultado sugere que, apesar de uma convergência aceitável, pode ter havido uma menor eficiência na exploração ou na explotação (Goldberg,1989 [2]) neste experimento em comparação aos outros. Pode ser interessante verificar se a diversidade populacional foi mantida adequadamente ao longo das gerações.

Portanto, os resultados mostram que os valores de melhor solução convergem para valores bastante próximos, com uma pequena variação entre os experimentos. Isso indica uma configuração consistente do algoritmo genético para a função Rastrigin. A dispersão observada é esperada em algoritmos estocásticos e reflete a importância de executar múltiplas rodadas para obter médias confiáveis e reduzir a influência de *outliers* (Hawkings, 1980 [72]). Os resultados globais evidenciam o equilíbrio alcançado entre exploração e explotação (Goldberg,1989 [2]), embora melhorias na diversidade ou na recombinação possam ser exploradas para reduzir variações entre os experimentos.

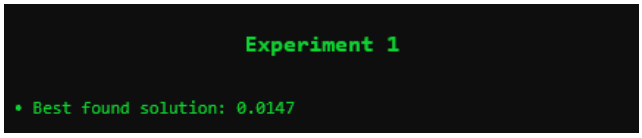


Figure 38: Melhor solução encontrada no primeiro experimento da primeira rodada

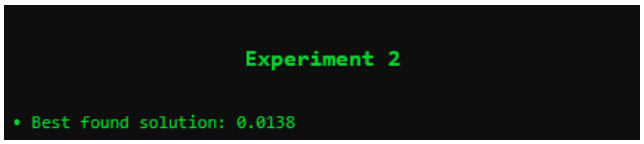


Figure 39: Melhor solução encontrada no segundo experimento da primeira rodada

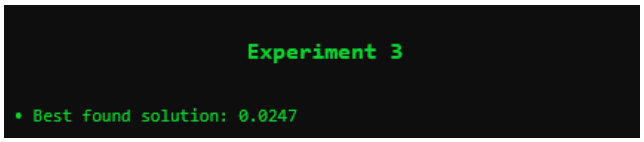


Figure 40: Melhor solução encontrada no terceiro experimento da primeira rodada

Figure 41: Melhor solução encontrada no quarto experimento da primeira rodada

- **Rodada 2**:

Tempo de Execução:

Na segunda rodada do cenário base, começando pela análise do tempo de execução, tem-se 21.02 segundos. Esse valor é bem consistente com o observado na primeira rodada, reforçando a estabilidade do tempo de processamento do algoritmo genético nas configurações estabelecidas.

A estabilidade do tempo de execução sugere que a complexidade do problema e as operações realizadas pelo algoritmo (como seleção, *crossover* e mutação) foram mantidas em um nível previsível. Essa consistência é essencial para análises comparativas entre diferentes rodadas e experimentos.

Best Fitness Per Generation & Experiments (Run 2)	
Number of Experiments:	4
Number of Generations:	40
Population Size:	100
Crossover Rate:	65%
Mutation Rate:	0.8%
Intent:	Minimize
Interval:	[-100,100]
Crossover Type:	One Point
Normalize Linear:	No
Elitism:	No
Steady State:	No
Steady State Without Duplicates:	No
Gap:	No
Execution Time:	21.02 seconds

Figure 42: Tempo de execução da segunda rodada - Otimização da função de Rastrigin

Box-Plot:

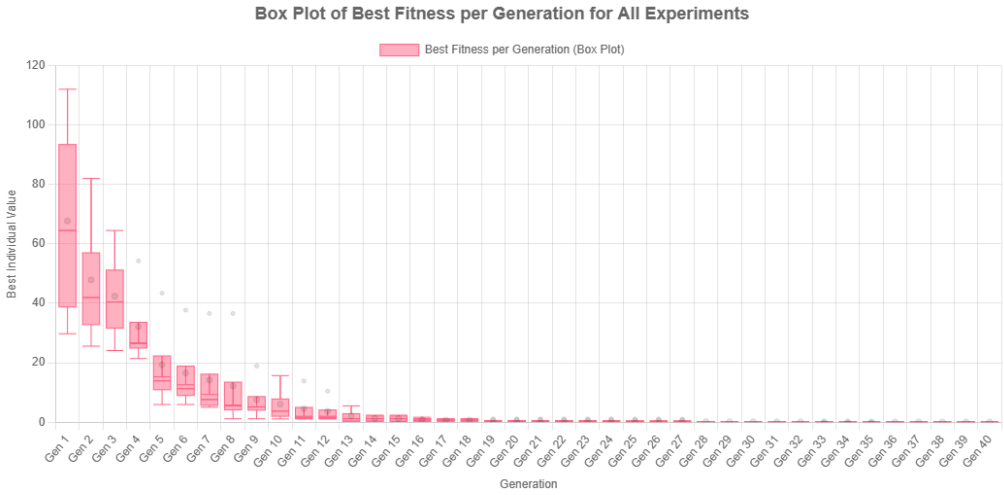


Figure 43: Box-Plot da segunda rodada - Otimização da função de Rastrigin

Primeira gração (Gen 1):

- **Mínimo**: 29,765 – representa o melhor indivíduo (*fitness* mais baixo) da população inicial.
- **Primeiro Quartil** (25%): 38,778 – indica que 25% dos indivíduos possuem *fitness* abaixo desse valor, evidenciando alta dispersão.
- **Mediana**: 64,512 – metade da população tem *fitness* abaixo desse valor, o que mostra que a maioria dos indivíduos está longe do ótimo.
- **Média**: 67,714 – valor médio reflete a predominância de *fitness* altos, indicando uma população inicial bastante heterogênea.
- **Terceiro Quartil** (75%): 93,448 – 75% da população tem *fitness* abaixo desse valor, reforçando a variabilidade inicial.
- **Máximo**: 112,067 – o indivíduo mais distante do ótimo.

A variabilidade inicial é alta, como esperado em gerações iniciais de algoritmos genéticos. Isso é essencial para garantir diversidade suficiente para a exploração.

Trigésima Nona Geração (Gen 39):

- **Mínimo**: 0,002 – reflete um progresso significativo, com o melhor indivíduo muito

próximo do ótimo.
- **Primeiro Quartil** (25%): 0,009 – boa parte da população já apresenta valores baixos de *fitness*, indicando convergência.
- **Mediana**: 0,07 – metade da população possui *fitness* abaixo desse valor, sugerindo homogeneidade crescente.
- **Média**: 0,069 – aproxima-se da mediana, confirmando distribuição equilibrada.
- **Terceiro Quartil** (75%): 0,129 – apenas 25% da população apresenta *fitness* acima deste valor, demonstrando a consolidação de soluções ótimas.
- **Máximo**: 0,132 – redução drástica comparada à Geração 1, indicando convergência da população em torno de soluções eficientes.

Na Geração 39, há uma clara estabilização, com a maioria dos indivíduos apresentando *fitness* próximos ao ótimo. O box-plot reflete o sucesso do algoritmo em explorar e depois explorar efetivamente o espaço de busca.

Melhor *Fitness* por Geração em Cada Experimento:

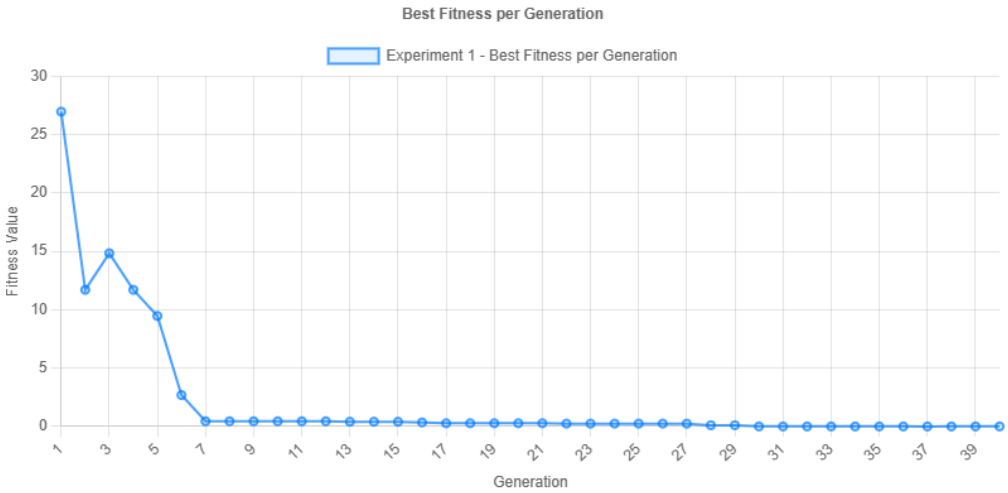


Figure 44: Gráfico de melhor *fitness* por geração no primeiro experimento da segunda rodada.

O gráfico do primeiro experimento (figura 44) indica uma melhora acentuada nas primeiras gerações, com o *fitness* reduzindo drasticamente, atingindo valores próximos de 0 por volta da décima quinta geração. Após isso, o *fitness* estabiliza, sugerindo que o algoritmo encontrou uma solução otimizada dentro desse intervalo de gerações.

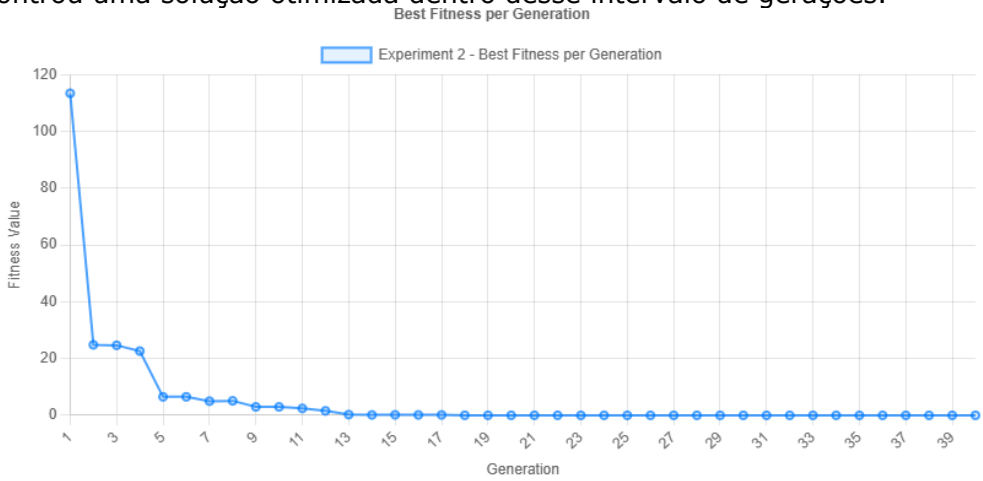


Figure 45: Gráfico de melhor *fitness* por geração no segundo experimento da segunda rodada.

No segundo experimento (figura 45), o comportamento inicial é mais abrupto, com uma

queda significativa do *fitness* na primeira geração. O *fitness* estabiliza mais rapidamente em comparação com o Experimento 1. Isso pode indicar um início mais favorável devido à diversidade inicial da população, ou talvez à seleção de combinações genéticas mais promissoras.

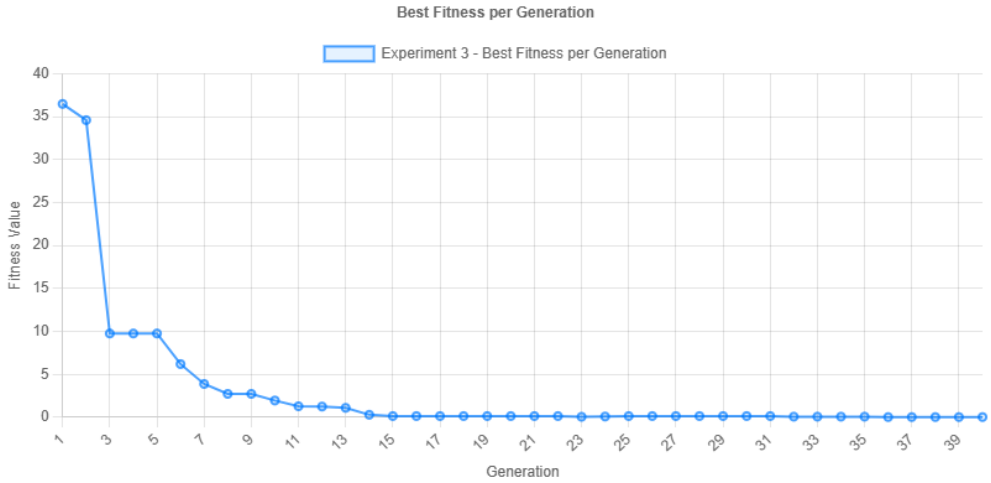


Figure 46: Gráfico de melhor *fitness* por geração no terceiro experimento da segunda rodada.

No terceiro experimento (figura 46), o padrão de redução no *fitness* é consistente, com uma descida progressiva até a décima geração, estabilizando-se a partir daí. Parece seguir um padrão equilibrado de exploração e explotação (Goldberg,1989 [2]), com uma convergência confiável para soluções boas.

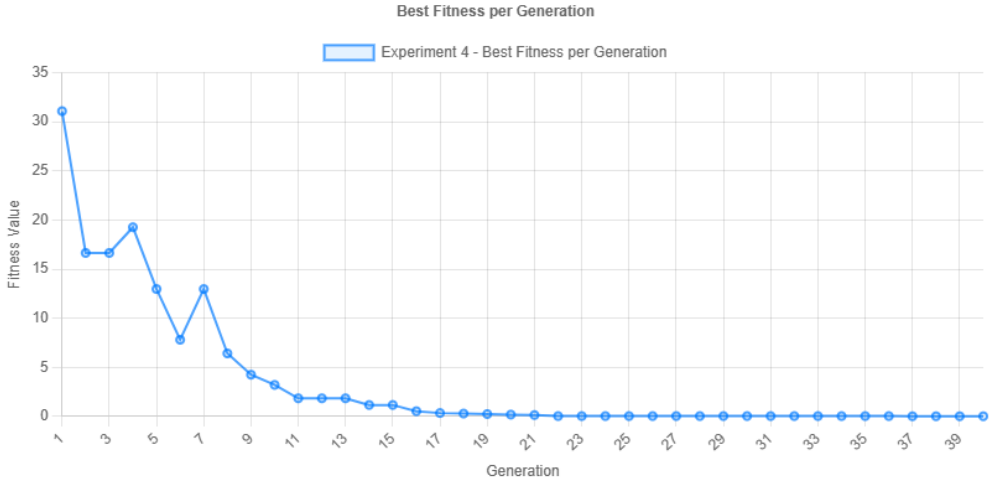


Figure 47: Gráfico de melhor *fitness* por geração no quarto experimento da primeira rodada.

No quarto experimento (figura 47, a redução no *fitness* é rápida nas primeiras gerações, mas apresenta um pequeno atraso em estabilizar-se em relação aos outros experimentos. Isso pode refletir uma população inicial com maior diversidade, exigindo mais iterações para alcançar uma solução otimizada.

Portanto, a rodada 2 deste cenário reforça o comportamento consistente do algoritmo genético. Em todos os experimentos, os melhores valores de *fitness* são atingidos até a décima quinta geração, demonstrando que as configurações permitem uma convergência eficiente. Apesar de pequenas variações no ritmo de estabilização, os resultados finais indicam soluções de alta qualidade em todos os casos até então.

Gráficos de Barras da última geração de cada experimento:

Observando de maneira geral, constata-se que a maioria dos indivíduos apresenta val-

ores de *fitness* baixos, indicando que o algoritmo convergiu para soluções próximas do ótimo. Contudo, ainda existem alguns indivíduos com valores de *fitness* relativamente altos em cada experimento, sugerindo que a diversidade genética foi mantida até certo ponto na última geração.

O **Experimento 1** (figura 48) apresenta alguns valores de *fitness* mais dispersos, especialmente com alguns indivíduos acima de 20. Isso pode indicar variações nos caminhos evolutivos devido às condições iniciais do algoritmo.

O **Experimento 2** (figura 49) tem um comportamento similar, com valores concentrados em níveis baixos, mas também com picos acima de 30, evidenciando algum grau de exploração tardia.

O **Experimento 3** (figura 50) se destaca por apresentar uma concentração ainda maior de indivíduos próximos ao mínimo de *fitness*, sugerindo que a exploração foi mais efetiva.

O **Experimento 4**, (figura 51) embora similar aos anteriores, também apresenta dispersões nos valores mais altos, especialmente acima de 20, o que demonstra variações individuais na otimização.

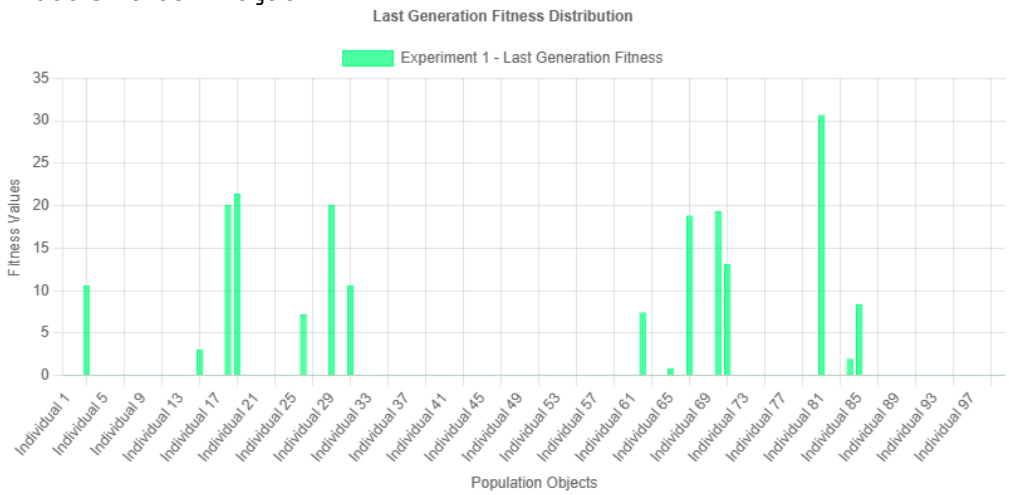


Figure 48: Gráfico de barras do primeiro experimento da segunda rodada - Otimização da função de Rastrigin

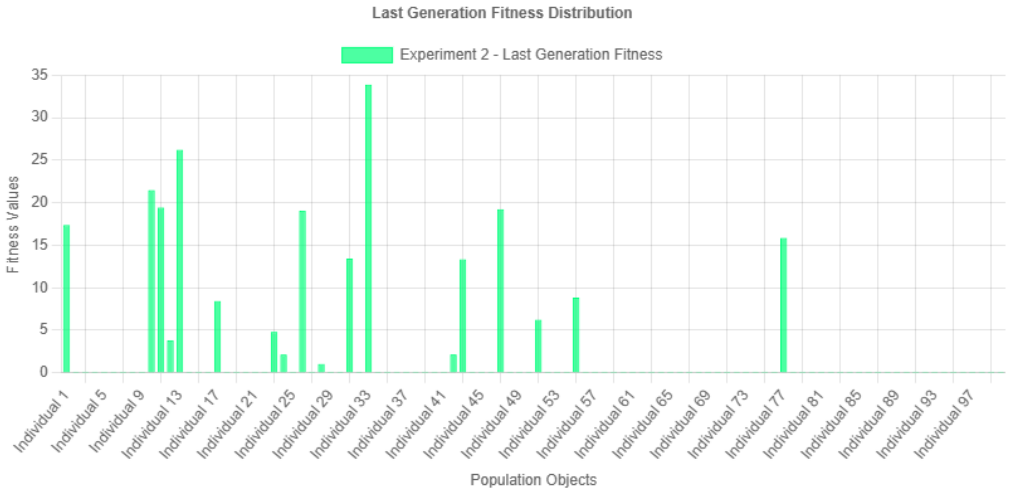


Figure 49: Gráfico de barras do segundo experimento da segunda rodada - Otimização da função de Rastrigin

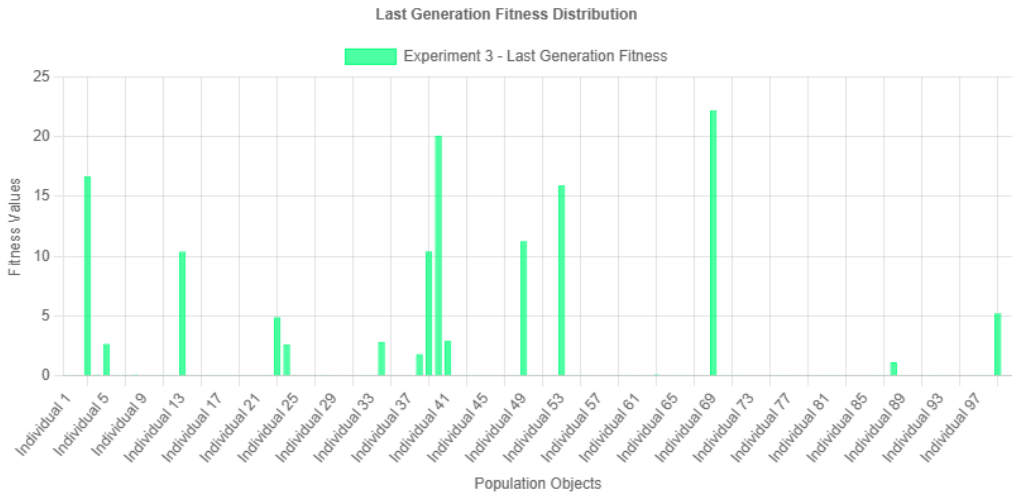


Figure 50: Gráfico de barras do terceiro experimento da segunda rodada - Otimização da função de Rastrigin

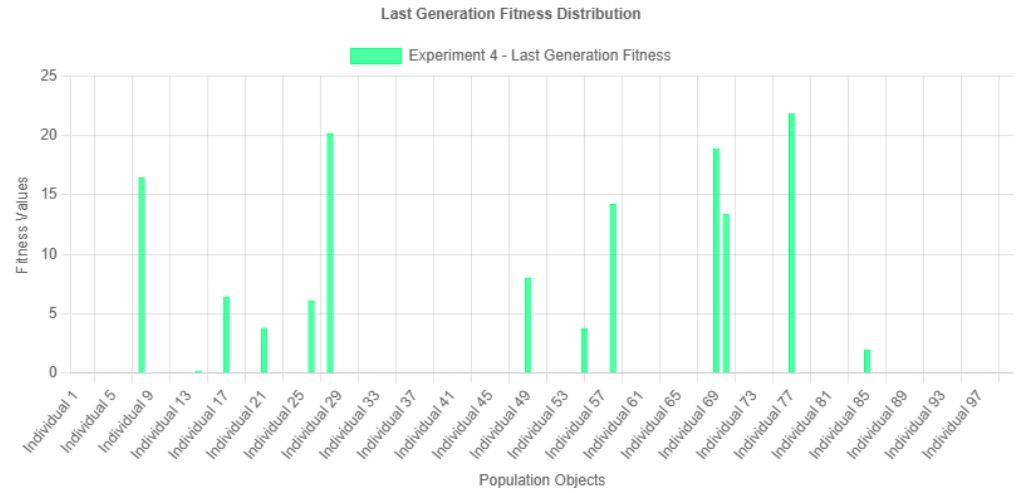


Figure 51: Gráfico de barras do quarto experimento da segunda rodada - Otimização da função de Rastrigin

Em todos os experimentos, há um padrão geral de concentração nos menores valores de *fitness*, indicando que o AG conseguiu realizar uma exploração eficiente do espaço de busca.

Os resultados indicam que o algoritmo manteve um equilíbrio entre exploração e explotação (Goldberg,1989 [2]), permitindo que algumas soluções de *fitness* mais altas persistissem na população enquanto convergia gradualmente para valores baixos em média.

A diversidade genética observada nos valores de *fitness* mais altos pode ser útil para evitar convergência prematura e explorar potenciais melhorias em iterações futuras.

Média do melhor *Fitness* por geração: 0.0685

Performance do AG longo de 40 gerações:

*** Média dos *fitnesses* com o passar das gerações ***

Os resultados da Rodada 2 do cenário base apresentam uma evolução que reforça as tendências observadas na Rodada 1.

Nas Gerações 1 a 4 (figura 52), nota-se uma redução substancial nos valores médios de *fitness*, com destaque para a Geração 1, que apresenta uma média inicial de 67.7139. Isso evidencia a eficácia do algoritmo em eliminar soluções menos otimizadas logo nas primeiras iterações. O impacto da exploração inicial é evidente, com uma dispersão significativa entre os valores de *fitness*, o que reflete uma busca ampla no espaço de soluções.

A partir da Geração 5, os valores médios começam a estabilizar de forma mais consistente. Por exemplo, a média na Geração 5 é 19.3108, e na Geração 10 já cai para 6.0616. Essa transição indica o início de uma fase de explotação (Goldberg,1989 [2]), onde o algoritmo foca em refinar as soluções mais promissoras.

Nas gerações 36 a 40 (figuras 53 e 54), o valor médio de *fitness* estabiliza em torno de *0.0685*, indicando que o algoritmo alcançou um platô de desempenho. A ausência de variações significativas nessa etapa mostra que as configurações do AG possibilitaram a convergência para soluções otimizadas. O ajuste fino (exploração local) dessas gerações demonstra o equilíbrio entre exploração e explotação (Goldberg,1989 [2]), característico de configurações consistentes.

Ao longo das gerações, os experimentos individuais mantêm uma convergência razoável entre si, com pequenas variações nos valores de *fitness*. Isso reflete solidez nos resultados, reforçando a confiabilidade do algoritmo configurado.

A semelhança entre os resultados médios das Rodadas 1 e 2 indica a robustez e previsibilidade do AG, uma característica desejável em problemas de otimização complexos como a função de Rastrigin. Então, a Rodada 2 confirma a estabilidade e eficiência do AG em identificar soluções otimizadas ao longo de 40 gerações. A evolução dos *fitnesses* médios demonstra uma transição natural entre exploração e explotação (Goldberg,1989 [2]), com desempenho consistente e resultados confiáveis em diferentes experimentos. Essas características tornam o algoritmo uma solução sólida para problemas de minimização em espaços de busca complexos.

Generations	Experiments				Average
	1º	2º	3º	4º	
gen 1	41.7818	112.0669	29.7651	87.2416	67.7139
gen 2	35.2562	48.6633	25.5951	82.0395	47.8885
gen 3	24.1253	46.7564	34.1937	64.5100	42.3963
gen 4	26.1129	26.7103	21.4428	54.3075	32.1434
gen 5	5.9497	12.5856	15.2676	43.4403	19.3108
gen 6	5.9497	12.5856	10.0120	37.7028	16.5625
gen 7	5.8972	9.3463	5.0283	36.5886	14.2151
gen 8	1.1456	5.1898	5.7686	36.5886	12.1731
gen 9	1.1456	5.1898	5.0283	18.9065	7.5675
gen 10	1.0931	5.1898	2.2959	15.6674	6.0616
gen 11	1.0931	1.9941	1.3136	13.8751	4.5690
gen 12	1.0931	1.9941	1.2724	10.4291	3.6972
gen 13	0.3299	1.9941	0.3690	5.4924	2.0463
gen 14	0.3299	1.9915	0.3690	2.3490	1.2598
gen 15	0.3299	1.9915	0.2524	2.3490	1.2307
gen 16	0.3299	1.6564	0.3690	1.0765	0.8579
gen 17	0.3299	1.1067	0.3690	1.0765	0.7205

Figure 52: Média dos *fitnesses* da primeira à décima sétima geração em todos os experimentos na segunda rodada

gen 18	0.3252	1.1067	0.3690	1.0765	0.7193
gen 19	0.3252	1.1067	0.2801	0.2163	0.4821
gen 20	0.3252	1.1067	0.2801	0.2163	0.4821
gen 21	0.3252	1.1067	0.2801	0.2163	0.4821
gen 22	0.3252	1.1067	0.2801	0.1868	0.4747
gen 23	0.3252	1.1067	0.2801	0.1868	0.4747
gen 24	0.3252	1.1067	0.2801	0.1868	0.4747
gen 25	0.1139	1.1067	0.2801	0.1868	0.4219
gen 26	0.1139	1.1067	0.2801	0.1868	0.4219
gen 27	0.1139	1.0485	0.2801	0.1868	0.4073
gen 28	0.1139	0.1207	0.2093	0.1868	0.1577
gen 29	0.1139	0.1207	0.2093	0.1868	0.1577
gen 30	0.1139	0.0625	0.1660	0.1868	0.1323
gen 31	0.1139	0.0625	0.1283	0.1868	0.1229
gen 32	0.1139	0.0625	0.1283	0.1324	0.1093
gen 33	0.1139	0.0116	0.1283	0.1324	0.0966
gen 34	0.1139	0.0116	0.1283	0.1324	0.0966
gen 35	0.1139	0.0116	0.1283	0.1324	0.0966
gen 36	0.0019	0.0116	0.1283	0.1324	0.0685

Figure 53: Média dos *fitnesses* da décima oitava à trigésima sexta geração em todos os experimentos na segunda rodada

gen 37	0.0019	0.0116	0.1283	0.1324	0.0685
gen 38	0.0019	0.0116	0.1283	0.1324	0.0685
gen 39	0.0019	0.0116	0.1283	0.1324	0.0685
gen 40	0.0019	0.0116	0.1283	0.1324	0.0685

Figure 54: Média dos fitnesses das últimas quatro gerações em todos os experimentos na segunda rodada

Melhores soluções em cada experimento:

Para o experimento 1 (figura 55), a melhor solução foi 0.0019, representando uma melhora substancial em relação à primeira rodada. Este valor sugere que o AG conseguiu se aproximar bastante do ótimo global da função Rastrigin. A eficiência em localizar soluções com *fitness* muito baixo demonstra um bom equilíbrio entre exploração e explotação (Goldberg,1989 [2]) ao longo das gerações.

No experimento 2 (figura 56), a melhor solução foi 0.0116 e representa uma solução de alta qualidade. Houve uma leve melhora com relação à rodada 1.

No experimento 3 (figura 57), a melhor solução foi 0.1283. Este experimento apresentou a terceira melhor solução entre os quatro, mas ainda com uma performance bastante próxima do ótimo. As diferenças podem estar associadas à forma como o AG explorou o espaço de busca neste caso específico.

No experimento 4 (figura 58), a melhor solução foi 0.1324. Este valor, embora um pouco mais alto que os demais, ainda mostra uma boa eficiência do AG, pois permanece significativamente próximo ao ótimo global esperado.

As melhores soluções encontradas indicam que o AG está convergindo consistentemente para valores muito próximos do ótimo global da função de Rastrigin. Isso evidencia que a parametrização aplicada (taxas de *crossover* e mutação, tamanho da população, e número de gerações) está adequada para esta função de *benchmark*.

A pequena variação entre os experimentos é esperada devido à natureza estocástica dos AGs e, pelo menos neste caso específico, não compromete a confiabilidade geral do método. A tendência é que, ao longo de mais rodadas, os valores continuem se estabilizando com menor variabilidade entre experimentos.

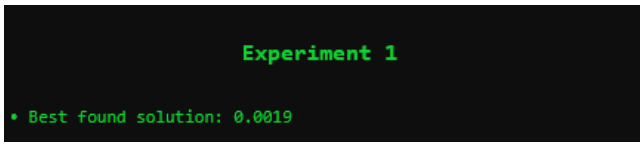


Figure 55: Melhor solução encontrada no primeiro experimento da primeira rodada

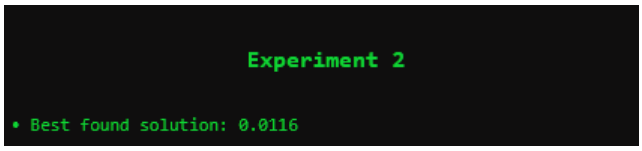


Figure 56: Melhor solução encontrada no segundo experimento da segunda rodada

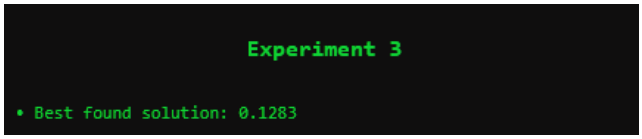


Figure 57: Melhor solução encontrada no terceiro experimento da segunda rodada

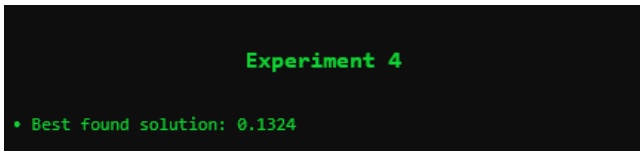


Figure 58: Melhor solução encontrada no quarto experimento da segunda rodada

- **Rodada 3**:

Tempo de Execução: Similarmente às rodadas passadas, obteve-se um tempo de execução de 21.73 segundos para a terceira rodada, um valor consistente com os resultados anteriores. Isso indica estabilidade na configuração do algoritmo genético quanto ao desempenho computacional e também reforça que as condições de execução, como parâmetros de *crossover* e mutação, bem como o tamanho da população, não geraram grandes variações na eficiência temporal entre as rodadas.

Best Fitness Per Generation & Experiments (Run 3)

Number of Experiments:	4
Number of Generations:	40
Population Size:	100
Crossover Rate:	65%
Mutation Rate:	0.8%
Intent:	Minimize
Interval:	[-100,100]
Crossover Type:	One Point
Normalize Linear:	No
Elitism:	No
Steady State:	No
Steady State Without Duplicates:	No
Gap:	No
Execution Time:	21.73 seconds

Figure 59: Tempo de execução da terceira rodada - Otimização da função de Rastrigin

Box-Plot:

A análise do box-plot para a terceira rodada do cenário base revela padrões similares aos das rodadas anteriores, consolidando a estabilidade do algoritmo genético nas configurações utilizadas.

Nas gerações iniciais, especialmente nas três primeiras, há uma ampla variação nos valores de *fitness*, como indicado pelos boxplots altos e *outliers* (Hawkings, 1980 [72]) visíveis acima dos limites superiores, como observa-se na figura 60. Isso reflete uma fase inicial de exploração, onde o algoritmo ainda está buscando soluções em diferentes partes do espaço de busca.

A partir da geração 5, observa-se uma redução significatiava na dispersão, com os boxplots se tomando cada vez mais compactos. Isso sugere uma transição para a fase de exploração mais local e intensiva.

Após a geração 10, os valores de *fitness* convergem para valores baixos, com as medianas praticamente se sobrepondo aos limites inferiores. Isso indica que a maioria das soluções já se encontra em uma região bem próxima ao ótimo local.

Da geração 20 em diante, o boxplot praticamente desaparece, sugerindo que o *fitness* da população atingiu uma homogeneidade muito alta, consolidando o resultado.

Em relação às rodadas anteriores, o comportamento do box-plot é muito similar. No entanto, a redução da dispersão ocorre de maneira ligeiramente mais rápida nesta rodada, o que pode ser um reflexo de variações naturais no comportamento do AG ou diferenças sutis na inicialização.

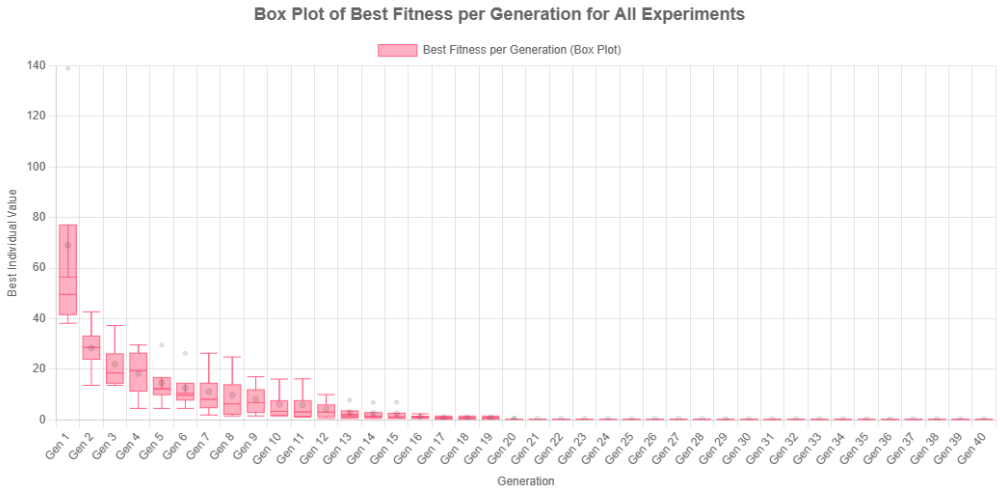


Figure 60: Box-Plot da terceira rodada - Otimização da função de Rastrigin

Melhor *Fitness* por Geração em Cada Experimento:

Os gráficos de melhor *fitness* por geração em cada experimento para a rodada 3 do cenário base também apresentam tendências similares aos das rodadas anteriores.

Nos quatro experimentos (figuras 61, 62, 63 e 64), observa-se uma rápida convergência nas primeiras gerações, indicando que o algoritmo encontrou soluções significativamente melhores de maneira eficiente logo no início do processo. Os valores de *fitness* diminuem drasticamente nas primeiras 10 gerações.

Após, aproximadamente, a décima quinta geração, o *fitness* se estabiliza, com melhorias marginais ou inexistentes, sugerindo que o algoritmo atingiu um ponto de convergência local ou global para os parâmetros configurados.

As trajetórias individuais mostram ligeiras diferenças no desempenho inicial e final, mas todas convergem para valores próximos a zero, corroborando com a consistência do algoritmo em diferentes experimentos. Comparados com os gráficos das rodadas anteriores, os padrões gerais permanecem praticamente idênticos.

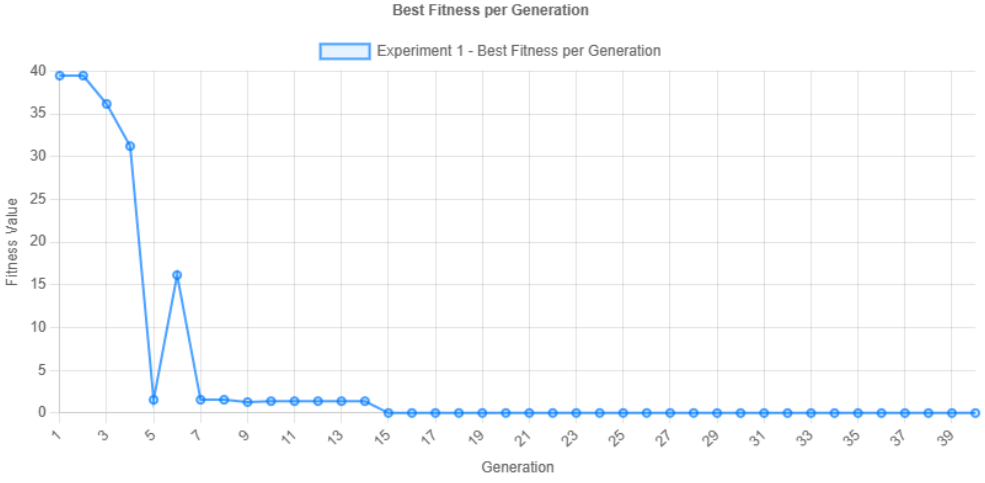


Figure 61: Gráfico de melhor *fitness* por geração no primeiro experimento da terceira rodada.

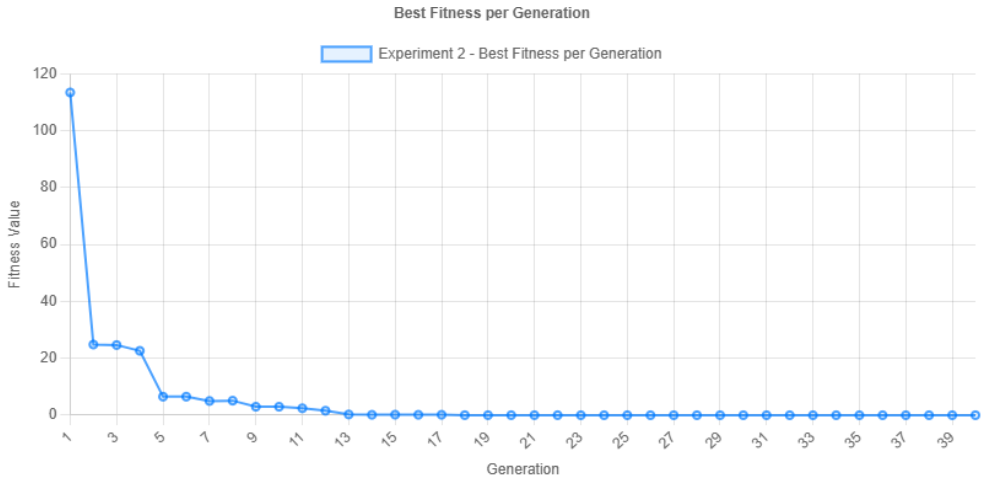


Figure 62: Gráfico de melhor *fitness* por geração no segundo experimento da segunda rodada.

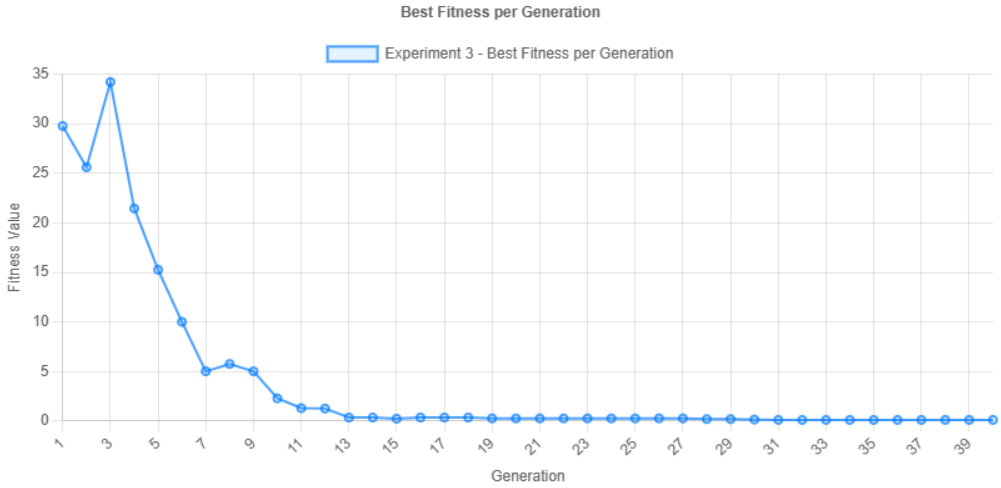


Figure 63: Gráfico de melhor *fitness* por geração no terceiro experimento da terceira rodada.

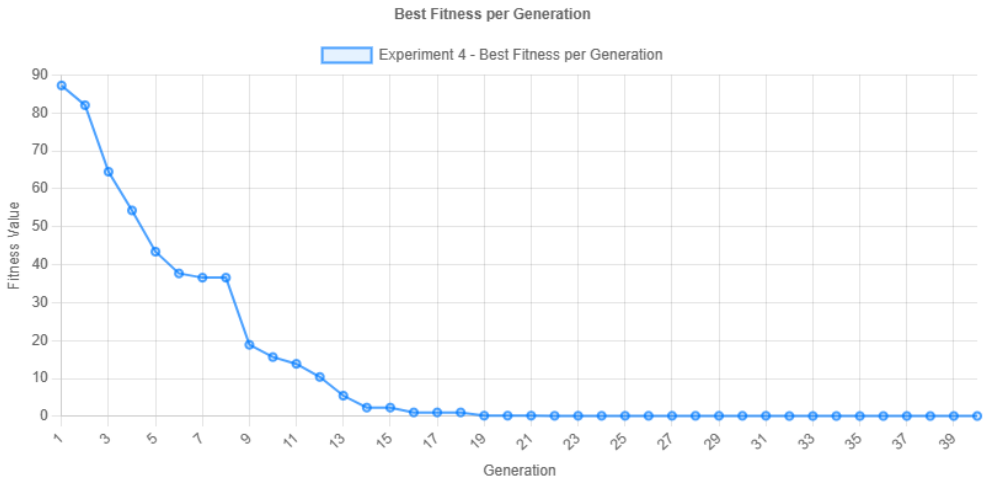


Figure 64: Gráfico de melhor *fitness* por geração no quarto experimento da terceira rodada.

Gráfico de Barras da última geração de cada experimento:

A distribuição dos valores de *fitness* nos indivíduos ainda apresenta variabilidade, com alguns indivíduos exibino valores de *fitness* significativamente mais elevados do que

outros. Esse padrão sugere que, embora o algoritmo tenha se aproximado de soluções ótimas, alguns indivíduos ainda podem ser afetados por mutações ou *crossover* menos eficazes.

A similaridade dos gráficos com as rodadas anteriores sugere uma consistência no comportamento do algoritmo genético para as configurações usadas. Essa consistência é positiva, pois indica estabilidade no desempenho do AG mesmo com múltiplas execuções.

Os indivíduos com menor valor de *fitness* tendem a predominar, o que é um indicativo de que o AG está convergindo efetivamente para soluções próximas ao ótimo. Essa predominância mostra que o algoritmo continua a ser bem sucedido em minimizar a função de *fitness*.

A presença de indivíduos com *fitness* mais elevados (*outliers* (Hawkings, 1980 [72])) pode indicar que, mesmo na última geração, ainda há alguma exploração residual (Kutner, 2004 [74]) ocorrendo. Esse equilíbrio entre exploração e explotação (Goldberg,1989 [2]) continua a ser fundamental para evitar uma convergência prematura.

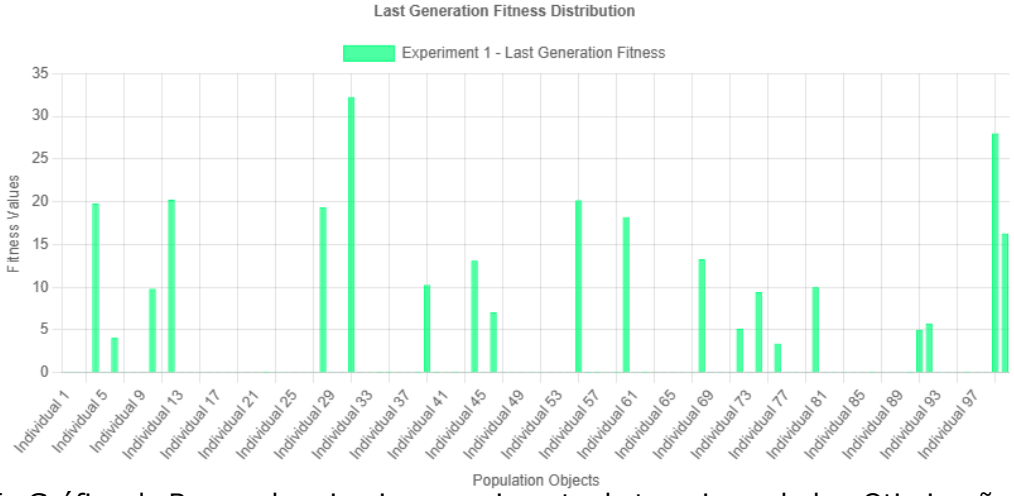


Figure 65: Gráfico de Barras do primeiro experimento da terceira rodada - Otimização da função de Rastrigin

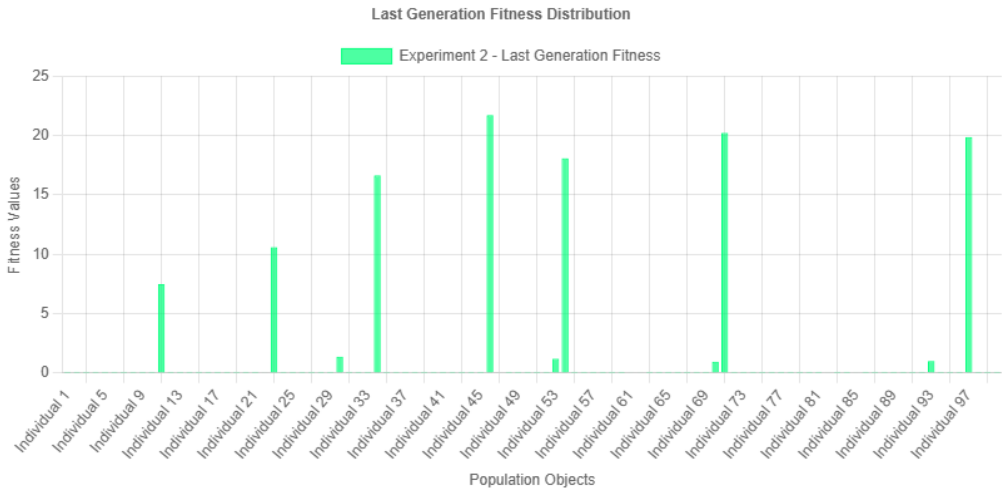


Figure 66: Gráfico de Barras do segundo experimento da terceira rodada - Otimização da função de Rastrigin

Figure 67: Gráfico de Barras do terceiro experimento da terceira rodada - Otimização da função de Rastrigin

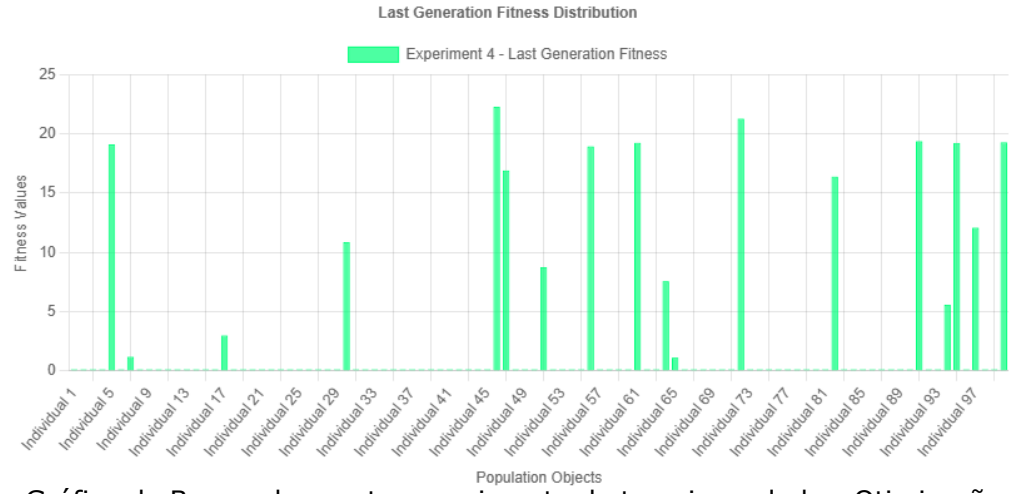


Figure 68: Gráfico de Barras do quarto experimento da terceira rodada - Otimização da função de Rastrigin

Performance do AG ao longo de 40 gerações:

*** Média dos *fitnesses* com o passar das gerações ***

A análise da média dos valores de *fitness* ao longo das gerações na terceira rodada do cenário base confirma a tendência de melhoria progressiva observada nas rodadas anteriores. As médias dos *fitnesses* diminuem consistentemente ao longo das gerações, indicando a capacidade do algoritmo em convergir para soluções otimizadas com o passar do tempo.

Gerações Iniciais (1 a 5): A média dos *fitnesses* é elevada, refletindo a aleatoriedade inicial da população. Isso é esperado, já que o algoritmo ainda está explorando amplamente o espaço de busca. Houve uma queda significativa nas primeiras gerações, o que demonstra que o algoritmo rapidamente elimina soluções menos eficientes.

Transição (6 a 15): Na figura 69, pode-se verifcar o ritmo de redução dos *fitnesses* torna-se mais lento à medida que o algoritmo refina as soluções. Essa desaceleração é característica da transição entre exploração (foco na diversidade de soluções) e explotação (foco no refinamento das melhores soluções) (Goldberg, 1989) [2].

Estabilização (16 em diante): Após a geração 16, figuras 69, 70 e 71, observa-se que as médias dos *fitnesses* atingem valores muito baixos, próximos do ponto de convergência. Os valores permanecem praticamente constantes até a geração 40, sugerindo

que o algoritmo encontrou soluções próximas ao ótimo global. A média final de *fitness* (0.0846) é consistente com a qualidade das soluções obtidas nos experimentos.

Os resultados desta rodada são altamente similares aos das rodadas anteriores, indicando a consistência das configurações do algoritmo genético no cenário base. A média final de *fitness* é ligeiramente inferior, sugerindo que houve uma pequena melhoria na capacidade de explotação. O padrão observado é característico de um bom equilíbrio entre exploração e explotação (Goldberg, 1989 [2]) no algoritmo genético. Apesar de variações nas gerações iniciais, o algoritmo demonstra uma convergência eficiente para soluções otimizadas.

Generations	Experiments				Average
	1º	2º	3º	4º	
gen 1	42.7139	139.0231	56.4517	38.1737	69.0906
gen 2	42.7139	29.9327	27.3506	13.6260	28.4058
gen 3	37.2885	14.7333	22.3439	13.6260	21.9979
gen 4	29.5972	25.2679	4.5102	13.6260	18.2503
gen 5	29.5972	11.7336	4.5102	12.4187	14.5649
gen 6	26.2981	10.5065	4.5102	8.9091	12.5560
gen 7	26.3376	10.5065	1.8863	5.7187	11.1123
gen 8	24.8682	10.1914	1.5110	2.4509	9.7554
gen 9	17.0341	10.1914	1.5110	3.3680	8.0261
gen 10	16.0951	4.7239	1.5110	1.9469	6.0692
gen 11	16.2006	4.7239	1.5110	1.0568	5.8731
gen 12	9.9766	4.5073	1.5110	0.5529	4.1369
gen 13	7.7846	2.1047	1.1441	0.5529	2.8966
gen 14	6.8790	1.4856	1.1441	0.5529	2.5154
gen 15	6.9920	0.7487	1.1441	0.5529	2.3594
gen 16	2.3814	0.7487	1.0805	0.5529	1.1909
gen 17	1.4758	0.2912	0.8693	0.5529	0.7973

Figure 69: Média dos *fitnesses* da primeira à décima sétima geração em todos os experimentos na terceira rodada

gen 18	1.4758	0.1503	0.7895	0.5529	0.7421
gen 19	1.4758	0.1503	0.7895	0.5529	0.7421
gen 20	0.1949	0.1503	0.5147	0.1705	0.2576
gen 21	0.1949	0.1183	0.1499	0.1705	0.1584
gen 22	0.1079	0.1183	0.1499	0.1705	0.1367
gen 23	0.1079	0.1183	0.1499	0.1705	0.1367
gen 24	0.1079	0.1183	0.1499	0.1705	0.1367
gen 25	0.1079	0.0957	0.1499	0.1226	0.1190
gen 26	0.1079	0.0957	0.1499	0.1226	0.1190
gen 27	0.1079	0.0957	0.1499	0.1226	0.1190
gen 28	0.1079	0.0957	0.1499	0.1226	0.1190
gen 29	0.0085	0.0573	0.1499	0.1226	0.0846
gen 30	0.0085	0.0799	0.1499	0.1226	0.0902
gen 31	0.0085	0.0573	0.1499	0.1226	0.0846
gen 32	0.0085	0.0573	0.1499	0.1226	0.0846
gen 33	0.0085	0.0573	0.1499	0.1226	0.0846
gen 34	0.0085	0.0573	0.1499	0.1226	0.0846
gen 35	0.0085	0.0573	0.1499	0.1226	0.0846
gen 36	0.0085	0.0573	0.1499	0.1226	0.0846

Figure 70: Média dos *fitnesses* da décima oitava à trigésima sexta geração em todos os experimentos na terceira rodada

gen 37	0.0085	0.0573	0.1499	0.1226	0.0846
gen 38	0.0085	0.0573	0.1499	0.1226	0.0846
gen 39	0.0085	0.0573	0.1499	0.1226	0.0846
gen 40	0.0085	0.0573	0.1499	0.1226	0.0846

Figure 71: Média dos *fitnesses* das últimas quatro gerações em todos os experimentos na terceira rodada

*** Melhores soluções em cada experimento ***:

As melhores soluções encontradas em cada experimento da terceira rodada no cenário base também demonstram uma consistência com as rodadas anteriores, destacando-se pelo refinamento progressivo dos valores de *fitness* ao longo das execuções.

Experimento 1: A melhor solução encontrada foi 0.0085, indicando que o algoritmo genético foi capaz de convergir para uma solução de alta qualidade, confirmando sua capacidade de exploração inicial eficiente e explotação final precisa.

Experimento 2: Com uma melhor solução de 0.0573, o experimento apresentou resultados próximos ao ótimo, mas ligeiramente acima do encontrado no Experimento 1, sugerindo variações nos parâmetros iniciais ou efeitos estocásticos que podem ter influenciado a convergência.

Experimento 3: O valor de 0.1499 reflete uma performance aceitável, embora não tão próxima do ótimo comparado aos experimentos anteriores. Este desvio pode estar relacionado à seleção dos indivíduos ou ao impacto do *crossover* e mutação.

Experimento 4: A melhor solução foi 0.1226, representando uma ligeira melhora em relação ao Experimento 3, mas ainda superior às soluções ótimas dos primeiros experimentos.

Essas variações entre os melhores *fitnesses* nos diferentes experimentos reforçam a importância do equilíbrio entre exploração e explotação (Goldberg, 1989 [2]) no AG. Apesar das diferenças nos valores finais, todos os experimentos convergiram para soluções com alta qualidade, demonstrando que o algoritmo é robusto, mesmo com flutuações naturais em execuções repetidas. Esses resultados reafirmam que os parâmetros configurados são adequados para o problema otimizado, mas também sugerem possíveis ajustes finos para reduzir ainda mais a dispersão entre as soluções finais.

Figure 72: Melhor solução encontrada no primeiro experimento da terceira rodada

Figure 73: Melhor solução encontrada no segundo experimento da terceira rodada

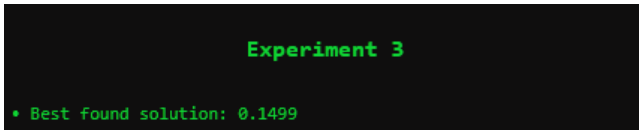


Figure 74: Melhor solução encontrada no terceiro experimento da terceira rodada

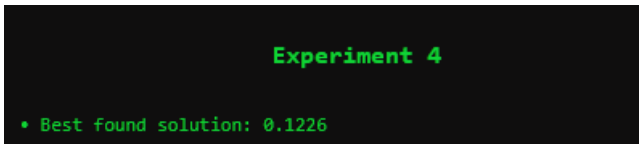


Figure 75: Melhor solução encontrada no quarto experimento da terceira rodada

Média do melhor *Fitness* por geração: 0.0846

Rodadas para o Cenário Experimental

Para este cenário, foi considerado o GA1 (ver tabela 7) modificado, fixando a taxa de *crossover* em 10% e a taxa de mutação em 80.8%.

No cenário base, a taxa de *crossover* era de 65%. Alta taxa de *crossover* promove a exploração do espaço de busca, permitindo uma maior troca de informações entre indivíduos. Isso geralmente acelera a convergência para uma solução ótima. Neste cenário experimental, a redução drástica na taxa de *crossover* limita a exploração do espaço de busca. Isso pode causar uma menor diversificação da população, potencialmente resultando em convergência prematura para ótimos locais.

Já uma baixa taxa de mutação mantém a estabilidade da população, promovendo a explotação (Goldberg, 1989 [2]) das soluções já encontradas. O aumento expressivo da taxa de mutação (de 0.8% para 80.8%) no cenário experimental promove a exploração, introduzindo maior variabilidade genética. Isso pode corrigir problemas de convergência prematura, mas também pode dificultar a estabilização do algoritmo em uma solução ótima.

O baixo *crossover* combinado com uma mutação alta pode atrasar a convergência para o ótimo global, já que as soluções geradas pelo *crossover* são menos exploradas e as mutações constantes podem impedir a estabilização. Por outro lado, o aumento da mutação deve manter a diversidade populacional mais alta, reduzindo o risco de ótimos locais, mas aumentando a variabilidade nos resultados entre as rodadas. A alta taxa de mutação pode também gerar dificuldade em refinar as melhores soluções, já que a mutação tende a alterar mesmo indivíduos

bem adaptados.

1. **Resumo Geral de Todas as Rodadas**:

   O cenário experimental apresentou uma redução rápida nos valores de *fitness* nas primeiras gerações, como pode-se observar na figura 76, mas essa queda é menos agressiva em relação ao Cenário Base.

   A taxa elevada de mutação parece ter mantido uma maior diversidade inicial, o que é esperado, já que a mutação introduz variabilidade e evita que o algoritmo convirja prematuramente para ótimos locais. No entanto, essa maior diversidade inicial também pode ter retardado um pouco o progresso inicial, dado que o *fitness* médio permanece mais alto nas primeiras gerações.

   A redução nos valores de *fitness* continua a um ritmo moderado, indicando que o algoritmo começa a se concentrar em regiões promissoras, mas ainda há uma exploração significativa devido à alta mutação. Isso sugere que, apesar de estar convergindo, o processo de explotação é mais lento, possivelmente por causa da menor taxa de *crossover*, que dificulta o refinamento das soluções.

   Os valores médios de *fitness* se estabilizam em torno de um nível mais alto do que no Cenário Base. Isso reflete a dificuldade do algoritmo em refinar ainda mais as soluções devido à menor contribuição do *crossover* no refinamento final. A alta mutação continua introduzindo diversidade, mas, nesta fase, pode estar prejudicando o ajuste fino das soluções.

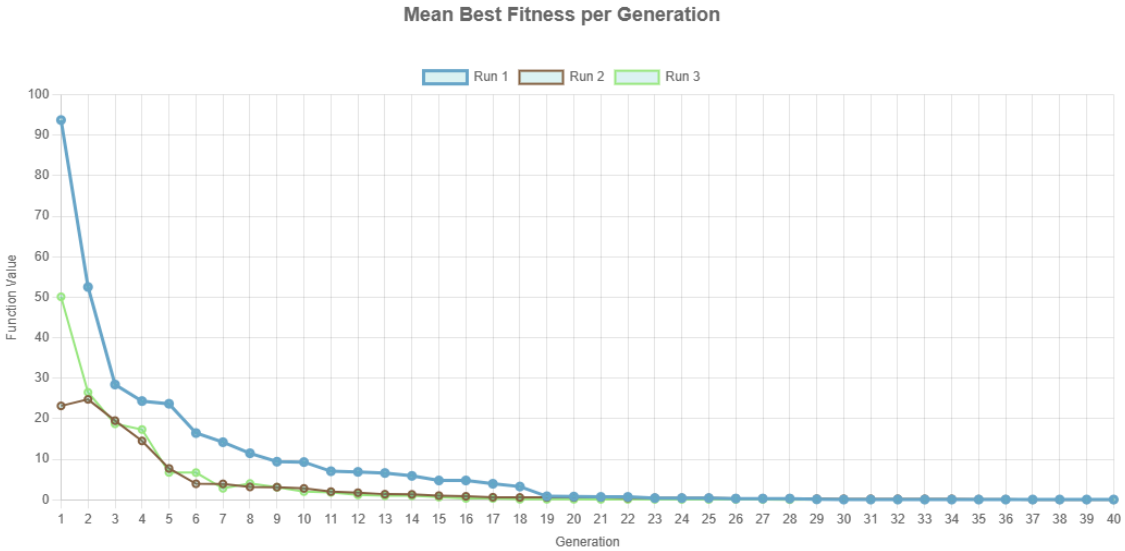


Figure 76: Gráfico consolidado de todas as rodadas para o cenário experimental para ponderação de taxas de *crossover* e mutação - média de melhor *fitness* por geração em cada experimento

A análise dos tempos de execução para o cenário experimental (com alta taxa de mutação de 80,8% e baixa taxa de *crossover* de 10%) em comparação ao cenário base (mutação 0,8% e *crossover* 65%) sugere que o aumento da taxa de mutação e a redução do *crossover* tiveram um impacto leve, mas consistente, no tempo de execução. Os tempos de execução médios nas três rodadas do cenário base foram consistentemente em torno de 21,7 segundos. Os tempos de execução para a rodada experimental ficou com uma média levemente superior, em torno de 24 segundos, como é possível visualizar nas imagens 77a, 77b e **??**.

Best Fitness Per Generation & Experiments (Run 1)	
Number of Experiments:	4
Number of Generations:	40
Population Size:	100
Crossover Rate:	10%
Mutation Rate:	80.8%
Intent:	Minimize
Interval:	[-100,100]
Crossover Type:	One Point
Normalize Linear:	No
Elitism:	No
Steady State:	No
Steady State Without Duplicates:	No
Gap:	No
Execution Time:	23.99 seconds

(a) Tempo de execução da Primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

Best Fitness Per Generation & Experiments (Run 2)	
Number of Experiments:	4
Number of Generations:	40
Population Size:	100
Crossover Rate:	10%
Mutation Rate:	80.8%
Intent:	Minimize
Interval:	[-100,100]
Crossover Type:	One Point
Normalize Linear:	No
Elitism:	No
Steady State:	No
Steady State Without Duplicates:	No
Gap:	No
Execution Time:	23.70 seconds

(b) Tempo de execução da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

Best Fitness Per Generation & Experiments (Run 3)	
Number of Experiments:	4
Number of Generations:	40
Population Size:	100
Crossover Rate:	10%
Mutation Rate:	80.8%
Intent:	Minimize
Interval:	[-100,100]
Crossover Type:	One Point
Normalize Linear:	No
Elitism:	No
Steady State:	No
Steady State Without Duplicates:	No
Gap:	No
Execution Time:	24.78 seconds

(c) Tempo de execução da terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

Figure 77: Parâmetros utilizados nas rodadas para o cenário experimental, com o tempo de execução em destaque.

O aumento do tempo de execução pode ser explicado pela maior taxa de mutação (80,8%). A mutação exige alterações adicionais em uma fração maior da população a cada geração, aumentando ligeiramente o custo computacional por iteração. Como a população tem 100 indivíduos e o número de gerações é 40, o impacto da mutação se acumula ao longo das execuções.

Por outro lado, a redução da taxa de *crossover* para 10% reduz a quantidade de combinações entre indivíduos, o que poderia, em tese, equilibrar o aumento do custo computacional da mutação. No entanto, parece que o impacto da mutação predomina, resultando em um tempo de execução um pouco maior.

Embora essa diferença seja pequena, ela sugere que os operadores genéticos, particularmente a mutação, tem um impacto direto na eficiência do algoritmo. Essa análise inicial deve ser complementada pela avaliação da qualidade das soluções encontradas e do comportamento da convergência para determinar se o aumento do tempo de execução é justificado pelos ganhos em desempenho ou diversidade.

2. **Detalhamento Por Rodada**:
    - **Rodada 1**:

Box-Plot: Aqui, observa-se maior dispersão inicial, especialmente nas primeiras gerações. Isso é esperado devido à alta taxa de mutação, que introduz mais diversidade na população, ampliando o intervalo de valores *fitness* na geração inicial. A redução dos valores *fitness* acontece de forma mais gradual. A dispersão inicial elevada é ajustada ao longo das gerações, refletindo o impacto da exploração promovida pela alta mutação. Há a presença de mais *outliers* (Hawkings, 1980 [72]) nas primeiras gerações, que vão se tornando menos frequentes nas gerações finais, quando compara-se com o box-plot da rodada 1 do cenário base (imagem 26). Isso indica que a alta mutação cria soluções extremas (tanto boas quanto ruins) no início, enquanto o processo evolutivo gradualmente refina as soluções. A mediana dos valores *fitness* apresenta um decréscimo mais lento em comparação ao cenário base na primeira rodada, refletindo uma convergência mais ampla e com maior dependência das gerações posteriores para atingir melhores soluções.

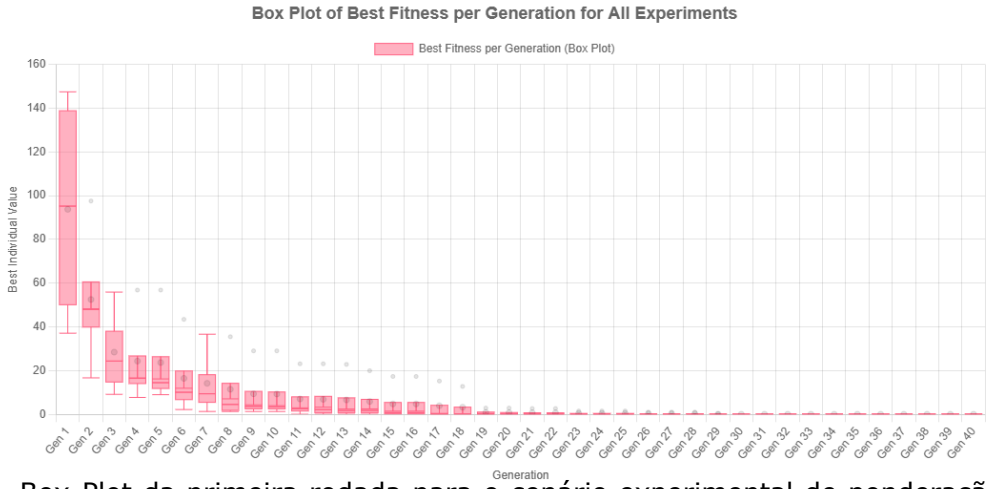


Figure 78: Box-Plot da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

Melhor Fitness por Geração em Cada Experimento:

No cenário experimental, observa-se maior variabilidade inicial, mas uma convergência mais uniforme entre os experimentos em comparação com o cenário base. A taxa de mutação elevada (80.8%) no cenário experimental provoca maior variação nos valores de *fitness* nas primeiras gerações, conforme visto, especialmente no Experimento 3 (figura 81). Apesar disso, a convergência final para valores baixos de *fitness* é mais consistente entre os experimentos no cenário experimental. Isso sugere que a mutação elevada melhora a capacidade de escapar de mínimos locais. A taxa de *crossover* reduzida para 10% no cenário experimental parece limitar a troca de informações entre os indivíduos, o que pode contribuir para maior estabilidade após algumas gerações.

As diferenças são particularmente evidentes ao analisar experimentos como o 3 no cenário experimental (figura 81) e o 2 no cenário base (figura 28) (também da primeira rodada), que demonstram como os parâmetros de mutação e *crossover* afetam tanto a exploração inicial quanto a convergência final.

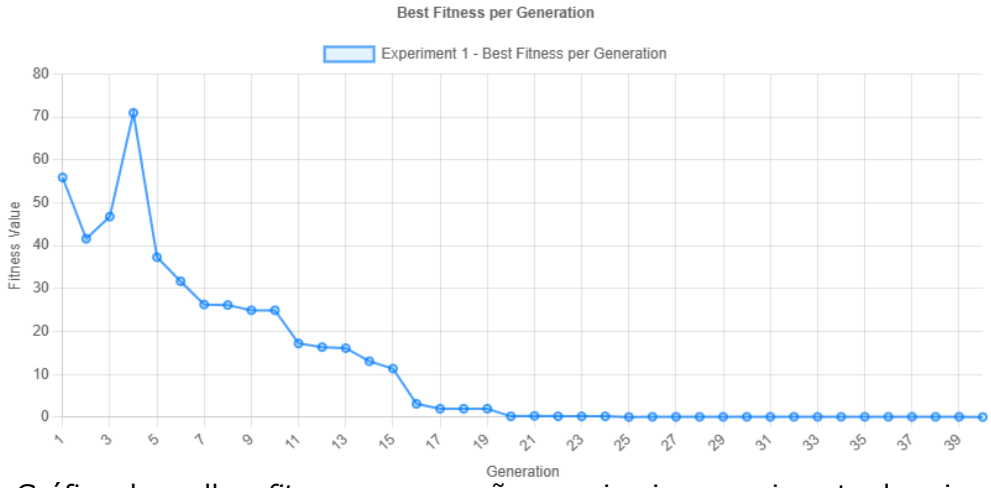


Figure 79: Gráfico de melhor *fitness* por geração no primeiro experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

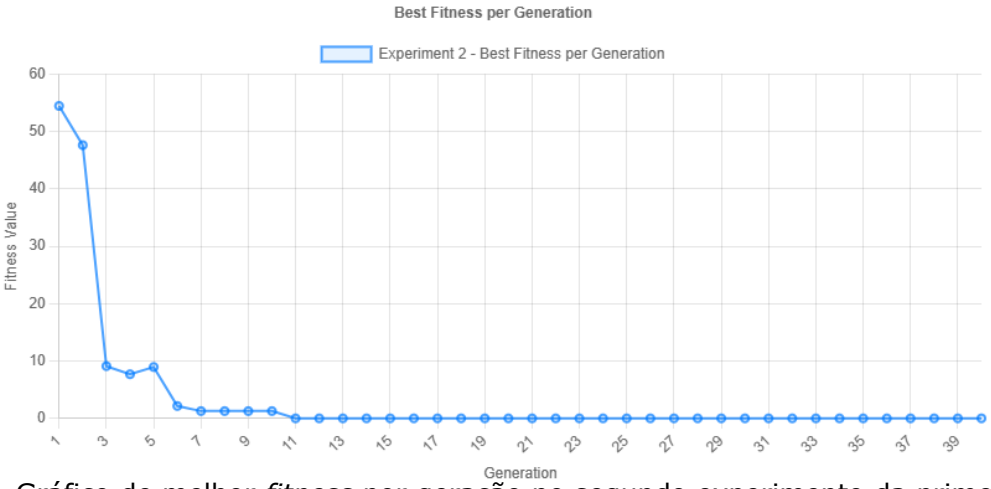


Figure 80: Gráfico de melhor *fitness* por geração no segundo experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

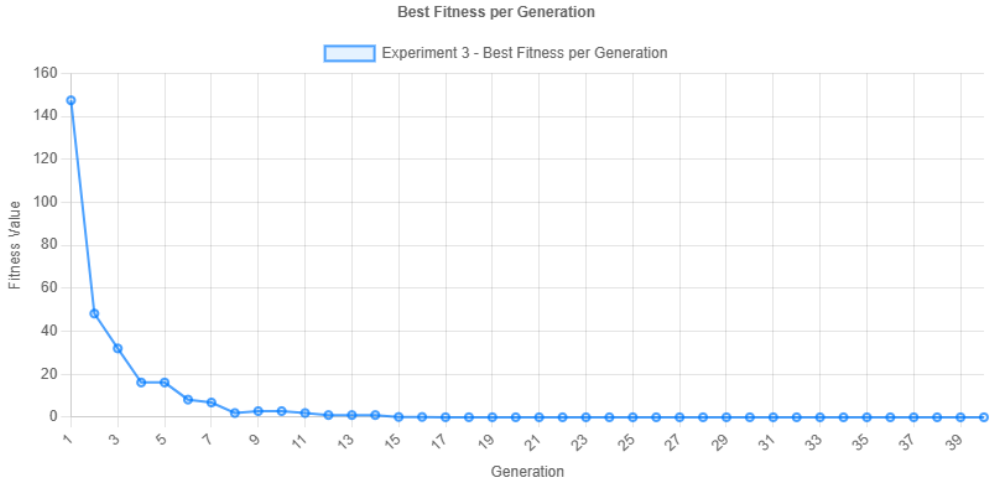


Figure 81: Gráfico de melhor *fitness* por geração no terceiro experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

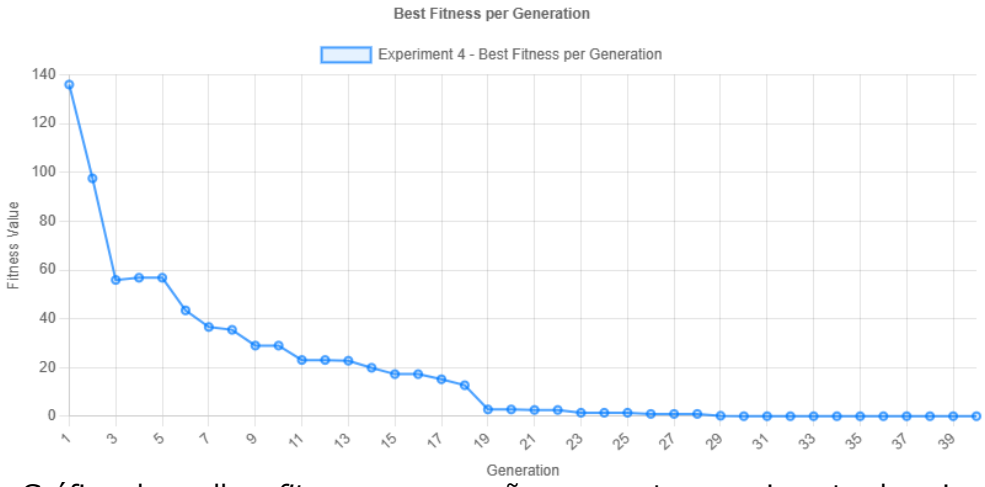


Figure 82: Gráfico de melhor *fitness* por geração no quarto experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

Gráficos de Barras da última geração de cada experimento:

Nos gráficos de evolução do *fitness* por geração, o cenário experimental apresenta, em geral, um declínio mais rápido no valor de *fitness* em algumas rodadas iniciais, estabilizando em valores baixos ao longo das gerações. Ao comparar com o cenário base da primeira rodada, os valores iniciais de *fitness* tendiam a variar mais, com flutuações significativas antes de estabilizar. Isso indica que as taxas ajustadas no cenário experimental contribuíram para maior consistência no processo de convergência.

No cenário experimental, as barras estão mais uniformemente distribuídas, sugerindo que os indivíduos mantiveram maior diversidade nos valores de *fitness* até a última geração.

A alta taxa de mutação no cenário experimental parece ter incentivado maior diversidade genética, permitindo que o algoritmo explorasse melhor o espaço de busca, resultando em uma convergência mais consistente. A menor taxa de *crossover* aparenta ter ajudado a preservar a diversidade e a evitar a convergência prematura para mínimos locais, o que é vantajoso em problemas complexos como a função de Rastrigin. A execução marginalmente mais longa no cenário experimental é compensada por resultados mais consistentes e uma melhor exploração do espaço de soluções.

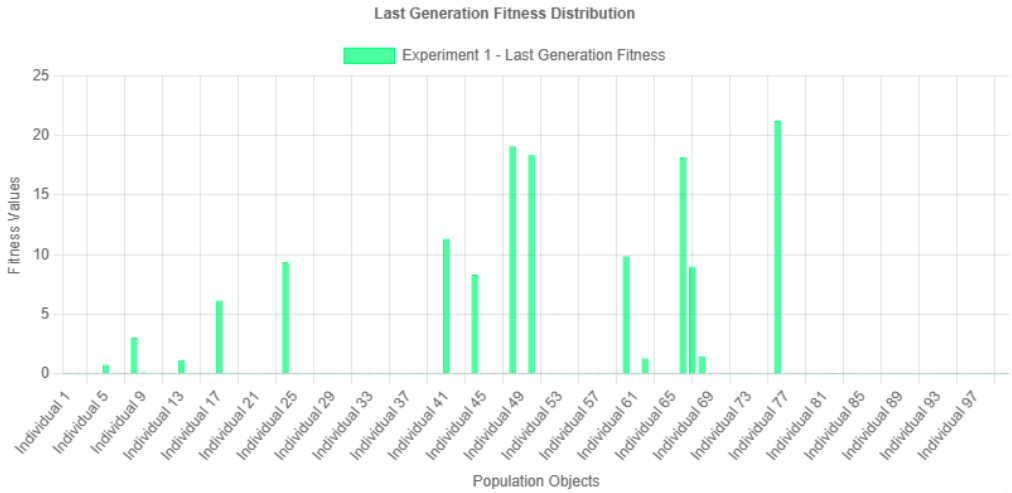


Figure 83: Gráfico de Barras do primeiro experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

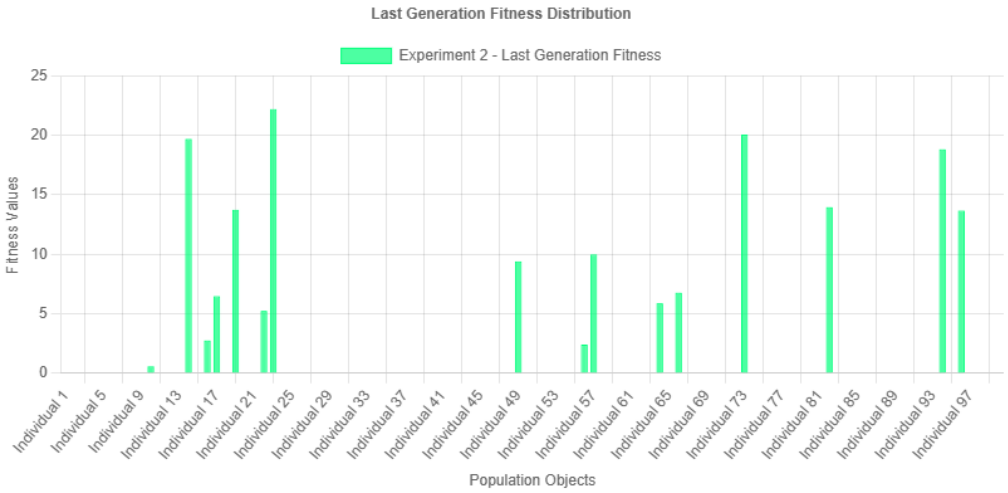


Figure 84: Gráfico de Barras do segundo experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

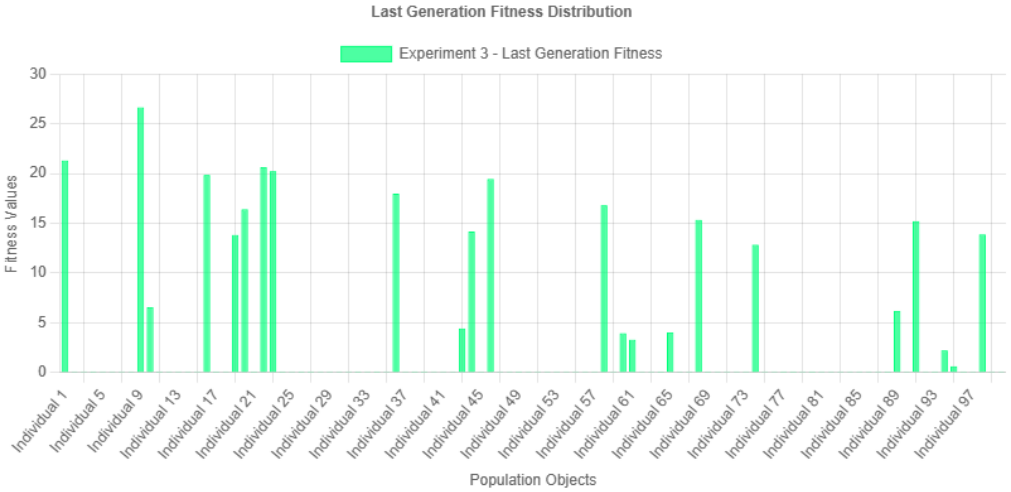


Figure 85: Gráfico de Barras do terceiro experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

Figure 86: Gráfico de Barras do quarto experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

Média do melhor *Fitness* por geração: 0.0310

Performance do AG longo de 40 gerações:

*** Média dos *fitnesses* com o passar das gerações ***

As primeiras gerações apresentam valores significativamente maiores de *fitness* médio, o que indica uma inicialização com maior variação ou soluções menos otimizadas. O decaimento do *fitness* médio ao longo das gerações é evidente, com estabilização por volta da geração 20 (figura 88). A média final se aproxima de valores muito baixos, o que mostra convergência para uma solução mínima, mas com menor eficiência no início em relação à primeira rodada do cenário base.

Há uma maior dispersão nos *fitness* individuais, indicando que a população final ainda mantém certa diversidade, mesmo após 40 gerações. Alguns indivíduos possuem valores consideravelmente mais altos, o que pode sinalizar dificuldade em explorar totalmente a região ótima do espaço de busca.

Os gráficos de melhor *fitness* mostram uma convergência mais lenta nas primeiras gerações, com estabilização após a geração 20. Em algumas rodadas, o melhor *fitness* atinge valores baixos mais rapidamente, mas há variação significativa entre as primeiras dos dois cenários.

O intervalo interquartil (IQR) (Tukey, 1977 [75]) inicial é maior, refletindo maior variabilidade na qualidade das soluções iniciais. A dispersão se reduz ao longo das gerações, mas a convergência completa ocorre mais lentamente.

Então, a combinação de taxa de mutação alta (80.8%) e taxa de *crossover* baixa (10%) no cenário experimental resulta em maior diversidade inicial e uma convergência mais lenta. Isso pode ser útil em cenários onde o espaço de busca é complexo e requer maior exploração, mas é menos eficiente em termos de tempo de execução e qualidade final média.

Generations	Experiments				Average
	1º	2º	3º	4º	
gen 1	37.1105	54.4644	147.4505	135.9427	93.7420
gen 2	16.6818	47.6331	48.2509	97.5354	52.5253
gen 3	16.6818	9.1479	32.0455	55.9063	28.4454
gen 4	16.6818	7.7351	16.1985	56.8211	24.3591
gen 5	12.7810	8.9942	16.1985	56.8211	23.6987
gen 6	11.9483	2.2016	8.2382	43.4422	16.4576
gen 7	11.9483	1.3188	6.8798	36.6479	14.1987
gen 8	7.0895	1.3188	1.9954	35.4903	11.4735
gen 9	4.3412	1.3188	2.8782	29.0274	9.3914
gen 10	3.9635	1.3188	2.8782	29.0274	9.2970
gen 11	2.9754	0.0527	1.9954	23.1179	7.0354
gen 12	3.2889	0.0527	1.0362	23.1179	6.8739
gen 13	2.4432	0.0527	1.0362	22.8379	6.5925
gen 14	2.5012	0.0527	1.0362	19.9446	5.8837
gen 15	1.4683	0.0527	0.1642	17.3294	4.7537
gen 16	1.4683	0.0527	0.1642	17.3294	4.7537
gen 17	0.4331	0.0527	0.0439	15.2237	3.9384

Figure 87: Média dos *fitnesses* da primeira à décima sétima geração em todos os experimentos na primeira rodada para o cenário experimental de ponderação de taxas de crossover e mutação

gen 18	0.1282	0.0527	0.0439	12.7787	3.2509
gen 19	0.4331	0.0527	0.0439	2.8729	0.8507
gen 20	0.1641	0.0527	0.0226	2.8729	0.7781
gen 21	0.0532	0.0527	0.0226	2.6660	0.6986
gen 22	0.0532	0.0527	0.0226	2.6660	0.6986
gen 23	0.0173	0.0527	0.0226	1.5256	0.4045
gen 24	0.0173	0.0527	0.0226	1.5256	0.4045
gen 25	0.0173	0.0527	0.0226	1.5256	0.4045
gen 26	0.0173	0.0527	0.0226	1.0324	0.2812
gen 27	0.0173	0.0527	0.0226	1.0360	0.2821
gen 28	0.0173	0.0527	0.0226	1.0324	0.2812
gen 29	0.0173	0.0527	0.0226	0.2938	0.0966
gen 30	0.0173	0.0527	0.0226	0.1038	0.0491
gen 31	0.0173	0.0527	0.0226	0.1038	0.0491
gen 32	0.0173	0.0527	0.0226	0.1038	0.0491
gen 33	0.0173	0.0527	0.0091	0.1038	0.0457
gen 34	0.0173	0.0527	0.0091	0.1038	0.0457
gen 35	0.0173	0.0527	0.0091	0.1038	0.0457
gen 36	0.0173	0.0527	0.0091	0.1038	0.0457

Figure 88: Média dos *fitnesses* da décima oitava à trigésima sexta geração em todos os experimentos na primeira rodada para o cenário experimental de ponderação de taxas de crossover e mutação

gen 37	0.0173	0.0527	0.0091	0.1038	0.0457
gen 38	0.0110	0.0527	0.0091	0.1038	0.0441
gen 39	0.0110	0.0527	0.0091	0.1038	0.0441
gen 40	0.0110	0.0527	0.0091	0.1038	0.0441

Figure 89: Média dos *fitnesses* das últimas quatro gerações em todos os experimentos na primeira rodada

Melhores soluções em cada experimento:

No cenário experimental, algumas soluções apresentaram valores melhores (menores) em relação ao cenário base, principalmente no *Experiment* 1 (figura 90) e *Experiment* 3 (figura 92). O cenário experimental apresenta maior dispersão de valores das melhores soluções entre os experimentos, com o *Experiment* 4 (figura 93) sendo o menos eficiente comparado aos outros. A alta taxa de mutação no cenário experimental favoreceu explorações mais amplas no espaço de busca, resultando em melhores soluções em alguns experimentos, mas também em maior inconsistência. Portanto, o cenário experimental mostrou ser promissor em explorar soluções mais otimizadas, mas carece de consistência entre os experimentos.

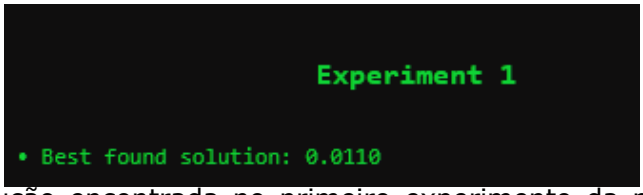


Figure 90: Melhor solução encontrada no primeiro experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

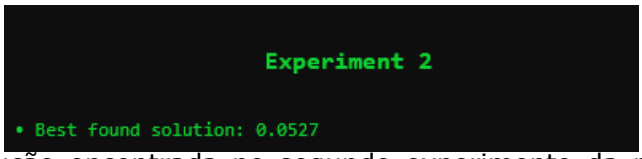


Figure 91: Melhor solução encontrada no segundo experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

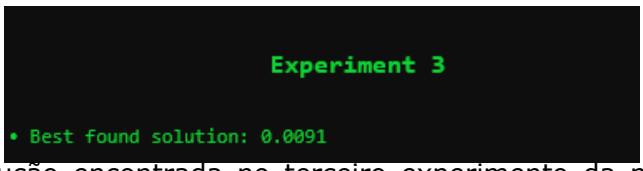


Figure 92: Melhor solução encontrada no terceiro experimento da primeira rodada para o cenário experimental de ponderação de taxas de crossover e mutação

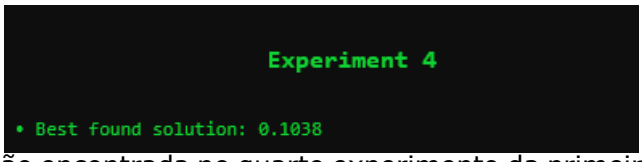


Figure 93: Melhor solução encontrada no quarto experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

- **Rodada 2**:

Box-Plot: Os resultados obtidos na segunda rodada do cenário experimental, representados pelo gráfico de box-plot na figura 94, evidenciam características significativas relacionadas ao comportamento da população em termos de convergência e variabilidade ao longo das gerações. Inicialmente, observa-se uma amplitude interquartil (Tukey, 1977 [75]) relativamente elevada nas gerações iniciais, com valores de *fitness* mostrando maior dispersão. No entanto, esta variabilidade reduz-se rapidamente após a Geração 10, indicando um processo de convergência eficiente para soluções próximas ao mínimo global.

A presença de *outliers* (Hawkings, 1980 [72]) nas primeiras gerações sugere que, no início do processo, a maior taxa de mutação desempenhou um papel crucial na exploração de diferentes regiões do espaço de busca. Este comportamento, embora gere uma dispersão inicial mais ampla, favorece a diversidade populacional e evita o fenômeno de *premature convergence* (Goldberg, 1989 [2]). A rápida estabilização dos valores interquartis nas gerações subsequentes demonstra que o algoritmo foi capaz de ajustar-se e explorar de maneira mais localizada, resultando em soluções mais consistentes.

Em comparação com cenários de menor taxa de mutação, os resultados indicam que o cenário experimental permite um equilíbrio interessante entre exploração e convergência. A baixa dispersão nas gerações finais reflete a eficácia do algoritmo em explorar as regiões mais promissoras do espaço de busca, minimizando os valores de *fitness* e atingindo soluções com maior qualidade.

Portanto, os dados coletados confirmam que a combinação de uma taxa de mutação elevada (80,8%) com uma taxa de *crossover* reduzida (10%) contribuiu para uma convergência acelerada e consistente, mostrando-se uma estratégia eficaz para o problema analisado. Estes resultados sugerem que ajustes na taxa de mutação podem ser utilizados como ferramenta para refinar a performance do algoritmo em aplicações futuras.

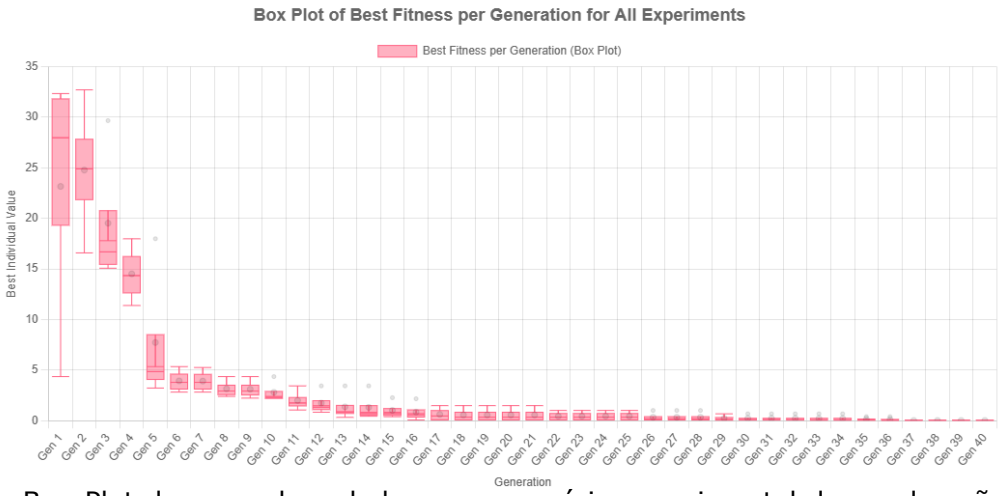


Figure 94: Box-Plot da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

Melhor Fitness por Geração em Cada Experimento:

Nos primeiros 10 a 15 ciclos, todos os experimentos demonstram uma queda acentuada nos valores de *fitness*, convergindo rapidamente para valores próximos de zero. Após a geração 15, os valores de *fitness* se estabilizam, sugerindo que o algoritmo alcançou soluções otimizadas consistentes dentro do intervalo definido para o problema. Os resultados para os quatro experimentos do cenário experimental são relativamente consistentes, indicando um comportamento robusto das configurações do algoritmo em termos de taxa de convergência e precisão. A configuração do cenário experimental (taxa de mutação elevada e *crossover* reduzido) promove uma redução mais homogênea nos valores de *fitness* ao longo das gerações iniciais, sem grandes oscilações.

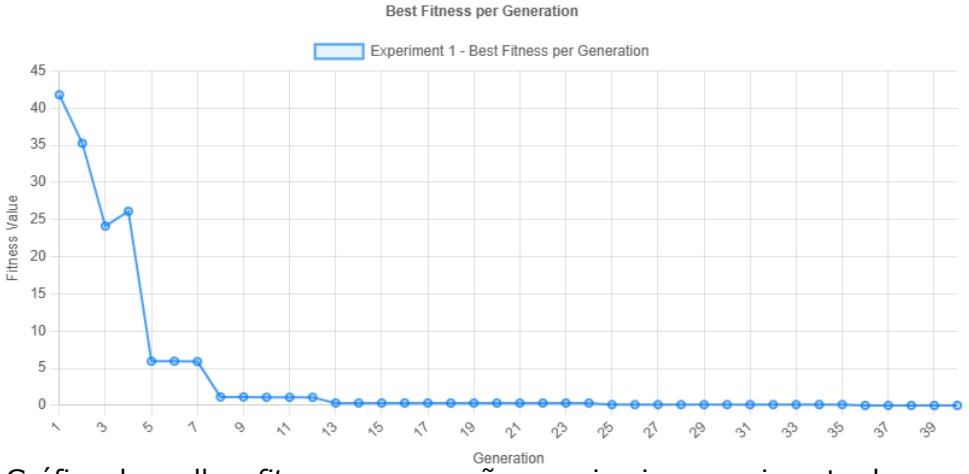


Figure 95: Gráfico de melhor *fitness* por geração no primeiro experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

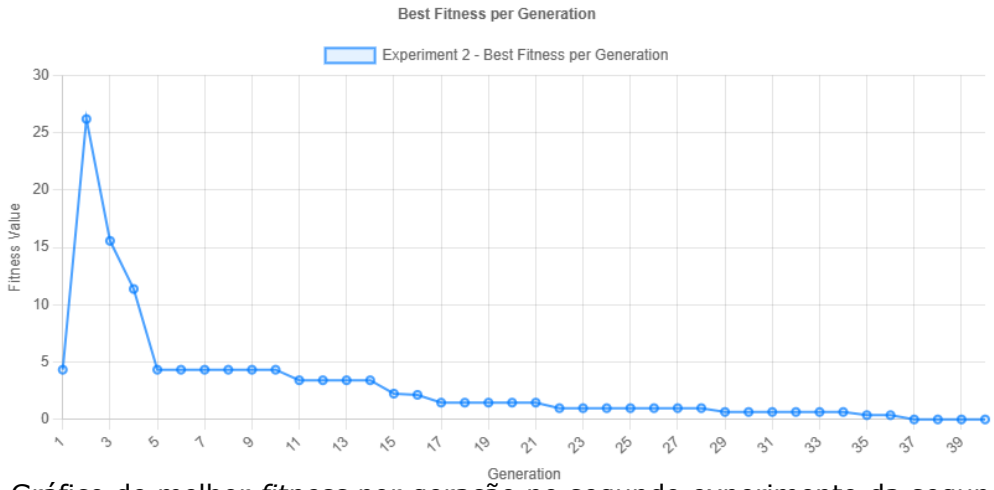


Figure 96: Gráfico de melhor *fitness* por geração no segundo experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

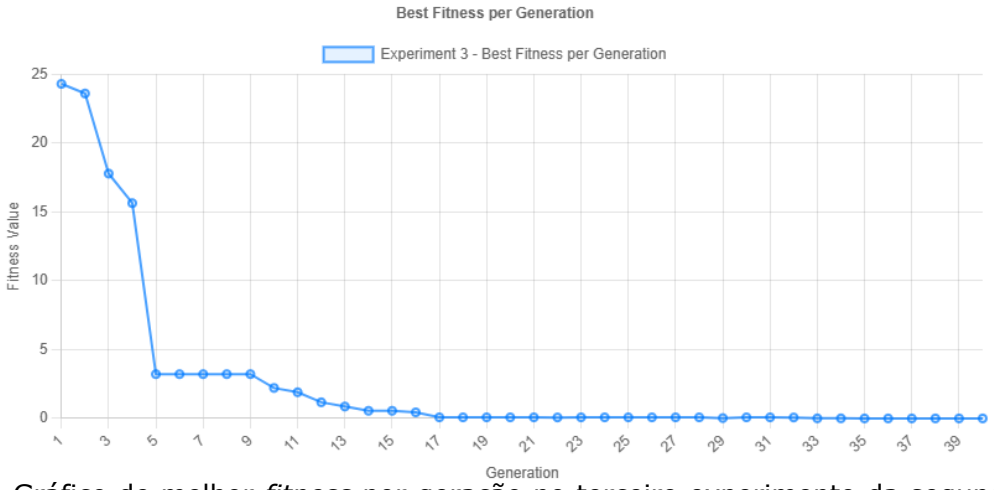


Figure 97: Gráfico de melhor *fitness* por geração no terceiro experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

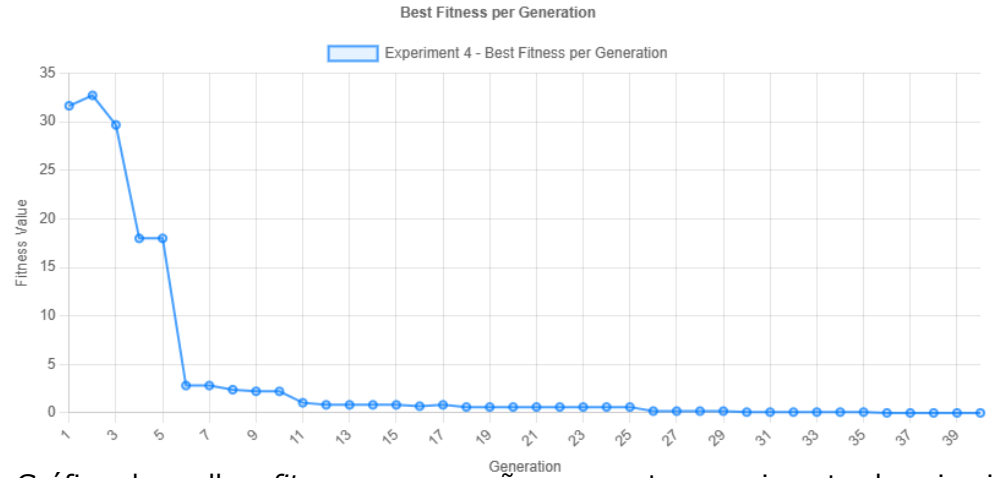


Figure 98: Gráfico de melhor *fitness* por geração no quarto experimento da primeira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

Gráficos de Barras da última geração de cada experimento:

Na análise dos gráficos de barras da segunda rodada no cenário experimental, observa-se uma distribuição heterogênea dos valores de *fitness* entre os indivíduos da última

geração. Essa característica sugere que o cenário experimental conseguiu manter a diversidade da população ao longo das iterações, evitando uma convergência uniforme e precoce. Tal padrão de distribuição é um indicativo positivo de que a metodologia empregada no cenário experimental conseguiu explorar uma ampla gama do espaço de busca, ao mesmo tempo em que permitiu a convergência gradual para soluções otimizadas.

Os valores de *fitness* distribuídos entre diversos indivíduos destacam a eficácia do cenário experimental em mitigar os efeitos de convergência prematura, como observado em estudos clássicos sobre algoritmos evolutivos (Eiben & Smith, 2003 [76]). A dispersão dos valores, com uma maior presença de *fitness* intermediários em relação a baixos ou muito altos, sugere que o algoritmo foi capaz de equilibrar exploração e exploração, garantindo que novas regiões do espaço de busca fossem visitadas enquanto mantinha pressão seletiva suficiente para melhorar gradativamente a qualidade das soluções.

Os resultados da segunda rodada corroboram a hipótese de que o cenário experimental favorece um balanceamento adequado entre a busca local e global, reduzindo significativamente o risco de estagnação em mínimos locais. Esses resultados podem fornecer *insights* para o design de algoritmos genéticos robustos, destacando a relevância de estratégias que priorizem a diversidade populacional ao longo do processo evolutivo.

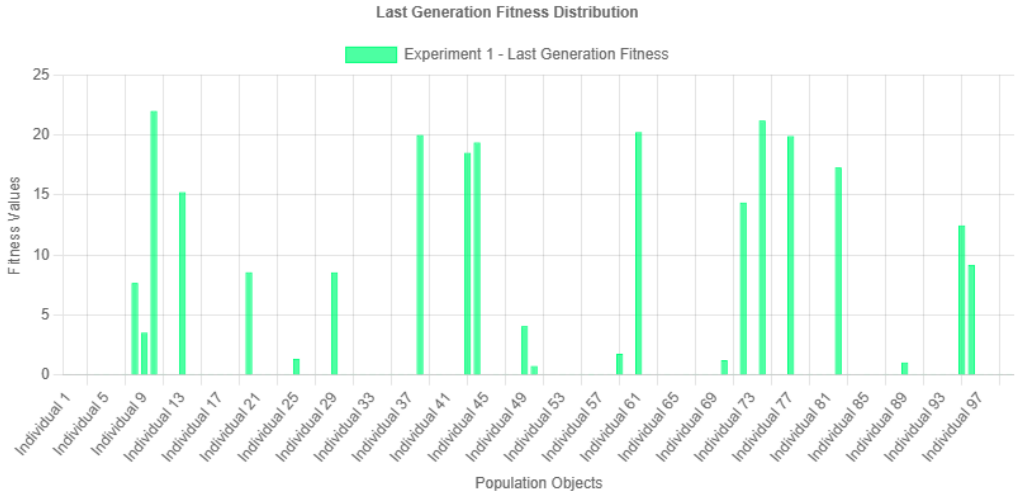


Figure 99: Gráfico de Barras do primeiro experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

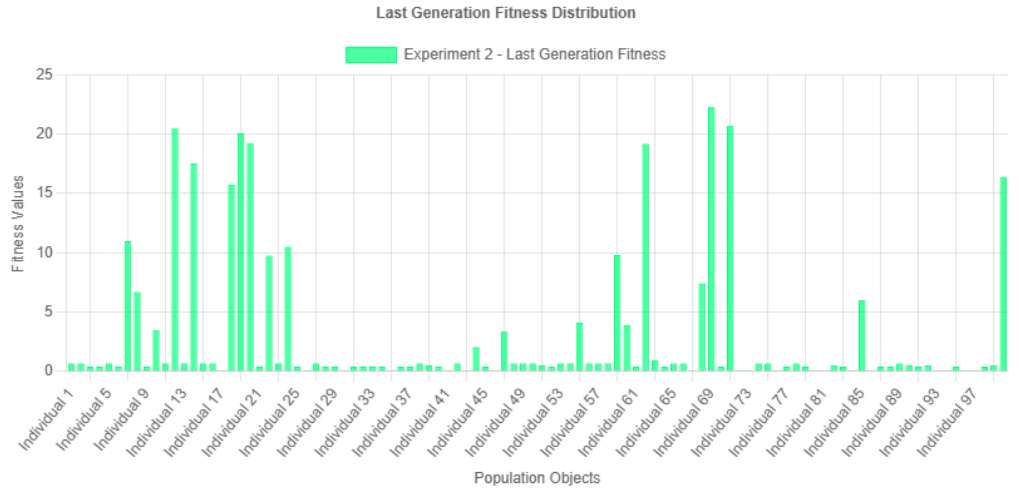


Figure 100: Gráfico de Barras do segundo experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

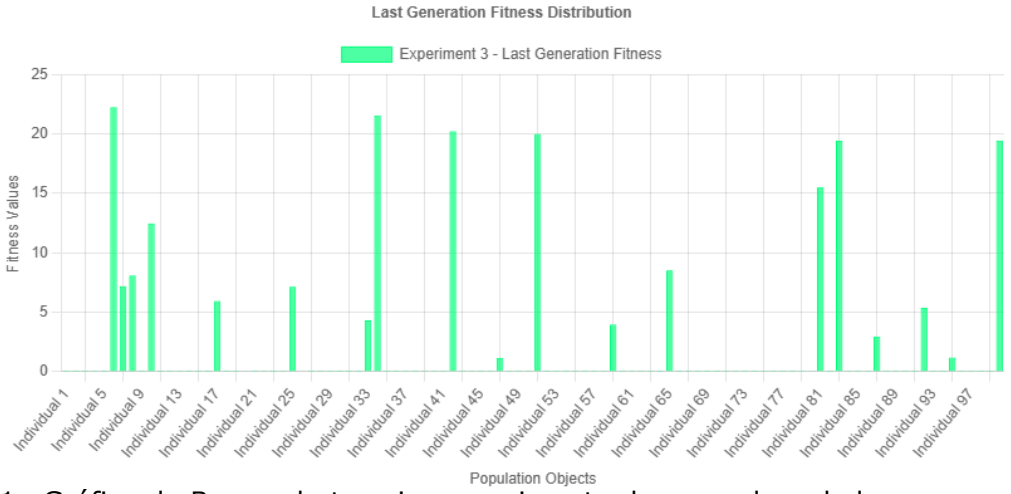


Figure 101: Gráfico de Barras do terceiro experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

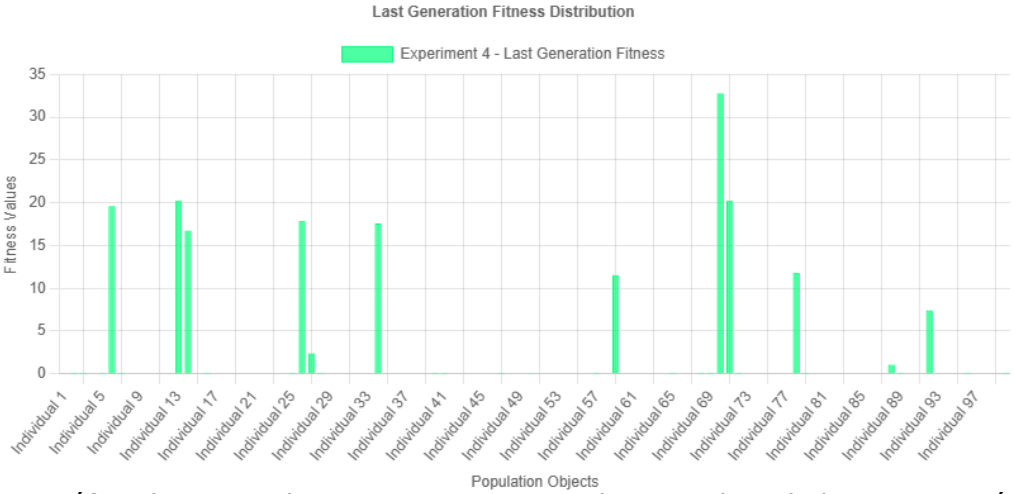


Figure 102: Gráfico de Barras do quarto experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação - Otimização da função de Rastrigin

Performance do AG longo de 40 gerações:

Média dos *fitnesses* com o passar das gerações

No cenário experimental, observa-se uma rápida redução dos valores médios de *fitness* nas primeiras gerações, indicando que o AG é eficiente em encontrar soluções promissoras em etapas iniciais. As médias de *fitness* continuam a decrescer gradativamente, estabilizando-se por volta da vigésima geração (figura 104. No entanto, há indícios de que o algoritmo enfrenta um possível efeito de *premature convergence* (Goldberg, 1989 [2]) , especialmente a partir da trigésima geração, onde as médias de *fitness* apresentam uma variação mínima entre as gerações subsequentes.

Essa estabilização pode ser atribuída à diminuição da diversidade genética na população, resultando em uma exploração limitada do espaço de busca. O desempenho do cenário experimental, contudo, é notavelmente consistente, com as soluções médias sendo progressivamente refinadas ao longo do tempo, o que reflete uma capacidade robusta de intensificação da busca no espaço de soluções.

Portanto, o cenário experimental converge de forma mais lenta e consistente na segunda rodada, permitindo uma maior exploração do espaço de busca. As médias de *fitness* finais do cenário experimental são inferiores às do cenário base na segunda rodada, indicando soluções de melhor qualidade. Ambos os cenários enfrentam o desafio da convergência

prematura, mas o impacto parece ser ainda mais pronunciado no cenário base, possivelmente devido a parâmetros de controle menos adequados para o problema.

Generations	Experiments				Average
	1º	2º	3º	4º	
gen 1	32.3442	4.3549	24.3110	31.6398	23.1625
gen 2	16.5869	26.1983	23.6112	32.7128	24.7773
gen 3	15.0726	15.5801	17.7934	29.6676	19.5284
gen 4	13.0434	11.3874	15.6464	17.9818	14.5148
gen 5	5.3408	4.3549	3.2209	17.9871	7.7259
gen 6	5.3408	4.3549	3.2209	2.8233	3.9350
gen 7	5.2527	4.3549	3.2209	2.8233	3.9129
gen 8	2.6375	4.3549	3.2209	2.3967	3.1525
gen 9	2.6375	4.3549	3.2209	2.2267	3.1100
gen 10	2.3985	4.3549	2.2157	2.2222	2.7978
gen 11	1.5894	3.4348	1.9074	1.0515	1.9958
gen 12	1.4825	3.4348	1.1774	0.8333	1.7320
gen 13	0.3284	3.4348	0.8678	0.8333	1.3661
gen 14	0.4353	3.4348	0.5596	0.8333	1.3157
gen 15	0.3919	2.2687	0.5596	0.8333	1.0134
gen 16	0.0694	2.1630	0.4449	0.6979	0.8438
gen 17	0.0694	1.4848	0.0821	0.8333	0.6174

Figure 103: Média dos *fitnesses* da primeira à décima sétima geração em todos os experimentos na segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

gen 18	0.0694	1.4848	0.0821	0.6094	0.5614
gen 19	0.0694	1.4848	0.0821	0.6094	0.5614
gen 20	0.0694	1.4848	0.0821	0.6094	0.5614
gen 21	0.0694	1.4848	0.0821	0.6094	0.5614
gen 22	0.0694	1.0058	0.0751	0.6094	0.4399
gen 23	0.0694	1.0058	0.0821	0.6094	0.4417
gen 24	0.0694	1.0058	0.0821	0.6094	0.4417
gen 25	0.0694	1.0058	0.0821	0.6094	0.4417
gen 26	0.0694	1.0058	0.0821	0.1883	0.3364
gen 27	0.0694	1.0058	0.0821	0.1883	0.3364
gen 28	0.0694	1.0058	0.0821	0.1883	0.3364
gen 29	0.0435	0.6736	0.0287	0.1883	0.2335
gen 30	0.0435	0.6736	0.0821	0.0849	0.2210
gen 31	0.0435	0.6736	0.0821	0.0849	0.2210
gen 32	0.0435	0.6736	0.0720	0.0849	0.2185
gen 33	0.0435	0.6736	0.0116	0.0849	0.2034
gen 34	0.0435	0.6736	0.0116	0.0849	0.2034
gen 35	0.0337	0.3926	0.0038	0.0849	0.1288
gen 36	0.0337	0.3926	0.0038	0.0044	0.1086

Figure 104: Média dos *fitnesses* da décima oitava à trigésima sexta geração em todos os experimentos na segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

gen 37	0.0337	0.0205	0.0038	0.0044	0.0156
gen 38	0.0337	0.0205	0.0038	0.0044	0.0156
gen 39	0.0337	0.0205	0.0038	0.0044	0.0156
gen 40	0.0275	0.0205	0.0038	0.0044	0.0141

Figure 105: Média dos *fitnesses* das últimas quatro gerações em todos os experimentos na segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

Melhores soluções em cada experimento:

Na segunda rodada do cenário experimental, observa-se uma melhora consistente nas soluções encontradas ao longo dos experimentos, com valores de *fitness* mínimos decrescendo gradualmente. Por exemplo, no Experimento 1 (figura 106), a melhor solução alcançada foi 0.0275, enquanto no Experimento 4 (figura 109) esse valor caiu para 0.0044. Essa redução progressiva indica um comportamento eficiente do algoritmo na exploração e exploração do espaço de busca, culminando em soluções otimizadas mais precisas.

Essa performance sugere que as condições experimentais proporcionaram diversidade genética suficiente e evitaram a convergência prematura, permitindo ao algoritmo refinar as soluções de forma estável ao longo das gerações.

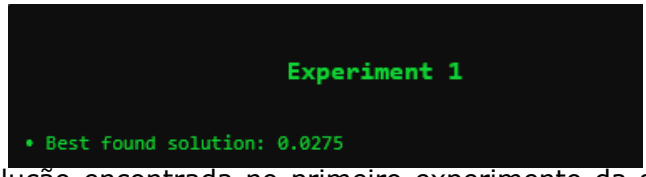


Figure 106: Melhor solução encontrada no primeiro experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

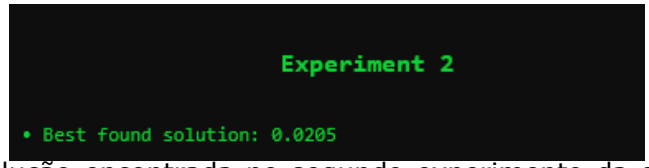


Figure 107: Melhor solução encontrada no segundo experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

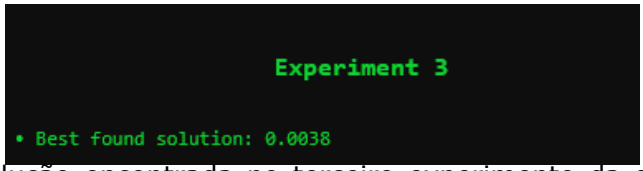


Figure 108: Melhor solução encontrada no terceiro experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

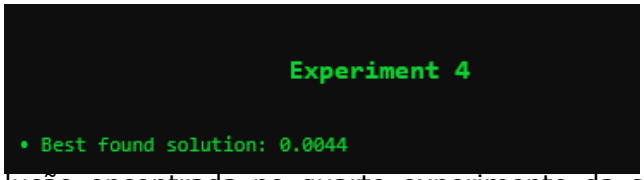


Figure 109: Melhor solução encontrada no quarto experimento da segunda rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

- **Rodada 3**:

Box-Plot: No cenário experimental, o comportamento do *fitness* ao longo das 40 gerações exibe uma melhora significativa nas primeiras gerações, como evidenciado pela redução acentuada dos valores máximos e medianos. O declínio inicial rápido sugere uma exploração eficiente do espaço de busca pelo algoritmo genético. A dispersão dos dados, representada pela amplitude interquartil (IQR) (Tukey, 1977 [75]), também diminui significativamente após as primeiras gerações, indicando um processo de convergência. No entanto, essa redução de dispersão pode levantar preocupações quanto à diversidade populacional, potencialmente associada ao fenômeno de convergência prematura, especialmente em momentos onde os *outliers* praticamente desaparecem. Isso sugere que o algoritmo pode estar explorando soluções similares, o que deve ser analisado quanto à sua robustez frente à busca global.

A terceira rodada reafirma a eficiência do cenário experimental na exploração inicial do espaço de busca, ao passo que o cenário base na terceira rodada exibia uma maior resiliência à perda de diversidade. Isso ressalta a necessidade de balancear a exploração inicial com mecanismos que evitem a convergência prematura nas gerações finais, especialmente em problemas com múltiplos ótimos locais, como a função Rastrigin.

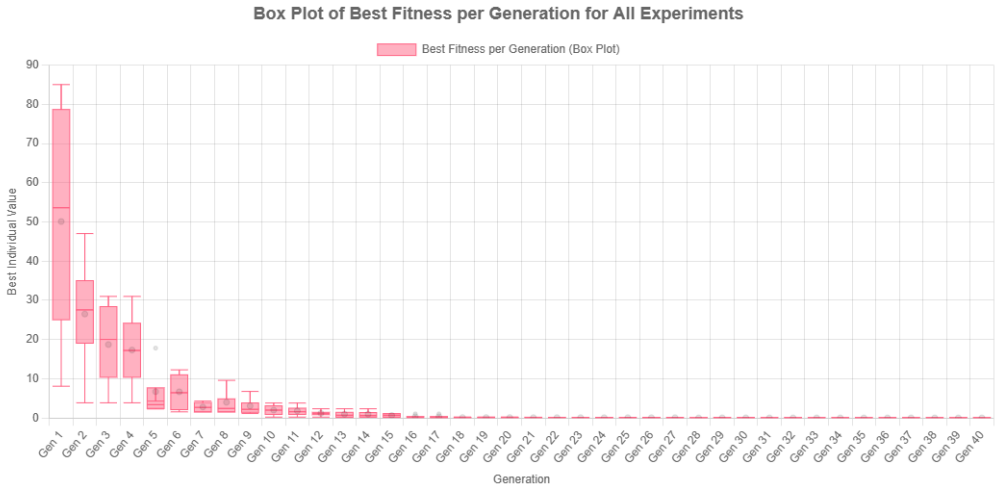


Figure 110: Box-Plot da terceira rodada para o cenário experimental de ponderação de taxas de crossover e mutação - Otimização da função de Rastrigin

Melhor Fitness por Geração em Cada Experimento:

Nos gráficos do cenário experimental para a terceira rodada, observa-se um padrão de convergência progressiva na redução dos valores de *fitness* ao longo das gerações. Os experimentos revelam uma rápida diminuição inicial do *fitness* nas primeiras gerações, o que pode ser atribuído a uma exploração inicial eficiente do espaço de busca pelo AG. Essa fase inicial reflete a capacidade do algoritmo em identificar soluções promissoras rapidamente.

A estabilização observada após cerca da décima geração indica a transição para a exploração fina do espaço de busca, onde o algoritmo refina as soluções já identificadas. A variação reduzida entre as melhores soluções nas gerações posteriores sugere um comportamento controlado e consistente, indicando que os parâmetros de ajuste do AG foram adequados para mitigar problemas de estagnação, como a convergência prematura.

Ao comparar os cenários base e experimental, na terceira rodada, pode-se observar que o cenário experimental se destaca por alcançar valores finais de *fitness* consistentemente melhores em todos os experimentos, destacando sua superioridade no refinamento das soluções ao longo das gerações.

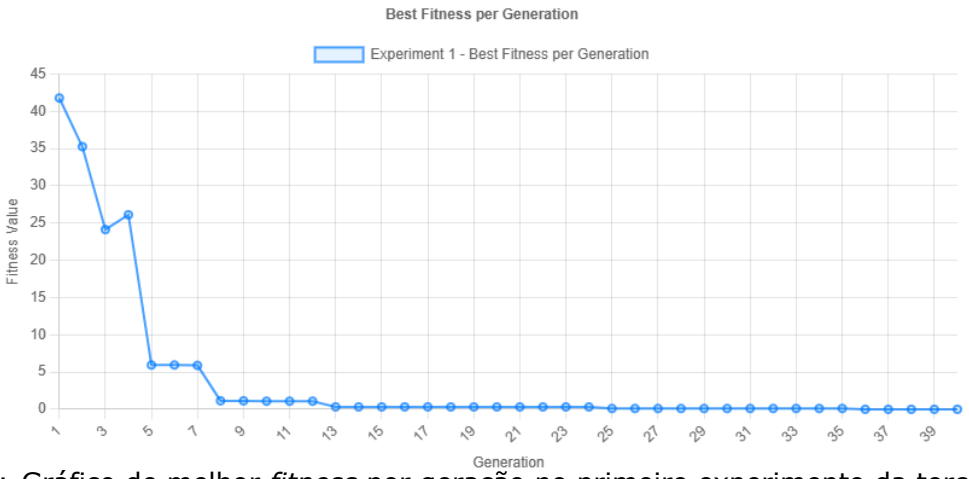


Figure 111: Gráfico de melhor *fitness* por geração no primeiro experimento da terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

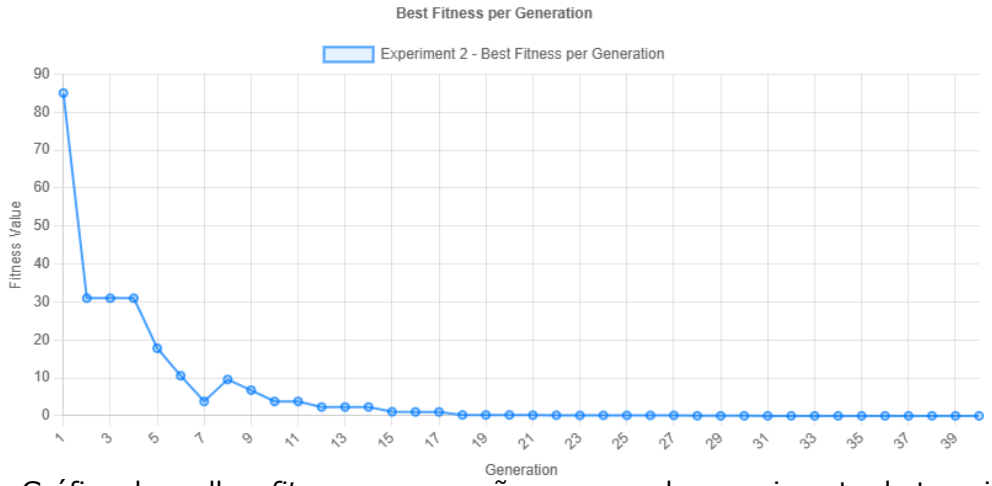


Figure 112: Gráfico de melhor *fitness* por geração no segundo experimento da terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

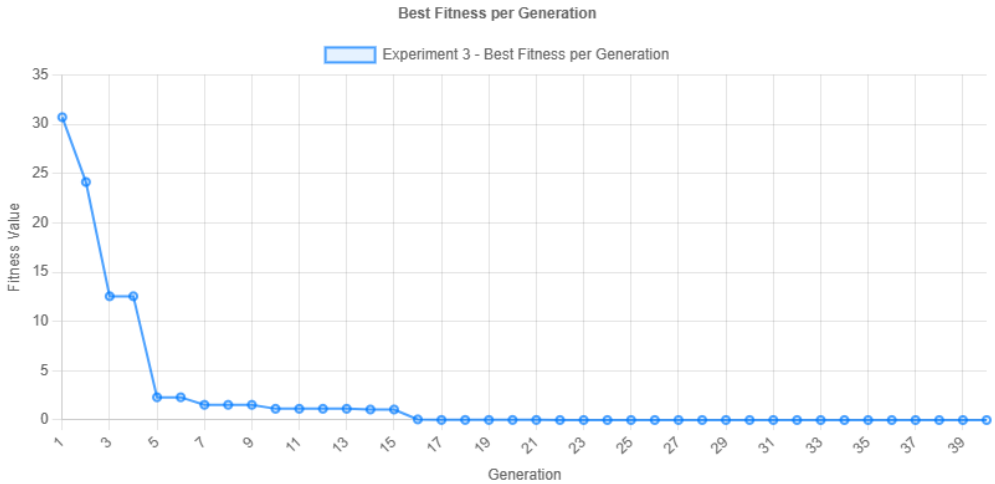


Figure 113: Gráfico de melhor *fitness* por geração no terceiro experimento da terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

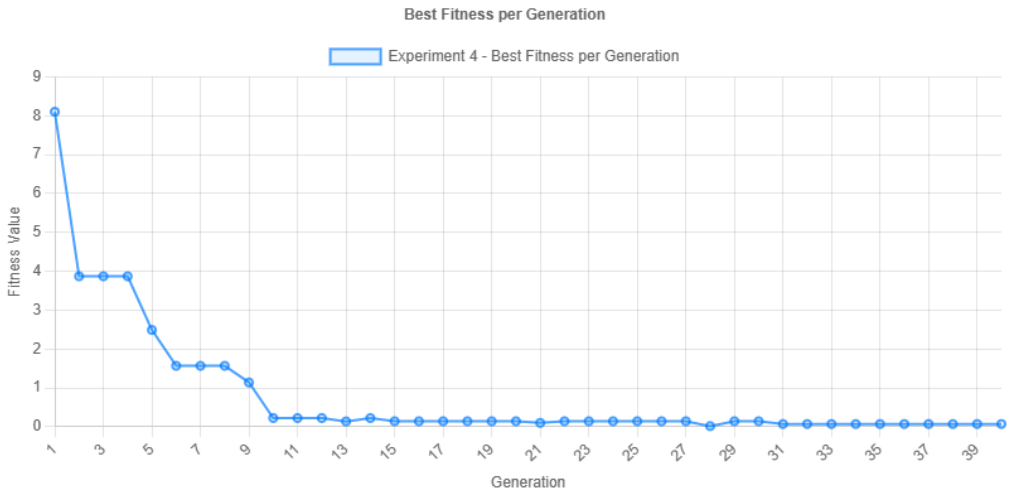


Figure 114: Gráfico de melhor *fitness* por geração no quarto experimento da terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

Gráficos de Barras da última geração de cada experimento:

Os gráficos de barras apresentados a seguir ilustram a distribuição dos valores de *fitness* na última geração de cada experimento da terceira rodada, dentro do cenário experimental de ponderação de taxas de *crossover* e mutação para a otimização da função de Rastrigin.

No primeiro experimento (Figura 115), a distribuição dos valores de *fitness* mostrou certa dispersão, com alguns indivíduos destacando-se com valores de *fitness* significativamente melhores. Isso evidencia a capacidade do algoritmo de encontrar soluções promissoras, embora ainda mantenha alguma diversidade residual na população.

No segundo experimento (Figura 116), houve uma maior concentração de valores de *fitness* intermediários. Este comportamento sugere que o algoritmo manteve uma diversidade moderada entre os indivíduos da população até a última geração, o que pode ser útil para evitar a convergência prematura e explorar regiões mais amplas do espaço de busca.

O terceiro experimento (Figura 117) apresentou uma distribuição mais uniforme dos valores de *fitness*, indicando menor pressão seletiva e uma configuração que favoreceu a manutenção da diversidade genética. Esse perfil pode ser estratégico para problemas que exigem maior exploração.

Por fim, no quarto experimento (Figura 118), a dispersão dos valores de *fitness* foi perceptível, com picos em indivíduos específicos. Este padrão reforça a eficiência do algoritmo em identificar soluções otimizadas enquanto preserva alguma variabilidade residual.

De forma geral, os gráficos revelam que o algoritmo genético conseguiu explorar o espaço de busca de forma eficaz, embora com diferentes graus de diversidade populacional entre os experimentos. Esses resultados são fundamentais para avaliar a influência das taxas de *crossover* e mutação na dinâmica evolutiva e no desempenho geral do algoritmo na otimização da função de Rastrigin.

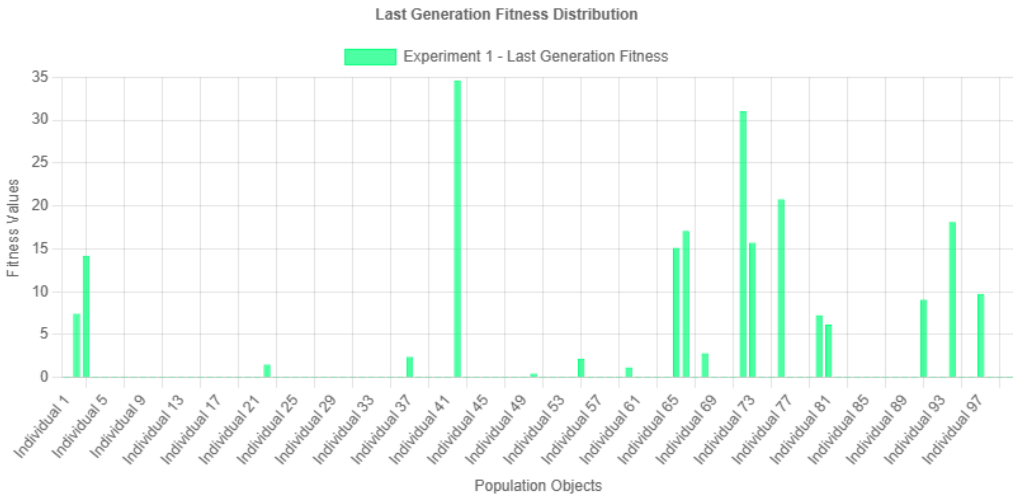


Figure 115: Gráfico de Barras do primeiro experimento da terceira rodada para o cenário experimental de ponderação de taxas de crossover e mutação - Otimização da função de Rastrigin

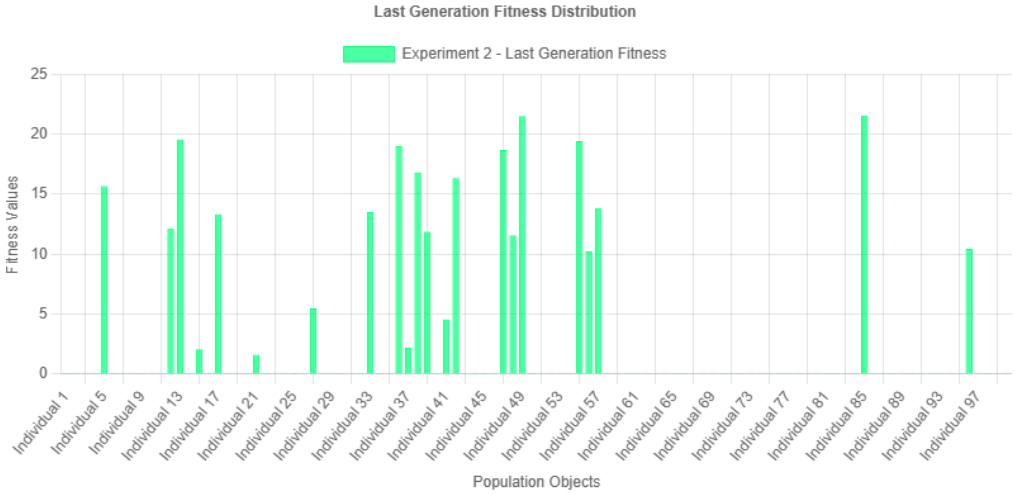


Figure 116: Gráfico de Barras do segundo experimento da terceira rodada para o cenário experimental de ponderação de taxas de crossover e mutação - Otimização da função de Rastrigin

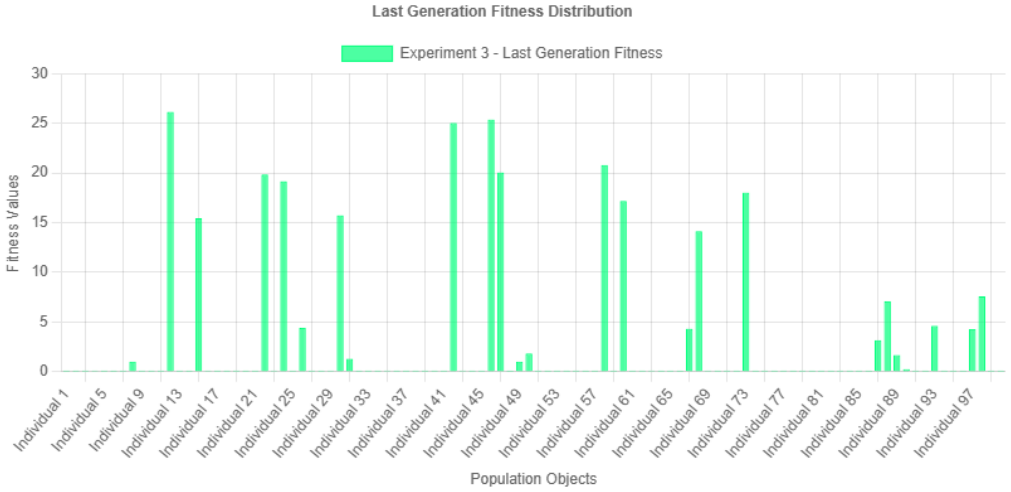


Figure 117: Gráfico de Barras do terceiro experimento da terceira rodada para o cenário experimental de ponderação de taxas de crossover e mutação - Otimização da função de Rastrigin

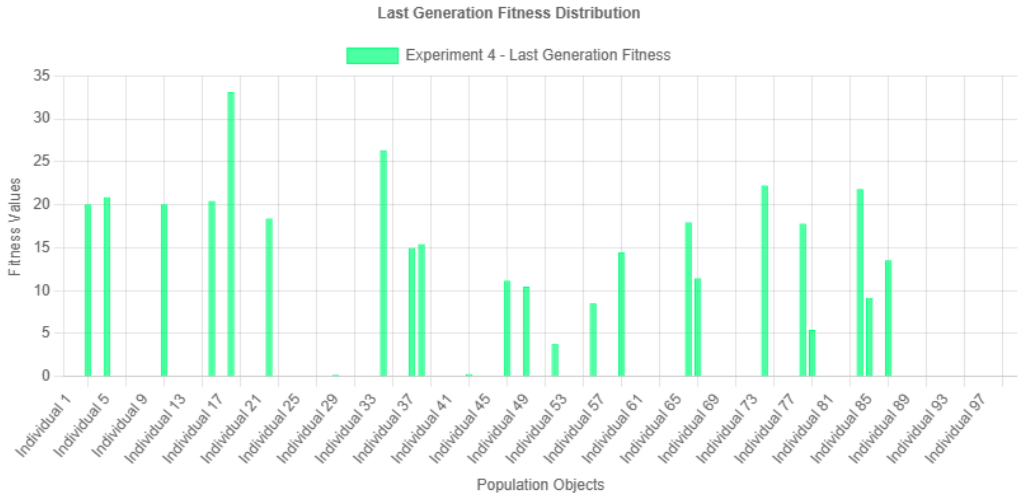


Figure 118: Gráfico de Barras do quarto experimento da terceira rodada para o cenário experimental de ponderação de taxas de crossover e mutação - Otimização da função de Rastrigin

Performance do AG longo de 40 gerações:

*** Média dos *fitnesses* com o passar das gerações***

Gerações Iniciais (Fig. 119): Durante as primeiras 17 gerações, é perceptível um rápido declínio nos valores médios de *fitness*. Este comportamento destaca a fase inicial de exploração do espaço de busca, na qual as operações de *crossover* e mutação desempenham um papel crucial na diversificação da população. A redução drástica do *fitness* nas primeiras cinco gerações demonstra a eficácia inicial na eliminação de soluções subótimas. No entanto, à medida que a busca se aproxima da geração 17, a média estabiliza, sugerindo uma transição gradual para a fase de exploração mais refinada.

Gerações Intermediárias (Fig. 120): Entre as gerações 18 e 36, observa-se uma diminuição mais lenta na média de *fitness*. O cenário experimental reflete uma exploração estável e um refinamento contínuo da qualidade das soluções. A aplicação das taxas de *crossover* e mutação ajustadas mostra-se eficaz na redução gradual dos valores médios, demonstrando que a população ainda explora novas combinações genéticas, mas em menor intensidade do que nas gerações iniciais.

Gerações Finais (Fig. 121): Nas últimas quatro gerações, a média de *fitness* se aproxima de valores mínimos consistentes. Isso sugere que o algoritmo alcançou uma convergência efetiva para uma região promissora do espaço de busca, com pouca variação entre os indivíduos da população. A estabilização do *fitness* médio reforça a hipótese de que o algoritmo genético, no cenário experimental, foi capaz de identificar soluções quase ótimas para a função de Rastrigin.

A alta taxa de mutação permitiu que o algoritmo explorasse mais regiões do espaço de busca, evitando a convergência prematura para ótimos locais. Isso é refletido nos gráficos, onde o *fitness* médio diminui gradualmente, indicando que o AG continuou a encontrar soluções melhores ao longo das gerações. Uma mutação alta pode retardar a convergência final, o que é percebido na persistência de *fitnesses* intermediários em algumas gerações mais avançadas. No entanto, isso também garante uma busca mais consistente antes de fixar-se nos ótimos globais.

O baixo *crossover* diminuiu o impacto da recombinação no algoritmo, limitando a mistura de material genético entre indivíduos. Isso fez com que a população dependesse mais da mutação para descobrir novas soluções. Com menos ênfase no *crossover*, o AG se torna menos propenso a explorações rápidas, permitindo um refinamento mais gradual das soluções.

Generations	Experiments				Average
	1º	2º	3º	4º	
gen 1	76.5601	85.0395	30.7064	8.1029	50.1022
gen 2	47.0427	31.0118	24.1382	3.8720	26.5162
gen 3	27.5340	31.0118	12.5561	3.8720	18.7435
gen 4	21.8886	31.0118	12.5561	3.8720	17.3321
gen 5	4.3093	17.8266	2.3090	2.4917	6.7341
gen 6	12.2826	10.5852	2.3090	1.5696	6.6866
gen 7	4.3093	3.8138	1.5461	1.5696	2.8097
gen 8	3.2986	9.6017	1.5461	1.5696	4.0040
gen 9	2.8611	6.7759	1.5461	1.1413	3.0811
gen 10	2.8611	3.8138	1.1464	0.2263	2.0119
gen 11	2.0667	3.8138	1.1464	0.2263	1.8133
gen 12	1.0561	2.3497	1.1464	0.2263	1.1946
gen 13	0.2510	2.3497	1.1464	0.1400	0.9718
gen 14	0.2510	2.3497	1.0708	0.2263	0.9745
gen 15	0.2510	1.1035	1.0708	0.1445	0.6424
gen 16	0.0365	1.0595	0.0686	0.1445	0.3273
gen 17	0.0365	1.0595	0.0134	0.1445	0.3135

Figure 119: Média dos *fitnesses* da primeira à décima sétima geração em todos os experimentos na terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

gen 18	0.0365	0.2496	0.0134	0.1445	0.1110
gen 19	0.0365	0.2496	0.0134	0.1445	0.1110
gen 20	0.0365	0.2496	0.0134	0.1445	0.1110
gen 21	0.0365	0.1965	0.0134	0.1026	0.0872
gen 22	0.0365	0.1734	0.0050	0.1445	0.0899
gen 23	0.0365	0.1506	0.0050	0.1445	0.0841
gen 24	0.0098	0.1506	0.0050	0.1445	0.0775
gen 25	0.0098	0.1506	0.0050	0.1445	0.0775
gen 26	0.0003	0.1506	0.0050	0.1445	0.0751
gen 27	0.0003	0.1506	0.0050	0.1445	0.0751
gen 28	0.0003	0.0365	0.0050	0.0194	0.0153
gen 29	0.0003	0.0312	0.0050	0.1445	0.0452
gen 30	0.0003	0.0312	0.0050	0.1445	0.0452
gen 31	0.0003	0.0235	0.0050	0.0731	0.0254
gen 32	0.0003	0.0235	0.0035	0.0731	0.0251
gen 33	0.0003	0.0235	0.0015	0.0731	0.0246
gen 34	0.0003	0.0235	0.0015	0.0731	0.0246
gen 35	0.0003	0.0235	0.0015	0.0731	0.0246
gen 36	0.0003	0.0235	0.0015	0.0731	0.0246

Figure 120: Média dos *fitnesses* da décima oitava à trigésima sexta geração em todos os experimentos na terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

gen 37	0.0003	0.0235	0.0000	0.0731	0.0242
gen 38	0.0003	0.0235	0.0000	0.0731	0.0242
gen 39	0.0003	0.0137	0.0000	0.0731	0.0218
gen 40	0.0003	0.0137	0.0000	0.0731	0.0218

Figure 121: Média dos fitnesses das últimas quatro gerações em todos os experimentos na terceira rodada para o cenário experimental de ponderação de taxas de crossover e mutação

Melhores soluções em cada experimento:

Os resultados indicam que a alteração para uma taxa de mutação elevada (80.8%) e *crossover* reduzida (10%) foi uma estratégia acertada. Isso levou a uma melhor exploração do espaço de busca e permitiu alcançar soluções mais próximas do ótimo global no cenário experimental, enquanto o cenário base ficou aquém devido às limitações impostas pela baixa mutação. Este comportamento evidencia a importância de equilibrar a exploração (mutação) e a exploração local (*crossover*) em problemas de otimização complexos como a função de Rastrigin.

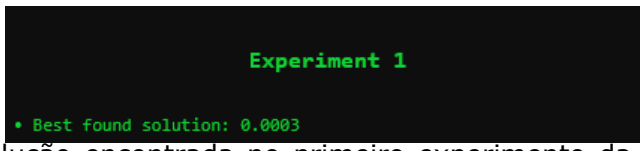


Figure 122: Melhor solução encontrada no primeiro experimento da terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

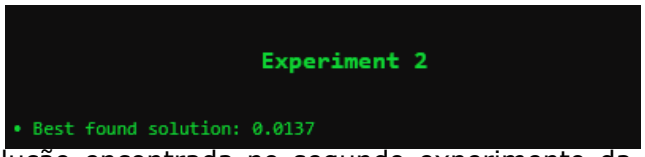


Figure 123: Melhor solução encontrada no segundo experimento da terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

Figure 124: Melhor solução encontrada no terceiro experimento da terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

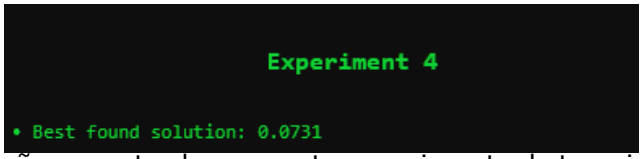


Figure 125: Melhor solução encontrada no quarto experimento da terceira rodada para o cenário experimental de ponderação de taxas de *crossover* e mutação

2 Tamanho da População e Normalização Linear

Para os próximos experimentos, será conduzida uma análise detalhada do impacto do tamanho da população e da normalização linear no desempenho do algoritmo genético, utilizando a **Função F6**, cuja descrição encontra-se no apêndice em 2. O planejamento experimental seguirá as diretrizes apresentadas na tabela 7:

1. **GA1 modificado**: Serão realizadas **5 rodadas**, variando o tamanho da população nos valores 15 (primeira), 45 (segunda), 75 (terceira), 105 (quarta) e 135 (quinta), sem aplicação de normalização linear e alterando o número de experimentos de 4 para 5.

2. **GA2 modificado**: Serão conduzidas 5 rodadas adicionais com as mesmas variações no tamanho da população em 15 (primeira), 45 (segunda), 75 (terceira), 105 (quarta) e 135 (quinta), porém com a aplicação de normalização linear (min=1; max=100) e alterando o número de experimentos de 4 para 5.

Essa abordagem foi planejada para permitir uma avaliação sistemática e comparativa da influência do tamanho da população e da normalização linear no desempenho do algoritmo. Os resultados serão analisados em termos de métricas como a melhor solução encontrada, a velocidade de convergência, a diversidade mantida ao longo das gerações e o tempo total de execução, permitindo conclusões robustas sobre a eficácia de cada configuração.

Para interpretação dos resultados, vale ressaltar que o ponto de máximo global conhecido da função Schaffer N.6 ocorre em

$$f(0,0) \approx 1 \tag{8}$$

1. **Resultados do teste pra GA1 modificado**:

   No primeiro teste da série de experimentos com a Função F6, foram avaliadas cinco rodadas do algoritmo genético (GA1), sem aplicação de normalização linear, e com diferentes tamanhos de população: 15, 45, 75, 105 e 135 indivíduos, em que cada rodada foi configurada para executar com um número de 5 experimentos. A análise teve como objetivo observar como o tamanho da população impacta o desempenho do algoritmo em termos de aptidão média máxima (*Mean Best Fitness*) por geração.

   O gráfico da figura 126 apresenta o valor médio da melhor aptidão ao longo de 40 gerações para as cinco configurações de tamanho de população. Cada linha representa uma rodada específica com um tamanho de população diferente.

   Rodada 1 (População: 15 indivíduos):

   A aptidão média inicia em aproximadamente 0,6, indicando um desempenho baixo comparado às rodadas com populações maiores. A curva mostra estabilidade desde o início, com pouca evolução ao longo das gerações. O menor tamanho populacional parece ter reduzido a diversidade genética, limitando o algoritmo em escapar de ótimos locais ou em explorar eficientemente o espaço de busca.

   Rodada 2 (População: 45 indivíduos):

   A aptidão média inicial foi maior que a da primeira rodada ($\approx$ 0,7). Observa-se um crescimento mais consistente nas primeiras gerações, estabilizando por volta da décima geração. O aumento da população proporcionou maior diversidade genética, permitindo uma exploração melhor do espaço de busca e uma melhora significativa no desempenho.

   Rodada 3 (População: 75 indivíduos): Esta configuração mostrou um desempenho inicial superior ao da rodada 2 ($\approx$ 0,8). A curva estabiliza rapidamente após a sétima geração, atingindo valores próximos a 0,9. O tamanho intermediário da população apresentou um equilíbrio entre exploração e diversidade genética, permitindo uma convergência eficiente sem comprometer a diversidade.

   Rodada 4 (População: 105 indivíduos):

   A Rodada 4 começa com uma aptidão média inicial de aproximadamente 0,8, similar à Rodada 3. É evidente que, embora a Rodada 4 tenha um desempenho superior em termos de aptidão média máxima, ela demora ligeiramente mais tempo (em torno de

uma geração após) para se estabilizar do que a Rodada 3. Este atraso relativo pode ser atribuído ao aumento da população, que, embora beneficie a diversidade genética e permita melhores soluções, requer mais iterações para a redistribuição dos indivíduos em direção à solução ótima.

Rodada 5 (População: 135 indivíduos):

A Rodada 5 exibe um comportamento interessante ao longo das gerações. Embora inicialmente superior à Rodada 4 em termos de aptidão média (até a oitava geração), o desempenho estabilizado é inferior ao da Rodada 4. A Rodada 5 começa com valores ligeiramente superiores aos da Rodada 4. Esse comportamento pode ser atribuído à maior população inicial, que proporciona maior diversidade genética, permitindo identificar soluções promissoras de forma mais rápida nas primeiras gerações. Por volta da oitava geração, a aptidão média da Rodada 5 começa a decrescer em relação à Rodada 4. Após a estabilização (em torno da décima primeira geração), o desempenho da Rodada 5 permanece consistentemente inferior ao da Rodada 4, sugerindo que o aumento da população além de 105 indivíduos não resultou em ganhos significativos.

O comportamento da Rodada 5 indica que, embora populações maiores possam acelerar a exploração inicial do espaço de busca (gerações iniciais), elas também podem sofrer com menor pressão seletiva ou redução da eficácia do *crossover* em populações mais densas, levando a estabilizações em valores inferiores. Isso sugere a existência de um ponto ótimo de população, onde o equilíbrio entre diversidade e convergência é melhor.

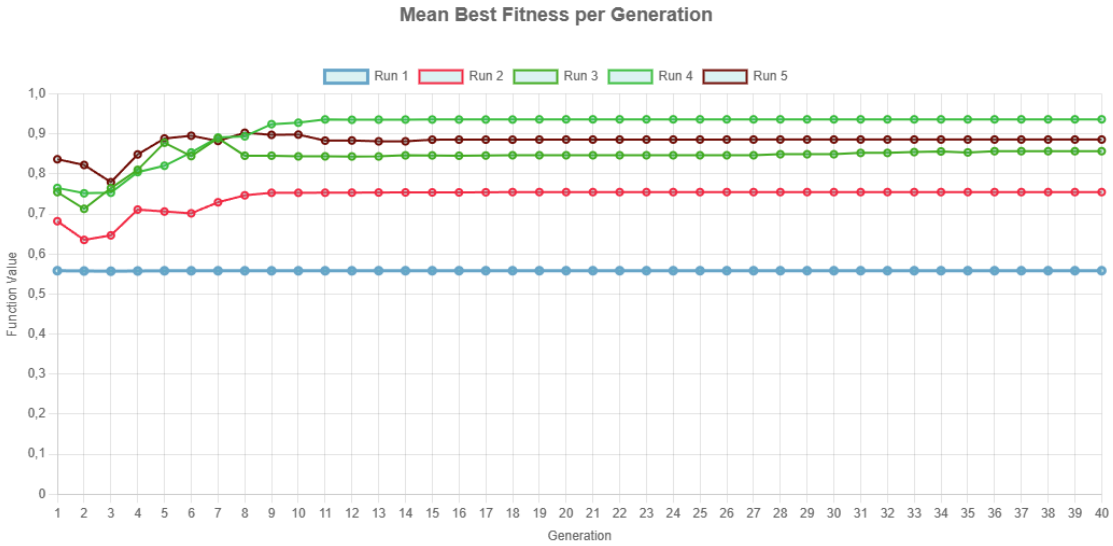


Figure 126: Resultados para o teste GA1 modificado utilizando a função F6

2. **Resultados do teste para GA2 modificado**:
   - **Estocasticidade aumentada:** O comportamento altamente oscilatório das curvas evidencia que a normalização linear introduziu um impacto significativo na seleção dos indivíduos. A amplificação das diferenças de *fitness* dentro do intervalo $[1, 100]$ favoreceu mudanças abruptas na probabilidade de seleção, promovendo maior aleatoriedade no processo de reprodução. Isso reduziu a capacidade do algoritmo de explorar e refinar gradualmente as regiões promissoras do espaço de busca, dificultando a convergência.
   - **Redução do impacto do tamanho da população:** Normalmente, um tamanho populacional maior favorece a diversidade genética inicial, o que melhora as chances de convergência para uma solução ótima. Contudo, sob a normalização linear, esse efeito foi suprimido. A escala aplicada parece ter mascarado os benefícios da diversidade inicial, tornando o comportamento do algoritmo quase independente do tamanho da população.
   - **Baixa convergência:** A convergência, que se caracteriza por uma redução progressiva da diversidade genética em direção à solução ótima, foi comprometida

pela normalização. O comportamento errático sugere que a pressão seletiva amplificada pelo intervalo $[1, 100]$ introduziu "saltos" frequentes no espaço de busca, reduzindo a eficácia da exploração local ao redor das melhores soluções identificadas.

- **Impacto da escala de normalização:** A amplificação das diferenças relativas de *fitness* no intervalo $[1, 100]$ favoreceu excessivamente os indivíduos ligeiramente melhores, resultando em:
  - Eliminação prematura de indivíduos menos aptos, reduzindo a diversidade.
  - Seleção excessiva de indivíduos com *fitness* apenas marginalmente superior, o que ocasionou comportamentos oscilatórios em vez de refinamento gradual.

  Uma escala menor, como $[1, 10]$, poderia reduzir esses efeitos adversos, preservando parte do comportamento esperado.

Portanto, a normalização linear com intervalo $[1, 100]$ prejudicou significativamente a estabilidade e eficiência do algoritmo genético, tornando o comportamento errático e eliminando a influência positiva do tamanho da população.

Reduzir o intervalo de normalização linear, minimizando o impacto das diferenças relativas de *fitness*, poderia resultar em um resultado menos cabuloso. Ao que parece, avaliar a necessidade de normalização apenas em cenários onde os valores de *fitness* sejam muito dispersos pode ser o ideal. Essas medidas podem restaurar a correlação esperada entre tamanho da população e desempenho, promovendo uma convergência mais consistente.

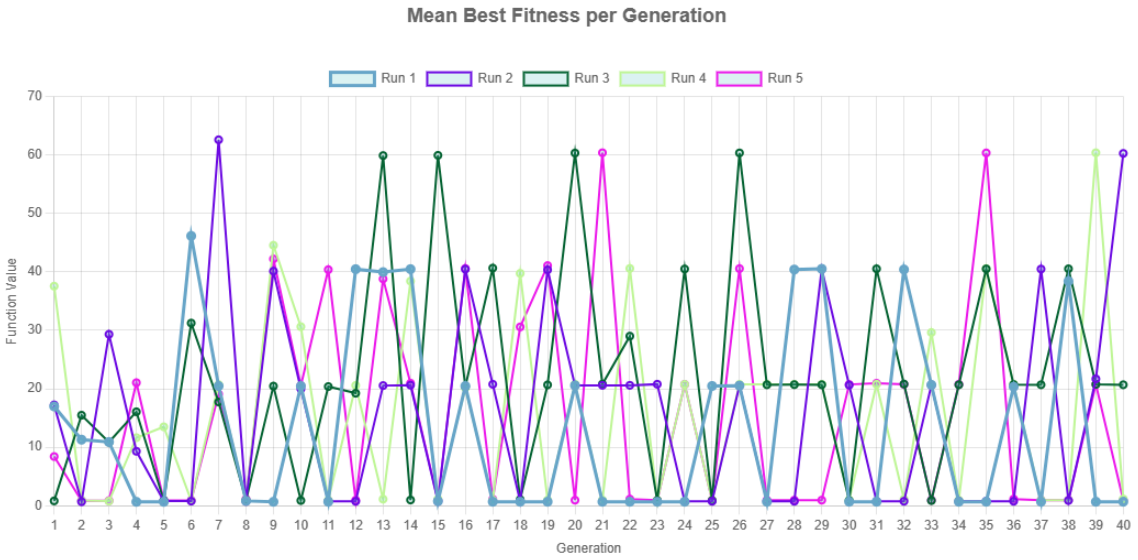


Figure 127: Resultados para o teste GA2 modificado utilizando a função F6

3 Plano Experimental para Operadores de Crossover

Nesta série experimental, o objetivo é avaliar o impacto dos diferentes tipos de *crossover* no desempenho do algoritmo genético. São realizadas análises considerando apenas a modificação nos operadores de *crossover*, utilizando o **GA1** da tabela 7, ajustado para **5 experimentos**, com 5 rodadas independentes para cada tipo de *crossover*.

1. **GA1 modificado (5 experimentos)**:
   - **Crossover de 1 ponto**: 5 rodadas independentes.
   - **Crossover de 2 pontos**: 5 rodadas independentes.
   - **Crossover de 3 pontos**: 5 rodadas independentes.

A hipótese inicial é que o *crossover* de 1 ponto, sendo o mais simples, apresentará maior rapidez na convergência, porém com menor diversidade nas gerações finais. O *crossover* de 2 pontos e uniforme, por outro lado, são esperados para fornecer maior diversidade e, potencialmente, melhores resultados, ao custo de um aumento no número de gerações necessárias para estabilização.

Os próximos testes serão conduzidos utilizando a **Função de Ackley**[16], cuja descrição detalhada está contida no apêndice, no apêndice, mais precisamente em 3.

Vale lembrar que a função Ackley tridimensional possui seu ponto de mínimo global em

$$f(0,0) = 0 \tag{9}$$

Resultados para o Crossover de 1 Ponto

A figura 128 apresenta os resultados obtidos utilizando o crossover de 1 ponto.

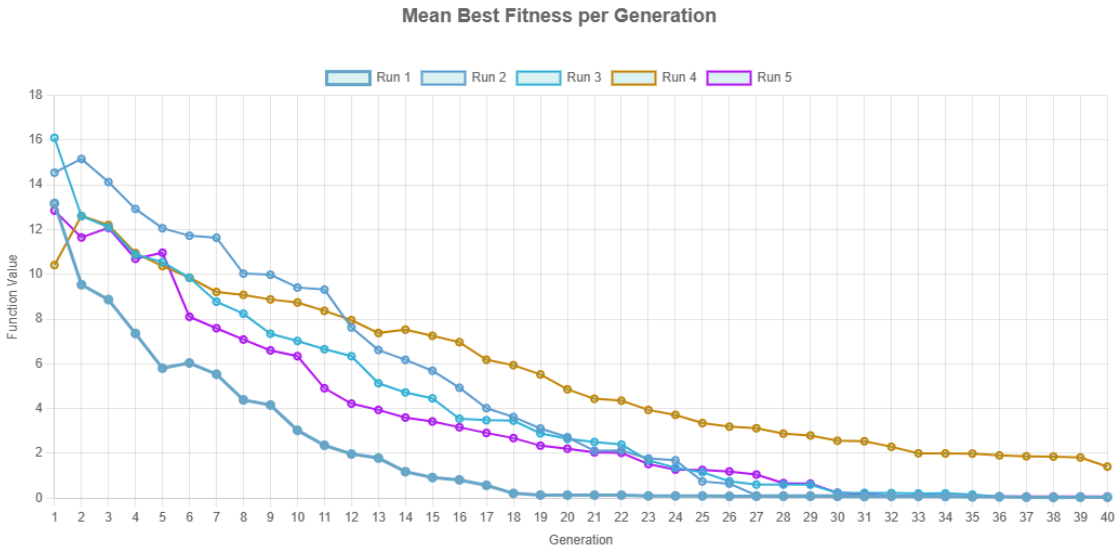


Figure 128: Resultados para o crossover de 1 ponto utilizando a função Ackley.

O *crossover* de 1 ponto demonstrou uma convergência gradual e uniforme ao longo das gerações. Após aproximadamente 25 gerações, com exceção da rodada 4, todas alcançaram valores próximos de zero, indicando boa capacidade de explorar a função e encontrar o ótimo global.

As trajetórias das cinco rodadas são bastante próximas, de maneira geral, sugerindo que o *crossover* de 1 ponto é confiável e consistente para a função Ackley. Esse tipo de *crossover* apresentou os melhores resultados gerais entre os três testados, com todas as rodadas convergindo para valores muito próximos ao ótimo global.

Resultados para o Crossover de 2 Pontos

A figura 129 apresenta os resultados obtidos utilizando o crossover de 2 pontos.

[16] Esta função é deveras fascinante, pois remete a um buraco negro prestes a desintegrar um corpo celeste.

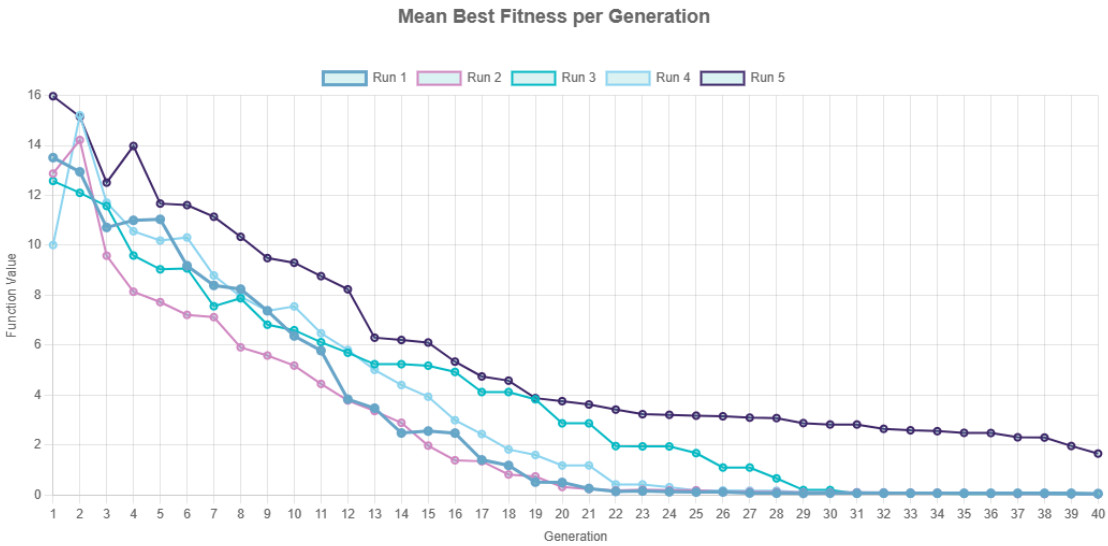


Figure 129: Resultados para o crossover de 2 pontos utilizando a função Ackley.

O *crossover* de 2 pontos apresentou uma convergência inicial ligeiramente mais rápida do que o *crossover* de 1 ponto, especialmente nas primeiras 15 gerações. No entanto, após este ponto, as rodadas demonstraram uma desaceleração na convergência.

Este tipo de *crossover* apresentou maior dispersão entre as rodadas. Algumas (e.g., *Run* 5) demoraram mais para atingir valores próximos de zero, indicando que a confiabilidade dos resultados é inferior ao *crossover* de 1 ponto.

Embora apresente boa velocidade inicial, o crossover de 2 pontos não alcançou valores tão baixos quanto o crossover de 1 ponto. Isso sugere que ele é menos eficiente para encontrar o ótimo global da função Ackley.

Resultados para o Crossover Uniforme

A figura 130 apresenta os resultados obtidos utilizando o crossover uniforme.

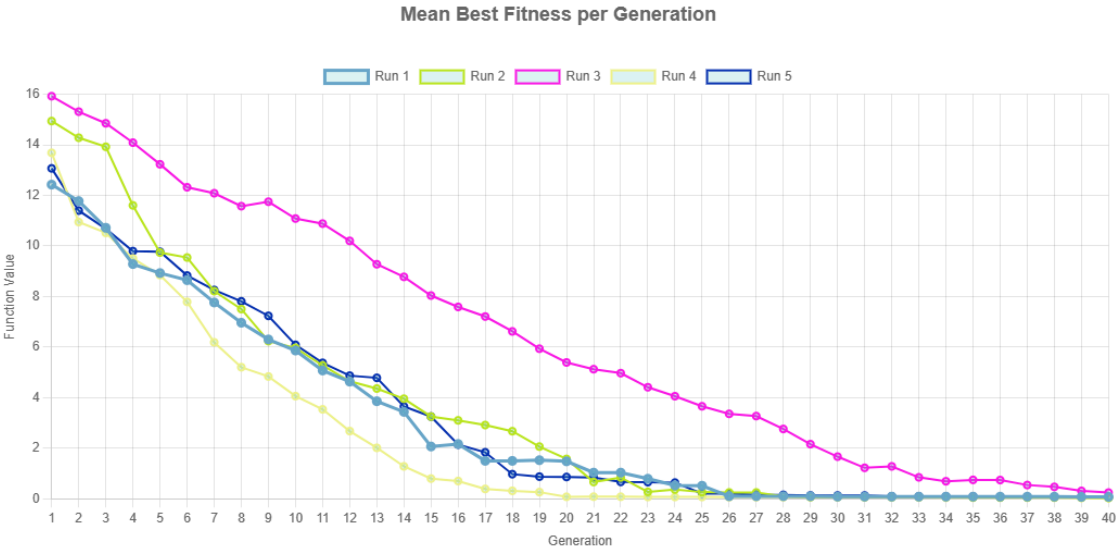


Figure 130: Resultados para o crossover uniforme utilizando a função Ackley.

O *crossover* uniforme demonstrou uma maior variabilidade na convergência inicial e ao longo de todo o processo de otimização. Algumas rodadas convergiram rapidamente, enquanto outras

apresentaram uma evolução bem mais lenta.

As trajetórias das rodadas foram bastante dispersas. A rodada 3, por exemplo, estabilizou-se em valores mais altos do que as outras, indicando maior aleatoriedade no comportamento deste tipo de *crossover*.

Os experimentos demonstraram que o *crossover* de 1 ponto é a configuração mais indicada para a função Ackley, devido à sua estabilidade e capacidade de atingir o ótimo global com consistência, promovendo mais equilíbrio entre convergência e custo computacional. O *crossover* de 2 pontos ou uniforme podem ser utilizados em cenários que priorizem maior chance de convergência, embora possam ser menos eficiente no longo prazo.

4 Elitismo

Nesta seção experimental, o objetivo é avaliar o impacto do **elitismo** no desempenho do algoritmo genético. O elitismo, como sabe-se, é uma técnica que preserva os melhores indivíduos de uma geração para a próxima, garantindo que o desempenho não se degrade ao longo do tempo. A comparação será realizada entre duas configurações:

- **GA1 (sem elitismo):** Utiliza a configuração padrão sem a aplicação do elitismo.
- **GA3 (com elitismo):** Incorpora o elitismo, garantindo que os melhores indivíduos de cada geração sejam preservados.

É utilizada a **Função Levy**[17] como o *benchmark* para este experimento. O mínimo global da função ocorre em $f(1,1) = 0$, e suas características multimodais representam um desafio significativo para algoritmos de otimização.

Configuração do Experimento

O plano experimental segue os parâmetros descritos na tabela **??**. Cada configuração será testada em **7 rodadas independentes**, permitindo uma análise estatística dos resultados.

Parâmetro	**Valor**
Número de Experimentos	7
Número de Gerações	40
Tamanho da População	100
Taxa de Crossover (%)	65,0
Taxa de Mutação (%)	0,8
Intervalo de Busca	[-100, 100]
Objetivo	Minimização
Elitismo	Ativado no GA3

Table 8: Parâmetros experimentais para avaliação do elitismo.

Os experimentos foram configurados para comparar o desempenho do **GA1 (sem elitismo)** com o **GA3 (com elitismo)**, como é definido na tabela 7.

Resultados Para as Rodadas Com e Sem Elitismo

Os resultados apresentados indicam uma diferença interessante entre as configurações com elitismo e sem elitismo. O elitismo, como mencionado, preserva os melhores indivíduos de cada geração, o que garante que a melhor solução encontrada até uma determinada geração não será perdida nas subsequentes. Isso pode acelerar a convergência inicial, mas também aumenta o risco de exploração limitada ou convergência prematura, especialmente em funções multimodais, como a Função Levy, onde múltiplos ótimos locais podem atrapalhar a busca global.

Sem elitismo (figura 131): O algoritmo convergiu rapidamente nas primeiras gerações (geralmente até a geração 10). Porém, várias rodadas ficaram aquém do ótimo global. Isso é perceptível nas curvas que mostram uma estabilização precoce em valores acima de 0 (ótimo global da função Levy). Existe mais variabilidade entre rodadas, evidenciada pelas diferenças nas alturas das curvas ao longo das gerações. A exploração excessiva sem retenção dos melhores

[17] A definição da função Levy está disponível no apêndice, mais precisamente em 4.

indivíduos parece ter dificultado a estabilização no ótimo global. Algumas rodadas terminaram em valores ligeiramente longe do ótimo.

Com elitismo (figura 132): O algoritmo converge de forma ligeiramente mais lenta nas gerações iniciais (1 a 10). Isso sugere que a introdução do elitismo desacelera a exploração inicial. Entretanto, as curvas mostram muito mais consistência em alcançar o ótimo global (0). Praticamente todas as rodadas convergem para valores ótimos. A variabilidade entre rodadas é reduzida, indicando uma maior estabilidade proporcionada pela preservação dos melhores indivíduos. A convergência é mais lenta nas primeiras gerações, mas o ganho de qualidade ao longo do tempo é evidente, com quase todas as rodadas atingindo o ótimo global. Esse comportamento reflete um equilíbrio mais adequado entre exploração (busca de novas soluções) e explotação (melhoria das melhores soluções já encontradas) (Goldberg,1989 [2]).

Para funções multimodais como a Levy, onde o equilíbrio entre exploração e refinamento é crucial, o elitismo se mostrou uma estratégia benéfica. Ele garante que o progresso obtido seja preservado, ao mesmo tempo que mantém a diversidade necessária para uma busca eficiente.

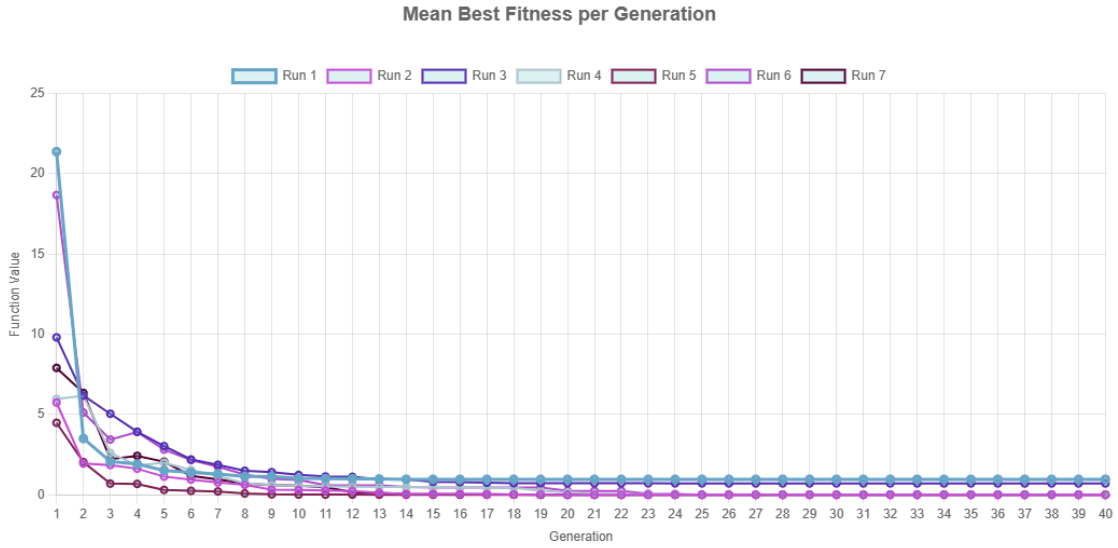


Figure 131: Resultados para as rodadas sem elitismo otimizando a função Levy

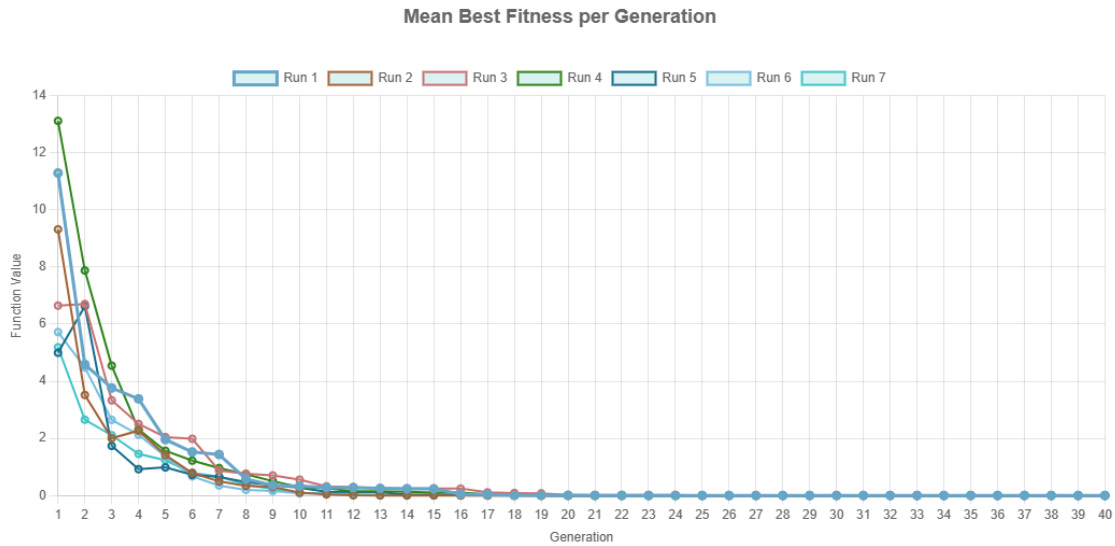


Figure 132: Resultados para as rodadas com elitismo otimizando a função Levy

5 *Steady-State*

Nesta seção experimental, o objetivo é avaliar o impacto do método de reprodução *Steady-State* no desempenho do algoritmo genético. O *Steady-State*, como sabe-se, é uma estratégia que substitui apenas uma fração da população a cada geração, permitindo uma maior estabilidade evolutiva. O experimento foi dividido em duas configurações:

- **GA4 (com duplicados):** Permite que indivíduos duplicados sejam mantidos na população, utilizando a configuração padrão do *Steady-State*.
- **GA5 (sem duplicados):** Evita a presença de indivíduos duplicados na população. É utilizando um *gap* de 10%, ou seja, substituindo apenas 10% da população a cada geração. Aqui, foi feita a comparação com os resultados obtidos ao executar **GA1** com 5 experimentos em cada rodada de um total de 5 rodadas.

Para esses experimentos, foi utilizada a função **Drop-Wave**[18] como *benchmark*, cujo mínimo global ocorre em

$$f(0,0) = -1 \tag{10}$$

A *Drop-Wave* é amplamente conhecida por seu caráter multimodal, apresentando múltiplos máximos e mínimos locais, o que a torna um desafio significativo para algoritmos de otimização.

A Tabela 7 apresenta os parâmetros experimentais utilizados para a avaliação do *Steady-State*.

Resultados do Steady-State Sem Gap

Os experimentos conduzidos com o método **Steady-State sem gap** apresentaram um desempenho significativamente limitado (figura 133) em termos de convergência para valores ótimos da função objetivo. Observa-se que a média do melhor fitness por geração permanece praticamente constante ao longo de todas as execuções, sem evidências de melhorias progressivas. Essa característica sugere que a estratégia de substituição no *Steady-State* sem gap não fornece a pressão seletiva necessária para explorar e otimizar a busca no espaço de soluções.

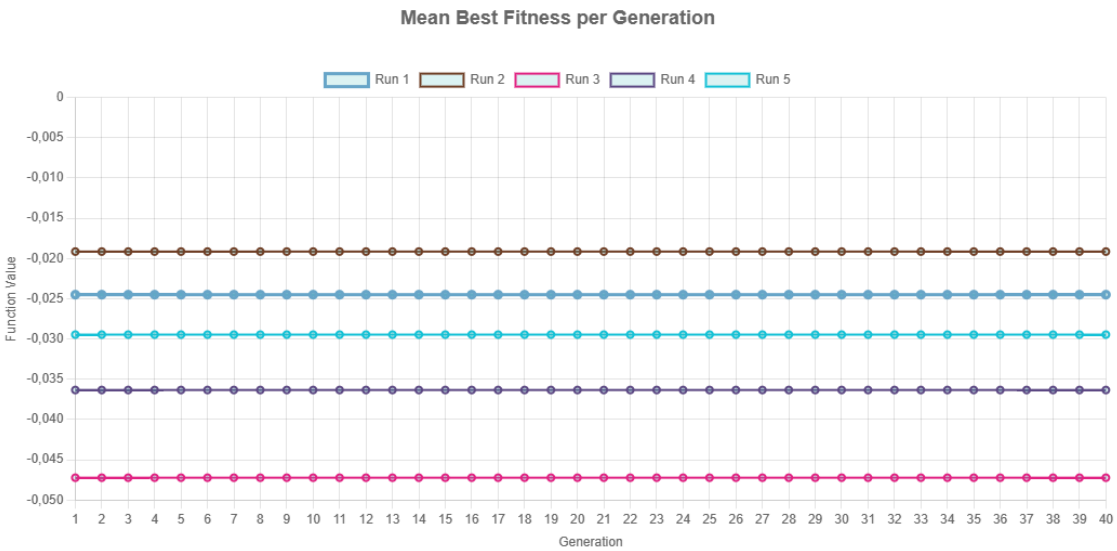


Figure 133: Resultados para as rodadas com Steady-State para otimização da função Drop-Wave

No **Steady-State sem gap**, a substituição de indivíduos na população é realizada de forma limitada, preservando a maior parte dos indivíduos da geração anterior. Isso resulta em uma

[18] A definição da função *Drop-Wave* está detalhada no apêndice, em 5.

taxa de exploração extremamente baixa, uma vez que apenas uma pequena fração da população é atualizada a cada iteração. Como consequência:

- O algoritmo tende a ficar preso em valores iniciais sub-ótimos, sem realizar melhorias significativas ao longo das gerações.
- A diversidade populacional é rapidamente perdida, levando a uma estagnação precoce da evolução.
- Os indivíduos mais aptos não conseguem exercer impacto suficiente na composição da próxima geração, devido à baixa taxa de reprodução e mutação efetiva.

Apesar do baixo desempenho em termos de qualidade de solução, o tempo de execução dos experimentos foi excepcionalmente curto, em torno de **5 segundos por rodada**. Esse resultado pode ser atribuído ao número reduzido de operações de reprodução e avaliação, uma vez que a maior parte da população permanece inalterada entre gerações.

Os resultados indicam que o ***Steady-State* sem gap**, embora eficiente em termos de tempo de execução, não é adequado para problemas que demandam uma exploração efetiva do espaço de busca. A falta de renovação populacional limita severamente a capacidade do algoritmo de escapar de ótimos locais e alcançar soluções globais. No entanto, sua simplicidade e rapidez de execução podem ser úteis em cenários onde a convergência rápida e local é suficiente ou quando combinado com outras estratégias que promovam maior exploração.

Resultados para *Steady-State* Sem Duplicados com GAP=10%

Os experimentos realizados com o *Steady-State* sem duplicados e gap de 10% demonstraram um desempenho significativamente superior em termos de convergência e qualidade das soluções alcançadas. A configuração aplicada promoveu um equilíbrio eficiente entre exploração e explotação, com resultados que destacam a robustez dessa abordagem para funções multimodais como a Drop-Wave.

- **Qualidade da Convergência:** O *Steady-State* sem duplicados com gap de 10% permitiu uma renovação gradual da população, assegurando que novos indivíduos fossem introduzidos em cada geração. Isso resultou em uma convergência mais eficiente, com valores de *fitness* próximos ao ótimo global sendo alcançados de maneira consistente ao longo das rodadas.
- **Diversidade Genética:** A exclusão de duplicados evitou que indivíduos idênticos ocupassem o espaço populacional, promovendo maior diversidade genética e reduzindo o risco de estagnação em ótimos locais.
- **Custo Computacional:** Embora o tempo médio por rodada tenha sido de aproximadamente 250 segundos, o aumento no custo computacional foi compensado pela qualidade superior das soluções encontradas, tornando essa configuração ideal para cenários em que a qualidade do resultado é prioritária.

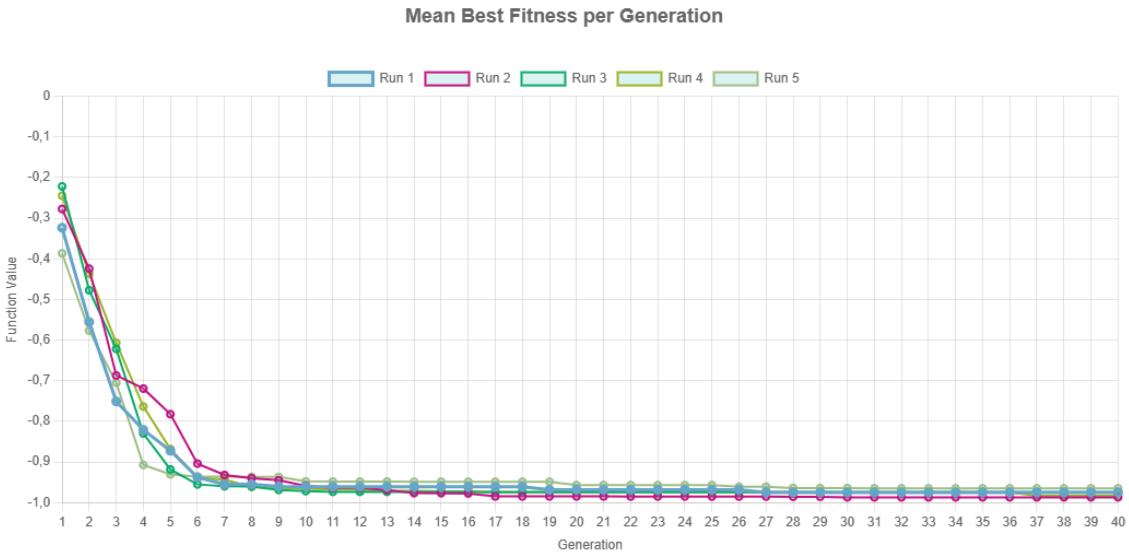


Figure 134: Resultados para as rodadas com *Steady-State* sem Duplicados com gap=10% para otimização da função Drop-Wave

Os experimentos sem a utilização do *Steady-State* (parâmetros **GA1** com número de experimentos igual a 5) apresentaram uma convergência mais lenta e menos eficiente. Essa configuração evidenciou limitações em contextos que demandam maior robustez e exploração do espaço de busca, especialmente para funções multimodais como a Drop-Wave.

- **Qualidade da Convergência:** A ausência de mecanismos para preservar a diversidade genética resultou em uma explotação prematura, comprometendo a capacidade do algoritmo de explorar regiões mais amplas do espaço de busca.
- **Custo Computacional:** O tempo médio por rodada foi bastante reduzido em comoparação com as execuções em que se utilizou *steady-state* sem duplicados. O tempo médio foi de 25 segundos, o que torna essa configuração adequada para aplicações onde o tempo de execução ou o custo computacional é uma restrição crítica.
- **Comparação com *Steady-State*:** Apesar do menor custo computacional, os resultados obtidos sem *Steady-State* foram inferiores em qualidade, reforçando a importância de mecanismos que promovam diversidade genética e adaptação ao longo das gerações.

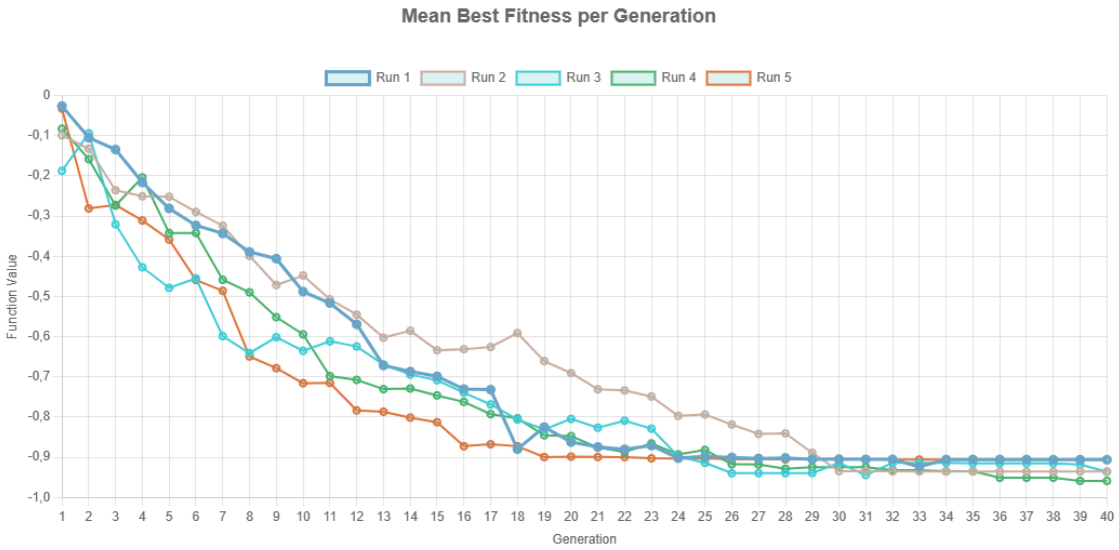


Figure 135: Resultados para as rodadas de GA1 com nºexp=5 para otimização da função Drop-Wave

e Conclusão dos Resultados e Análises

Os experimentos realizados no GADEMO Web proporcionaram uma análise detalhada dos efeitos de parâmetros específicos de Algoritmos Genéticos, como taxas de *crossover*, *mutação*, *elitismo* e variações do método *Steady-State*, aplicados a diferentes funções de *benchmark*. A diversidade das funções testadas, incluindo Rastrigin, Ackley, Levy, *Drop-Wave*, entre outras, garantiu uma validação mais robusta da plataforma desenvolvida e permitiu identificar padrões de comportamento dos AGs em contextos variados de otimização.

Os principais resultados alcançados demonstram:

- A eficácia do método *Steady-State sem duplicados com GAP de 10%*, que apresentou melhor convergência para soluções ótimas na função *Drop-Wave*, ainda que com maior custo computacional. Esse resultado destaca a importância de mecanismos para evitar convergência prematura e duplicações desnecessárias na população.
- A **influência das taxas de *crossover* e *mutação*** na exploração do espaço de busca, mostrando que parâmetros ajustados adequadamente para cada função contribuem para uma convergência mais eficiente.
- A comparação entre experimentos com e sem *elitismo* evidenciou que a preservação de indivíduos com maior aptidão pode acelerar o processo de busca, mas deve ser usada com moderação para evitar a perda de diversidade genética.

Os gráficos gerados permitiram uma visualização clara da evolução do melhor *fitness* ao longo das gerações, consolidando a capacidade da plataforma de fornecer análises visuais precisas e intuitivas. Além disso, a utilização de múltiplas funções de *benchmark* e configurações distintas amplia o caráter generalizável dos resultados, alinhando-se com o estado da arte no campo dos AGs.

f Contribuição para o Estado da Arte

O GADEMO Web se destaca pela combinação de uma interface moderna, escalável e acessível com um sistema robusto de processamento de algoritmos genéticos. Em comparação com ferramentas existentes, a plataforma oferece:

- **Flexibilidade no ajuste de parâmetros**, permitindo uma ampla gama de experimentos com diferentes operadores e configurações.

- **Visualização interativa dos resultados**, facilitando o entendimento dos efeitos de cada parâmetro sobre a convergência do algoritmo.
- **Implementação modular baseada em *microsserviços***, o que garante escalabilidade e possibilita futuras expansões.

Essa contribuição é relevante tanto para o ensino de algoritmos evolutivos, ao tornar conceitos complexos mais acessíveis, quanto para a pesquisa, ao proporcionar uma ferramenta flexível e escalável para experimentos comparativos e otimização de funções complexas.

g Considerações Finais da Discussão

A análise dos resultados reforça a importância de explorar diferentes estratégias em Algoritmos Genéticos, como o uso de *Steady-State* e ajustes finos dos parâmetros. Além disso, os testes com diversas funções de *benchmark* evidenciam a capacidade do GADEMO em lidar com cenários distintos de otimização, destacando sua aplicabilidade em problemas do mundo real.

Com base nos resultados obtidos, trabalhos futuros podem explorar otimizações adicionais na eficiência computacional, integração de novas funções objetivo e implementação de técnicas híbridas com aprendizado de máquina, consolidando o GADEMO Web como uma ferramenta essencial no contexto educacional e de pesquisa.

h Promovendo a Reprodutibilidade

Como parte do compromisso com a ciência aberta e reprodutível, o sistema *GADEMO* foi disponibilizado para acesso público. Essa disponibilização inclui tanto a interface *frontend* quanto a API *backend*. Adicionalmente, o código-fonte completo está disponível em repositórios públicos no GitHub, permitindo que outros pesquisadores ou interessados possam explorar, testar e expandir este trabalho.

O *frontend* pode ser acessado no link:

https://gademo-zxig.onrender.com

Enquanto isso, a documentação da API (*Swagger*) está disponível em:

https://gademo-api.onrender.com/docs

Além disso, os repositórios de código-fonte, contendo o *frontend* e *backend*, estão disponíveis no GitHub:

- ***Frontend***: https://github.com/palomaflsette/gademo-front
- ***Backend***: https://github.com/palomaflsette/GAdemo-api

Essa disponibilização busca promover a transparência, incentivar o reuso do código em projetos futuros e facilitar a validação dos experimentos realizados neste trabalho.

9 Conclusões & Trabalhos futuros

a Trabalhos Científicos

Os resultados alcançados neste trabalho destacam o potencial de melhorias substanciais no sistema GADEMO, especialmente no que diz respeito à flexibilização da avaliação de funções objetivo e à adaptação a contextos de otimização mais diversificados. Durante o desenvolvimento, emergiu a proposta de um módulo adaptativo para avaliação dinâmica de funções objetivo, integrando estratégias como:

- **Amostragem ampla inicial:** uma abordagem exploratória para identificar regiões promissoras sem depender de informações prévias sobre o ótimo.
- **Refinamento iterativo por grade adaptativa:** um método eficiente para concentrar a busca em torno das regiões mais relevantes, ajustando progressivamente os limites de busca.
- **Penalidades dinâmicas:** uma possível aplicação para restringir intervalos de busca sugeridos pelo usuário, promovendo maior alinhamento com os resultados da análise inicial.

Essa metodologia visa superar as limitações do sistema atual, permitindo que o GADEMO suporte funções arbitrárias e métricas de avaliação personalizáveis. Contudo, para sua implementação prática, são necessários estudos futuros para abordar os seguintes desafios:

1. **Adaptação a Funções Complexas:** Investigar como o módulo pode lidar com superfícies multimodais, regiões de descontinuidade e outros desafios inerentes a funções complexas.
2. **Eficiência Computacional:** Desenvolver soluções que reduzam os custos computacionais do refinamento iterativo, tornando o processo mais escalável.
3. **Métricas Alternativas:** Criar mecanismos de avaliação independentes da função objetivo, permitindo análises mais robustas e generalizáveis.

Adicionalmente, foi identificado que essa proposta possui um potencial significativo para se expandir em projetos futuros, especialmente em áreas como otimização multiobjetivo e aprendizado de máquina. O uso combinado de *adaptive grid search* e *adaptive random search* foi considerado como uma abordagem inicial promissora para coletar informações sobre o espaço de busca. Ainda que não implementada no contexto deste trabalho, esta ideia aponta para a possibilidade de explorar novos paradigmas na otimização de funções arbitrárias sem restrição.

1 Perspectivas de Pesquisa

Com base nas discussões realizadas, propõe-se que esta abordagem seja desenvolvida em trabalhos futuros com foco em:

- Investigações mais profundas sobre heurísticas independentes para definição de intervalos de busca iniciais em funções desconhecidas;
- Aplicações do método em problemas de engenharia complexos, como ajuste de hiperparâmetros em redes neurais profundas;
- Análise comparativa com métodos tradicionais de otimização, como *Powell* e *BFGS*, avaliando o impacto da análise exploratória no desempenho global.

Dessa forma, espera-se que as ideias apresentadas neste trabalho sirvam como base para projetos futuros, contribuindo para a evolução de algoritmos genéticos e para a otimização computacional em contextos diversos.

b Aprimoramento da Plataforma

Além dos avanços científicos discutidos, identificou-se a necessidade de aprimoramentos na plataforma GADEMO para melhorar a experiência do usuário e otimizar a eficiência operacional do sistema. Tais aprimoramentos incluem melhorias de usabilidade, integração com servidores mais sólidos e a expansão do suporte a dispositivos móveis. Estes esforços visam não apenas facilitar o uso da ferramenta, mas também potencializar sua aplicabilidade em cenários acadêmicos e práticos.

1 Seção de Tour com Instruções

Uma seção de *tour* interativo será desenvolvida, com o objetivo de orientar o usuário sobre o funcionamento de cada elemento da interface da plataforma. Esta funcionalidade permitirá que novos usuários compreendam rapidamente as principais funcionalidades do GADEMO, reduzindo a curva de aprendizado e incentivando a adoção da ferramenta em contextos educacionais e profissionais.

2 Pop-up de Descrição de *Bugs*

A introdução de um sistema de notificações *pop-up* detalhando os *bugs* conhecidos da plataforma eliminará a necessidade de inspecionar elementos manualmente. Este recurso promoverá maior transparência no uso do sistema e fornecerá instruções claras para contornar problemas específicos, reduzindo o atrito na experiência do usuário.

3 Versão Mobile Aprimorada

A expansão e otimização da versão mobile da plataforma permitirá um uso mais eficiente em dispositivos móveis, facilitando o acesso em qualquer lugar. Este aprimoramento será particularmente útil para estudantes e profissionais que necessitam acompanhar experimentos em andamento ou realizar análises rápidas enquanto estão fora do ambiente de trabalho.

4 Integração com Servidores da Universidade

Atualmente, a plataforma opera em servidores gratuitos, o que pode resultar em lentidão e limitações de desempenho. A integração com servidores institucionais, como os da universidade, trará benefícios substanciais em termos de estabilidade, capacidade de processamento e confiabilidade. Essa melhoria será essencial para atender a demandas mais complexas, como experimentos com grandes volumes de dados ou configurações computacionalmente intensivas.

5 Motivações e Benefícios

As melhorias propostas são motivadas pela necessidade de tornar a plataforma mais acessível, intuitiva e robusta. Os benefícios incluem:

- Maior engajamento de usuários devido a uma interface mais amigável;
- Redução do tempo gasto para solucionar problemas técnicos;
- Capacidade de suportar projetos de maior escala e complexidade;
- Acessibilidade ampliada através de dispositivos móveis, promovendo flexibilidade no uso.

Dessa forma, espera-se que esses aprimoramentos consolidem o GADEMO como uma ferramenta versátil e eficaz tanto para uso acadêmico quanto profissional, atendendo às demandas crescentes por soluções de otimização personalizáveis.

A Apêndice

a Apêndice A - Funções de *Benchmarck*

1 Função Rastrigin

A **função Rastrigin** (Leonid A. Rastrigin, 1974 [77]) é amplamente usada em otimização matemática e inteligência artificial, como um *benchmark* para testar algoritmos de otimização. Ela pertence à classe das funções multimodais, ou seja, apresenta vários mínimos locais. Essa característica torna a função bastante desafiadora para algoritmos de otimização, pois é "fácil" cair em um mínimo local ao invés do mínimo global.

Definição da Função

A função Rastrigin é dada pela equação 11 em n variáveis:

$$f(\mathbf{x}) = An + \sum_{i=1}^{\infty} [x_i^2 - Acos(2_i)] \tag{11}$$

Onde:

- $\mathbf{x} = (x_1, x_2, ..., x_n)$ é o vetor de variáveis de entrada;
- A é uma constante, geralmente escolhida como $A = 10$;
- n é o número de variáveis.

Características principais

1. Mínimo global:
   - O mínimo global ocorre em $\mathbf{x} = (0, 0, ..., 0)$ para todas as variáveis n.
   - O valor da função nesse ponto é $f(0, 0, ..., 0) = 0$.
2. Mínimos locais:
   - A função possui uma grande quantidade de mínimos locais devido ao termo oscilatório $-Acos(2\pi x_i)$, o que cria um padrão ondulado;
   - Esse comportamento aumenta o desafio de encontrar o mínimo global.
3. Escalabilidade:
   - A função pode ser definida em qualquer número de variáveis n, em qualquer dimensão, o que a torna flexível para testar algoritmos em diferentes configurações.
4. Intervalo de busca:
   - a função é frequentemente avaliada no intervalo $x_i \in [-5.12, 5.12] \ \forall \ i$.

Função Rastrigin Tridimensional

Como o GADEMO foi desenvolvido para lidar com funções tridimensionais[19], será considerado este fato para os experimentos e a função pode ser visualizada[20] na imagem 136.

[19] Como diria Carl Sagan, "a quarta dimensão está além de nossa intuição e percepção cotidiana, mas não é menos real". No entanto, ao restringir-se às otimizações de funções de duas variáveis os gráficos tornam-se não apenas mais "visíveis", mas também amigáveis para os meros mortais. Afinal, se já é difícil imaginar como uma maçã plana em Flatland lidaria com um cubo, quem somos nós para ousar dominar gráficos quadridimensionais sem um portal interdimensional? Talvez fosse necessário um buraco de minhoca para visualizar isso... ou um café bem forte.

[20] Se for de interesse, o código MATLAB para a plotagem da função foi disponibilizado no apêndice b.

Para $n = 2$, a função Rastrigin pode ser escrita como:

$$f(x, y) = 20 + x^2 - 10cos(2\pi x) + y^2 - 10cos(2\pi y) \tag{12}$$

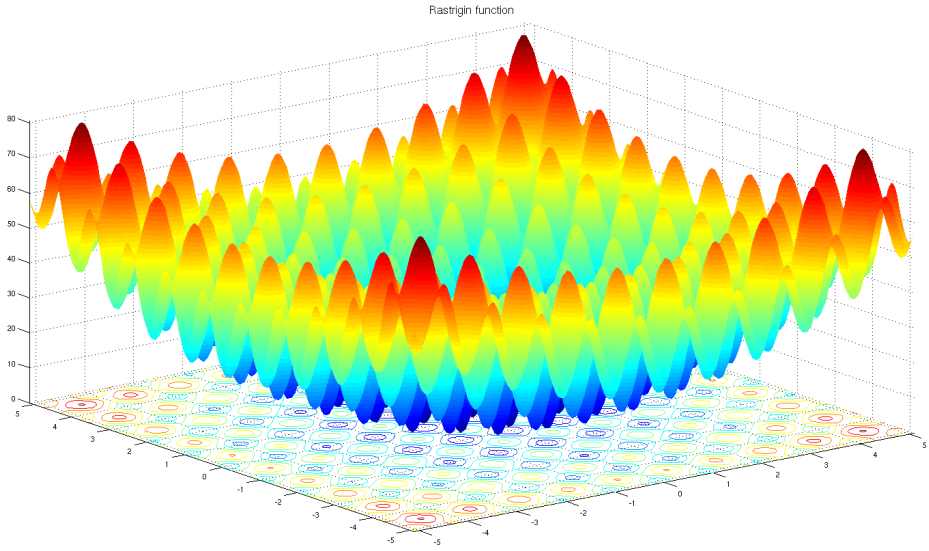


Figure 136: Função Rastrigin com 3 dimensões

2 Função F6

A **função F6** (J. D. Schaffer, 1989 [78]), também conhecida como a *Função Ackley de Sexta Forma*, é uma função de teste utilizada frequentemente em algoritmos de otimização. Assim como a função Rastrigin, ela pertence à classe das funções multimodais, apresentando vários mínimos locais, tornando-se um desafio clássico para algoritmos que buscam o mínimo global.

Definição da Função

A função F6[21] é definida matematicamente pela equação 13 para duas variáveis:

$$f(x, y) = 0.5 - \frac{\sin^2\left(\sqrt{x^2 + y^2}\right) - 0.5}{\left[1 + 0.001 \cdot (x^2 + y^2)\right]^2} \tag{13}$$

Onde:

- x e y são as variáveis de entrada;
- $\sqrt{x^2 + y^2}$ representa a distância euclidiana do ponto (x, y) até a origem;
- O denominador contém um termo de suavização que depende de $x^2 + y^2$.

Características principais

1. Mínimo global:
   - O mínimo global ocorre no ponto $(0, 0)$.
   - O valor da função nesse ponto é $f(0, 0) \approx 1$.

[21] O código para a plotagem desta função encontra-se em 2

2. Mínimos locais:
    - A função possui diversos mínimos locais distribuídos em anéis concêntricos ao redor da origem;
    - Este comportamento cria um desafio significativo para algoritmos de otimização, especialmente os baseados em gradiente.
3. Escalabilidade:
    - Embora definida originalmente para duas variáveis, a função pode ser adaptada para dimensões superiores, mas perde sua característica visual típica.
4. Intervalo de busca:
    - Normalmente, a função é avaliada no intervalo $x, y \in [-100, 100]$.

Função F6 Tridimensional

Em alguns dos realizados para o GADEMO, a função F6 foi considerada em duas variáveis[22]. A figura 137 ilustra o comportamento da função no intervalo definido.

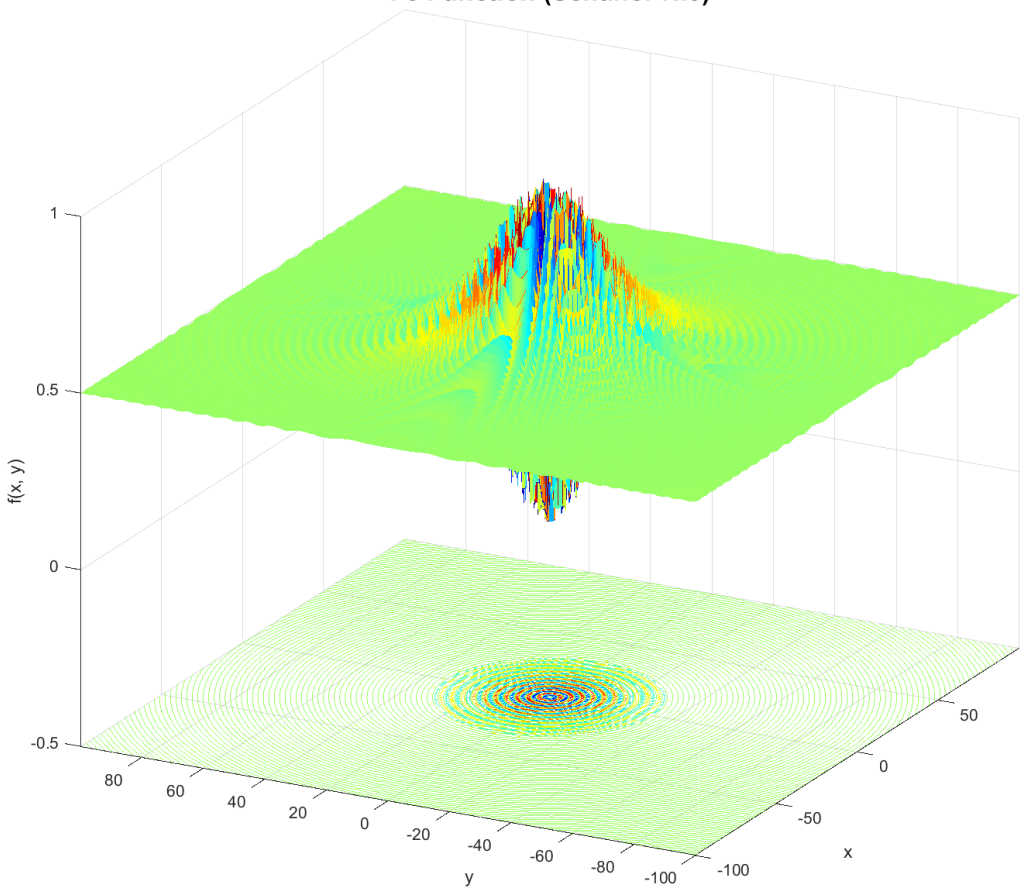


Figure 137: Função F6 em 3 dimensões

Essa função é particularmente interessante para avaliar a consistência de algoritmos genéticos, dado seu padrão de ondulações e a dificuldade em localizar o mínimo global sem cair em mínimos locais.

[22] Essa decisão simplifica a visualização e análise da função em gráficos tridimensionais, permitindo uma avaliação intuitiva dos resultados obtidos pelos algoritmos de otimização.

3 Função Ackley

A **função Ackley** (David H. Ackley, 1987 [79]) é amplamente reconhecida em estudos de otimização e inteligência artificial como um importante *benchmark* para a avaliação de algoritmos de busca. Essa função pertence à classe das funções multimodais, apresentando um grande número de mínimos locais que desafiam algoritmos de otimização a escapar desses pontos e encontrar o mínimo global. A natureza ondulada e simétrica da função a torna uma escolha comum em experimentos que avaliam a capacidade de exploração e exploração de algoritmos genéticos e outras meta-heurísticas.

Definição da Função

A função Ackley é definida pela equação 14 em n variáveis:

$$f(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + \exp(1) \tag{14}$$

Onde:

- $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ é o vetor de variáveis de entrada;
- a, b e c são constantes que determinam o comportamento da função. Os valores típicos são $a = 20$, $b = 0.2$ e $c = 2\pi$;
- n é o número de variáveis.

Características principais

1. Mínimo global:
   - O mínimo global ocorre em $\mathbf{x} = (0, 0, \ldots, 0)$ para todas as variáveis n;
   - O valor da função nesse ponto é $f(0, 0, \ldots, 0) = 0$.
2. Mínimos locais:
   - A função possui diversos mínimos locais que tornam a busca pelo mínimo global desafiadora para algoritmos de otimização;
   - O termo oscilatório $\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right)$ contribui para a formação de um padrão altamente ondulado.
3. Simetria:
   - A função apresenta uma simetria radial em torno do ponto $(0, 0, \ldots, 0)$, característica que pode influenciar algoritmos que dependem de gradientes ou de amostragem uniforme.
4. Intervalo de busca:
   - A função é frequentemente avaliada no intervalo $x_i \in [-32.768, 32.768] \ \forall \ i$.

Função Ackley Tridimensional

Para fins de visualização, a função Ackley será representada em três dimensões, considerando $n = 2$. A função é definida como:

$$f(x, y) = -20 \exp\left(-0.2\sqrt{0.5(x^2 + y^2)}\right) - \exp\left(0.5(\cos(2\pi x) + \cos(2\pi y))\right) + 20 + \exp(1) \tag{15}$$

A função Ackley tridimensional pode ser visualizada na figura 138, destacando sua complexa topologia de vales e picos.

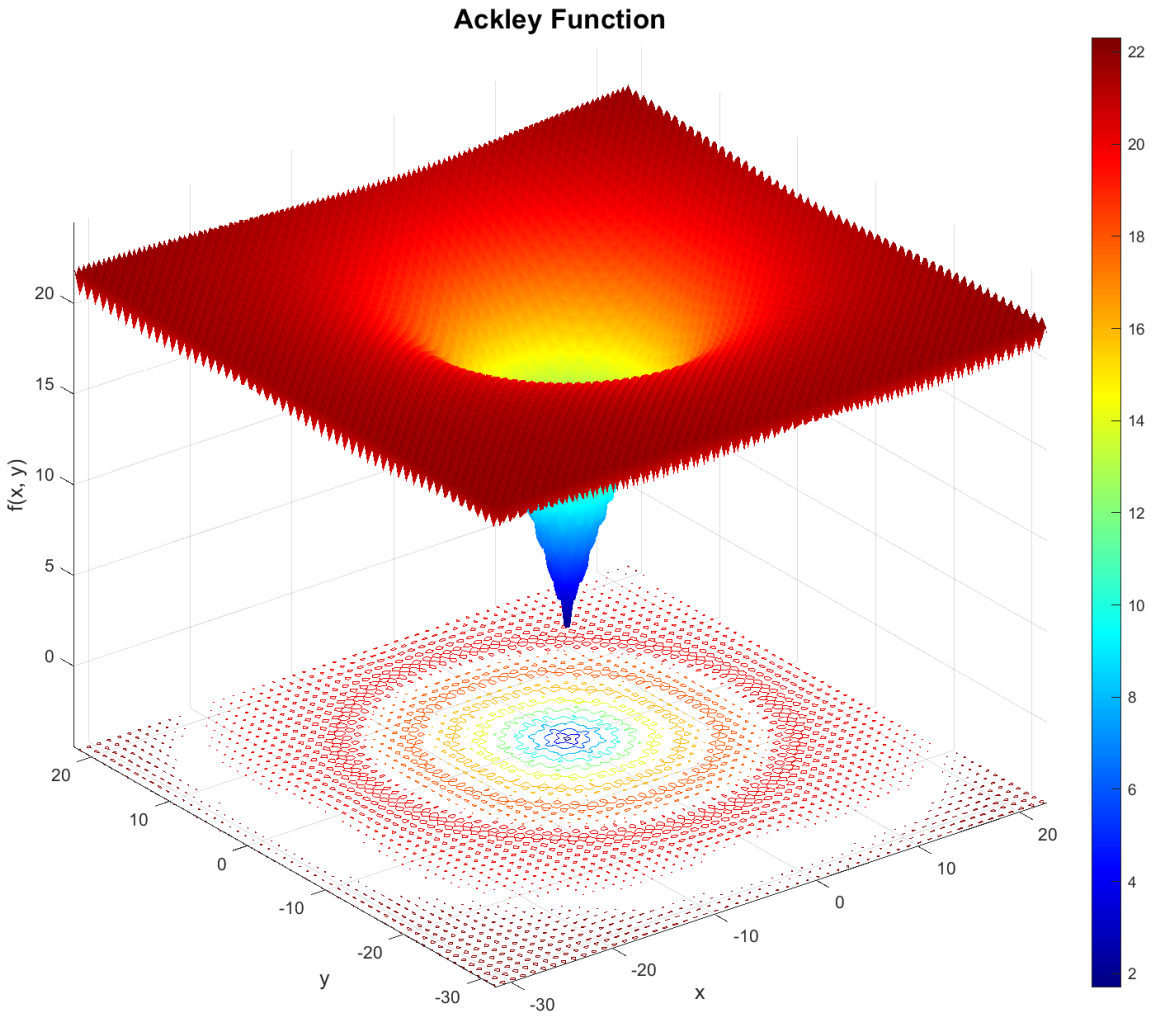


Figure 138: Função Ackley com 3 dimensões

4 Função Levy

A **função Levy** (Jean-Paul Levy & Michel Montalvo, 1985 [80]) é amplamente utilizada em estudos de otimização global e meta-heurísticas, sendo reconhecida como um desafio devido à sua natureza multimodal e à presença de muitos mínimos locais. Sua estrutura complexa e padrões altamente não-lineares tornam a função uma escolha popular para avaliar a capacidade de exploração e convergência de algoritmos de busca, como algoritmos genéticos, partículas e enxames.

Definição da Função

A função Levy é definida pela equação 16 em n variáveis:

$$f(\mathbf{x}) = \sin^2(\pi w_1) + \sum_{i=1}^{n-1} (w_i - 1)^2 \left[1 + 10\sin^2(\pi w_i + 1)\right] + (w_n - 1)^2 \left[1 + \sin^2(2\pi w_n)\right] \quad (16)$$

Onde:

- $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ é o vetor de variáveis de entrada;
- $w_i = 1 + \frac{x_i - 1}{4}$ para cada $i = 1, \ldots, n$;
- n é o número de variáveis.

Características principais

1. Mínimo global:
   - O mínimo global ocorre em $\mathbf{x} = (1, 1, \ldots, 1)$ para todas as variáveis n;
   - O valor da função nesse ponto é $f(1, 1, \ldots, 1) = 0$.
2. Mínimos locais:
   - A função possui diversos mínimos locais que dificultam a busca pelo mínimo global, especialmente em altas dimensões;
   - Os termos oscilatórios envolvendo $\sin^2$ criam um padrão ondulado que desafia algoritmos de busca a escapar de mínimos locais.
3. Complexidade:
   - A função combina características de amplas áreas planas, vales profundos e picos estreitos, resultando em um cenário de otimização altamente complexo.
4. Intervalo de busca:
   - A função é frequentemente avaliada no intervalo $x_i \in [-10, 10] \ \forall \ i$.

Função Levy Tridimensional

Para fins de visualização[23], a função Levy será representada em três dimensões, considerando $n = 2$. A função é definida como:

$$f(x, y) = \sin^2(\pi w_1) + (w_1 - 1)^2 \left[1 + 10 \sin^2(\pi w_1 + 1)\right] + (w_2 - 1)^2 \left[1 + \sin^2(2\pi w_2)\right] \quad (17)$$

Onde $w_1 = 1 + \frac{x-1}{4}$ e $w_2 = 1 + \frac{y-1}{4}$. A figura 139 apresenta a função Levy tridimensional, destacando sua complexidade e estrutura ondulada, característica que a torna um teste essencial para algoritmos de otimização.

[23] Código MATLAB para plotagem encontra-se em 4

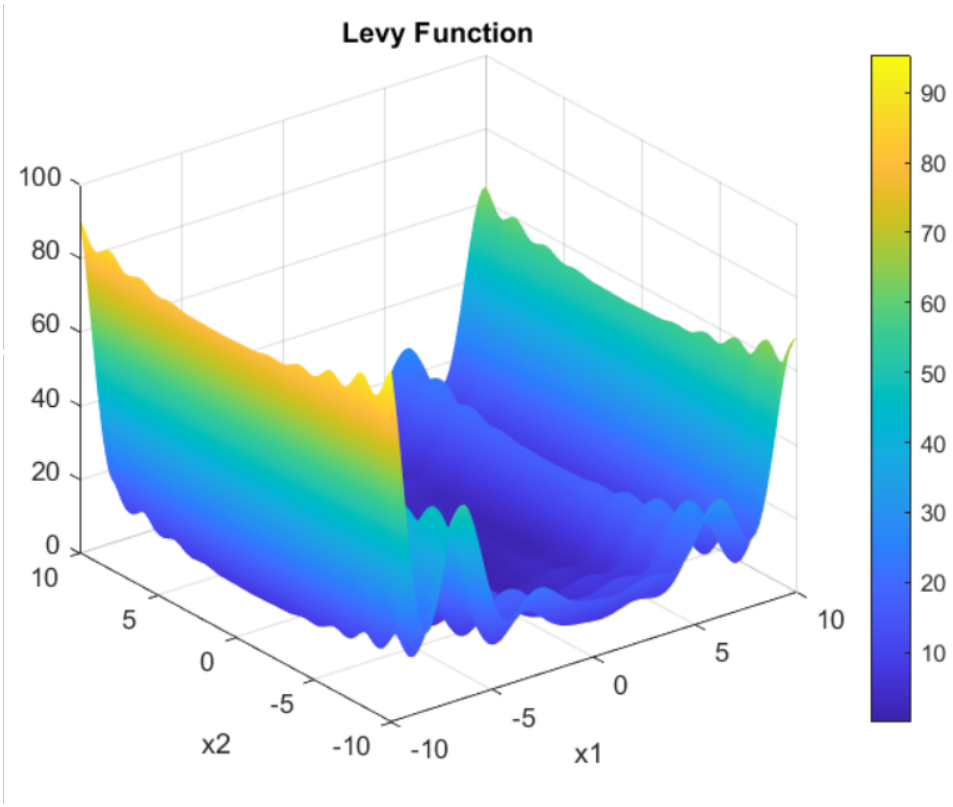


Figure 139: Função Levy com 3 dimensões

5 Função Drop-Wave

A **função Drop-Wave** é amplamente reconhecida em estudos de otimização como um dos desafios clássicos para algoritmos de busca global (El-Ghazali Talbi, 2009 [81]). Sua estrutura caracteriza-se por ondas concêntricas que convergem para um único mínimo global no centro do domínio. Essa função é utilizada para avaliar a capacidade de exploração e a habilidade de escape de mínimos locais de algoritmos de otimização.

Definição da Função

A função Drop-Wave é definida pela equação 18 em duas variáveis:

$$f(x,y) = -\frac{1+\cos(12\sqrt{x^2+y^2})}{0.5(x^2+y^2)+2} \tag{18}$$

Onde:

- x e y são as variáveis de entrada, que determinam a posição no domínio da função;
- A função combina termos oscilatórios e uma diminuição radial em direção ao centro, criando uma topologia altamente multimodal.

Características principais

1. Mínimo global:
   - O mínimo global ocorre em $(x,y) = (0,0)$;
   - O valor da função nesse ponto é $f(0,0) = -1$.
2. Mínimos locais:

- A função apresenta diversos mínimos locais dispostos em anéis concêntricos devido ao termo $\cos(12\sqrt{x^2+y^2})$;
- Essa característica dificulta a convergência de algoritmos de otimização para o mínimo global.

3. Simetria:
   - A função é simétrica em relação ao ponto $(0,0)$, o que pode ser explorado por algoritmos que utilizam estratégias de busca baseadas em gradientes ou amostragens uniformes.
4. Intervalo de busca:
   - A função é geralmente avaliada no intervalo $x, y \in [-5.12, 5.12]$, cobrindo uma ampla área de interesse.

Função Drop-Wave Tridimensional

Para fins de visualização[24], a função Drop-Wave será representada em três dimensões, considerando o intervalo $x, y \in [-5.12, 5.12]$. A figura 140 apresenta a representação tridimensional da função, evidenciando suas ondulações concêntricas e o mínimo global em $(0,0)$.

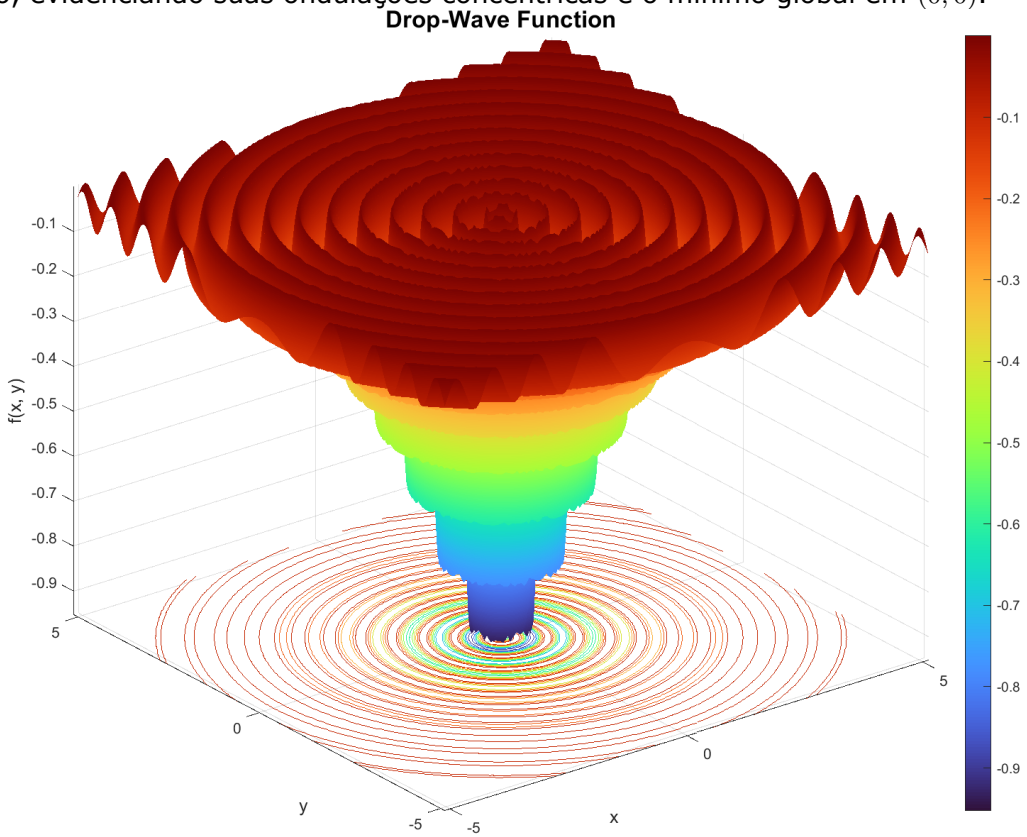


Figure 140: Função Drop-Wave com 3 dimensões

[24] Código para plotagem em 5

b Apêndice B - Códigos MATLAB Para Plotagem de Funções

1 Função Rastrigin

```
% RASTRIGIN FUNCTION
% Author: Paloma Sette
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x = linspace(-5.12,5.12,200);
y = linspace(-5.12,5.12,200);

f = @(x,y) 10*2 + x.^2 + y.^2 - 10*cos(2*pi*x) - 10*cos(2*pi*y);

[xx,yy] = meshgrid(x,y);

zz = reshape(f(xx(:),yy(:)), 200, 200);

figure
surfc(xx,yy,zz);
shading interp;

title('Rastrigin function','FontSize',16)
axis([-5.12 5.12 -5.12 5.12 -30 85])
```

2 Schaffer Function N.6

```
% SCHAFFER FUNCTION N. 6
% Author: Paloma Sette
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x = linspace(-100, 100, 200);
y = linspace(-100, 100, 200);

f6 = @(x, y) 0.5 - (sin(sqrt(x.^2 + y.^2)).^2 - 0.5) ./ (1 + 0.001*(x.^2 + y.^2)).^2;

[xx, yy] = meshgrid(x, y);

zz = reshape(f6(xx(:), yy(:)), 200, 200);

% Plota a superfície com curvas de nível bem definidas
figure
surfc(xx, yy, zz);
colormap('jet');
shading flat;

% Adiciona estética ao gráfico
title('F6 Function (Schaffer N.6)','FontSize',16)
xlabel('x');
ylabel('y');
zlabel('f(x, y)');
axis([-100 100 -100 100 -0.5 1]); % Define os limites dos eixos
grid on;
```

3 Função Ackley

```
% ACKLEY FUNCTION
% Author: Paloma Sette
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x = linspace(-32.768, 32.768, 200);
y = linspace(-32.768, 32.768, 200);
```

```
    a = 20;
    b = 0.2;
    c = 2 * pi;
    f = @(x, y) -a * exp(-b * sqrt(0.5 * (x.^2 + y.^2))) ...
        - exp(0.5 * (cos(c * x) + cos(c * y))) + a + exp(1);

    [xx, yy] = meshgrid(x, y);

    zz = f(xx, yy);

    % Plota a função Ackley em 3D com curva de nível
    figure
    surfc(xx, yy, zz);
    shading interp;
    colormap(jet);

    % Configurações do gráfico
    title('Ackley Function','FontSize',16)
    xlabel('x','FontSize',12)
    ylabel('y','FontSize',12)
    zlabel('f(x, y)','FontSize',12)
    axis([-32.768 32.768 -32.768 32.768 -10 25])
    colorbar; % Adiciona uma barra de cores para representar os valores
```

4 Função Levy

```
% LEVY FUNCTION
% Author: Paloma Sette
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x = linspace(-10, 10, 200);
y = linspace(-10, 10, 200);

f = @(x, y) sin(pi .* (1 + ((x - 1) ./ 4))).^2 + ...
    ((x - 1).^2 .* (1 + 10 .* sin(pi .* (1 + ((x - 1) ./ 4)) + 1).^2)) + ...
    ((y - 1).^2 .* (1 + sin(2 .* pi .* (1 + ((y - 1) ./ 4))).^2));

[xx, yy] = meshgrid(x, y);

zz = f(xx, yy);

figure
surfc(xx, yy, zz);
shading interp;
colormap(parula);

title('Levy Function','FontSize',16)
xlabel('x','FontSize',12)
ylabel('y','FontSize',12)
zlabel('f(x, y)','FontSize',12)
axis([-10 10 -10 10 0 250])
colorbar; % Adiciona uma barra de cores para representar os valores
```

5 Função Drop-Wave

```
% DROP-WAVE FUNCTION
% Author: Paloma Sette
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
x = linspace(-5.12, 5.12, 200);
y = linspace(-5.12, 5.12, 200);

f = @(x, y) -(1 + cos(12 * sqrt(x.^2 + y.^2))) ./ (0.5 * (x.^2 + y.^2) + 2);

[xx, yy] = meshgrid(x, y);

zz = f(xx, yy);

figure;

surfc(xx, yy, zz);

shading interp;
colormap('turbo');
colorbar;

title('Drop-Wave Function', 'FontSize', 16);
xlabel('x', 'FontSize', 12);
ylabel('y', 'FontSize', 12);
zlabel('f(x, y)', 'FontSize', 12);
axis tight;
grid on;
```

References

[1] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1998.

[2] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, USA: Addison-Wesley, 1989.

[3] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 2nd ed., ser. Natural Computing Series. Springer, 2015.

[4] J. Zhang, K. Liu, Y. Tan, and X. He, "Recent advances in the industrial applications of genetic algorithms," *IEEE Transactions on Industrial Applications*, vol. 57, no. 4, pp. 1345–1358, 2021.

[5] S. Kumar and A. Verma, "Teaching genetic algorithms through interactive tools: Challenges and opportunities," *International Journal of Computer Science Education*, vol. 10, no. 2, pp. 45–62, 2019.

[6] M. R. Silva and P. E. Santos, "Interactive tools for teaching evolutionary computation: A systematic review," *Brazilian Journal of Computers in Education*, vol. 28, no. 2, pp. 78–95, 2020.

[7] X. Li and Y. Wu, "Design principles for educational software in computer science," *Journal of Computing Education*, vol. 8, no. 3, pp. 112–128, 2022.

[8] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, 2000, thesis that introduced REST architectural style.

[9] C. Rodriguez, M. Baez, F. Daniel, F. Casati, J. C. Trabucco, L. Canali, and G. Percannella, "Rest apis: A large-scale analysis of compliance with principles and best practices," *International Conference on Web Engineering*, pp. 21–39, 2016.

[10] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed. O'Reilly Media, 2021.

[11] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: A state-of-the-art review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, 2019.

[12] P. Santos, M. Carvalho, and O. Belo, "Comparing cloud platform alternatives for microservices deployments," in *International Conference on Information Technology and Systems*. Springer, 2020, pp. 283–292.

[13] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 1st ed. Ann Arbor: The University of Michigan Press, 1975.

[14] G. Van Rossum and F. L. Drake Jr, "The python language reference manual," 2011.

[15] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "Deap: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 2171–2175, 2012.

[16] C. Severance, "Javascript: Designing a language in 10 days," *Computer*, vol. 45, no. 2, pp. 7–8, 2012.

[17] E. Marcotte, "Responsive web design: Advanced techniques," in *International Conference on Web Engineering*. Springer, 2017, pp. 510–511.

[18] C. Darwin, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, 1st ed. London, UK: John Murray, 1859, reimpresso em várias edições e traduções desde então, também conhecido como "A Origem das Espécies".

[19] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, ser. Problemata (Stuttgart). Frommann-Holzboog, 1973. [Online]. Available: https://books.google.com.br/books?id=-WAQAQAAMAAJ

[20] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. London, UK: Springer-Verlag, 1996.

[21] R. Linden, *Algoritmos Genéticos: Uma Importante Ferramenta da Inteligência Computacional*. Rio de Janeiro, RJ: Editora Brasport, 2006.

[22] A. Barboza, "Simulação e técnicas da computação evolucionária aplicadas a problemas de programação linear inteira mista," Ph.D. dissertation, Universidade Tecnológica Federal do Paraná (UTFPR), CPGEl, Curitiba, PR, 2005.

[23] H. Aytug, M. Khouja, and F. Vergara, "A review of the use of genetic algorithms to solve production and operations management problems," *International Journal of Production Research*, vol. 41, no. 17, pp. 3955–4009, 2003.

[24] L. Almeida, Y. Valdivia, M. Vellasco, and M. Pacheco, "Otimização de alternativas para o desenvolvimento de campos de petróleo," *Gestão & Produção*, vol. 14, no. 3, pp. 489–503, 2007.

[25] R. Concilio, "Contribuições à solução de problemas de escalonamento pela aplicação conjunta de computação evolutiva e otimização com restrições," Master's thesis, Universidade Estadual de Campinas (UNICAMP), Campinas, SP, 2000.

[26] G. Soares, "Algoritmos genéticos: Estudo, novas técnicas e aplicações," Master's thesis, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, MG, 1997.

[27] L. Davis, *Handbook of Genetic Algorithms*. New York, USA: Van Reinhold Nostrand, 1991.

[28] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, USA: MIT Press, 1996.

[29] M. Soares, "Análise do uso de algoritmos genéticos na otimização do planejamento mestre da produção," Master's thesis, Pontifícia Universidade Católica do Paraná (PUC-PR), Curitiba, PR, 2006.

[30] C. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*. New York, USA: McGraw-Hill, 1995.

[31] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[32] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2010.

[33] S. Torabi, S. Ghomi, and B. Karimi, "A hybrid genetic algorithm for the finite horizon economic lot and delivery scheduling in supply chains," *European Journal of Operational Research*, vol. 173, no. 1, pp. 173–189, 2006.

[34] J. Koza, M. A. Keane, and M. J. Streeter, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Hingham, USA: Kluwer Academic Publishers, 2003.

[35] T. Bäck, D. Fogel, and Z. Michalewicz, *Evolutionary Computation 2: Advanced Algorithms and Operators*. London: IOP Publishing, 2000.

[36] C. Hicks, "A genetic algorithm tool for optimising cellular or functional layouts in the capital goods industry," *International Journal of Production Economics*, vol. 104, no. 2, pp. 598–614, 2006.

[37] M. Falcone, "Estudo comparativo entre algoritmos genéticos e evolução diferencial para otimização de um modelo de cadeia de suprimento simplificada," Master's thesis, Pontifícia Universidade Católica do Paraná (PUC-PR), Curitiba, PR, 2004.

[38] M. Teles and H. Gomes, "Comparação de algoritmos genéticos e programação quadrática sequencial para otimização de problemas em engenharia," *Teoria e Prática na Engenharia Civil*, vol. 10, no. 15, pp. 29–39, 2010.

[39] E. Bento and N. Kagan, "Algoritmos genéticos e variantes na solução de problemas de configuração de redes de distribuição," *Revista Controle & Automação*, vol. 19, no. 3, pp. 302–315, 2008.

[40] M. A. C. Pacheco, "Algoritmos genéticos: Princípios e aplicações," Rua Marques de São Vicente 225, Gávea, CEP 22453-900, Rio de Janeiro, RJ, Brasil, 2004.

[41] F. Azadivar and G. Tompkins, "Simulation optimization with qualitative variables and structural model changes: A genetic algorithm approach," *European Journal of Operational Research*, vol. 113, pp. 169–182, 1999.

[42] A. Konak, D. Coit, and A. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering and System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[43] K. A. D. Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, Ann Arbor, MI, 1975, ph.D. Thesis.

[44] T. Yang, Y. Kuo, and C. Cho, "A genetic algorithms simulation approach for the multi-attribute combinatorial dispatching decision problem," *European Journal of Operational Research*, vol. 176, no. 3, pp. 1859–1873, 2007.

[45] R. Neppalli, C. Chen, and J. Gupta, "Genetic algorithms for the two-stage bicriteria flow shop problem," *European Journal of Operational Research*, vol. 95, no. 2, pp. 356–373, 1996.

[46] Y. Yun and M. Gen, "Performance analysis of adaptive genetic algorithms with fuzzy logic and heuristics," *Fuzzy Optimization and Decision Making*, vol. 2, no. 2, pp. 161–175, 2003.

[47] J. Tanomaru, "Motivação, fundamentos e aplicações de algoritmos genéticos," in *Anais do II Congresso Brasileiro de Redes Neurais*. Curitiba, PR: Editora desconhecida, 1995, pp. 373–403.

[48] E. Lacerda and A. C. P. L. F. Carvalho, "Introdução aos algoritmos genéticos," in *Sistemas Inteligentes: Aplicações a Recursos Hídricos e Ciências Ambientais*. Porto Alegre, RS: Editora da Universidade Federal do Rio Grande do Sul, 1999, vol. 1, pp. 99–148.

[49] S. Hwang and R. He, "A hybrid real-parameter genetic algorithm for function optimization," *Advanced Engineering Informatics*, vol. 20, no. 1, pp. 7–21, 2006.

[50] P. Pendharkar, "The theory and experiments of designing cooperative intelligent systems," *Decision Support Systems*, vol. 43, pp. 1014–1030, 2007.

[51] T. James, R. Barkhi, and J. Johnson, "Platform impact on performance of parallel genetic algorithms: Design and implementation considerations," *Engineering Applications of Artificial Intelligence*, vol. 19, pp. 843–856, 2006.

[52] L. Wen, L. Gao, X. Li, and H. Li, "A new genetic algorithm based evolutionary neural architecture search for image classification," *Swarm and Evolutionary Computation*, vol. 75, 2022.

[53] F. Concatto, W. Zunino, L. A. Giancoli, R. Santiago, and L. C. Lamb, "Genetic algorithm for epidemic mitigation by removing relationships," *arXiv preprint arXiv:1707.05377*, 2017.

[54] A. d. Silva and T. Ludermir, "Otimização de redes neurais através de algoritmos genéticos celulares," *arXiv preprint arXiv:2107.08326*, 2021.

[55] L. F. R. Moreira, "Aplicação de algoritmo genético em problemas de engenharia logística," 2021.

[56] K. De Jong, *Evolutionary computation: a unified approach*. MIT press, 2016.

[57] M. Richards, *Software architecture patterns*. O'Reilly Media, 2015.

[58] J. J. Garrett, *The elements of user experience: user-centered design for the web and beyond*. New Riders, 2010.

[59] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[60] S. Newman, *Building microservices: designing fine-grained systems*. O'Reilly Media, 2021.

[61] T. Back, D. B. Fogel, and Z. Michalewicz, *Handbook of evolutionary computation*. CRC Press, 1997.

[62] C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007.

[63] E. Freeman and E. Freeman, *Use a cabeça! Padrões de projetos (Design Patterns)*, ser. Use a Cabeça! Alta Books, 2007. [Online]. Available: https://books.google.com.br/books?id=VRSPPgAACAAJ

[64] T. E. Oliphant, *Guide to NumPy*. USA: Trelgol Publishing, 2006.

[65] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, 2008.

[66] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "Deap: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, 2012.

[67] O. Čertík *et al.*, "Sympy: Symbolic mathematics in python," Online, 2006, available at: https://www.sympy.org.

[68] T. Christie, "Uvicorn: Lightning-fast asgi server," Online, 2018, available at: https://www.uvicorn.org.

[69] S. Ramírez, "Fastapi: Modern web framework for apis with python," Online, 2018, available at: https://fastapi.tiangolo.com.

[70] A. Ronacher, "Flask: Microframework for python," Online, 2010, available at: https://flask.palletsprojects.com.

[71] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, 1st ed. Boston, MA: Addison-Wesley, 1999.

[72] D. M. Hawkins, *Identification of Outliers*. London: Chapman and Hall, 1980.

[73] D. C. Montgomery and G. C. Runger, *Applied Statistics and Probability for Engineers*, 6th ed. Wiley, 2014.

[74] M. H. Kutner, C. J. Nachtsheim, and J. Neter, *Applied Linear Regression Models*, 4th ed. McGraw-Hill Education, 2004.

[75] J. W. Tukey, *Exploratory Data Analysis*. Reading, MA: Addison-Wesley, 1977, popularized the use of the IQR and box plots in data analysis.

[76] A. E. Eiben and J. E. Smith, "Introduction to evolutionary computing," *Natural Computing Series*, 2003.

[77] L. A. Rastrigin, *Systems of Extremal Control*. Moscow: Nauka, 1974.

[78] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proceedings of the International Conference on Genetic Algorithms*, 1989, pp. 93–100.

[79] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, 1987.

[80] J.-P. Levy and M. Montalvo, "Exact methods for numerical optimization of complex systems," *Journal of Optimization Theory and Applications*, vol. 46, no. 4, pp. 571–590, 1985.

[81] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. Hoboken, NJ: Wiley, 2009.