Centro Federal de Educação Tecnológica de Minas Gerais
PROFMAT - Mestrado Profissional em Matemática em Rede Nacional

Lucas Amaral Tavares

Algoritmos Evolutivos e Gamificação: Uma Proposta de Atividade para a Educação Básica

Belo Horizonte
2023

Lucas Amaral Tavares

ALGORITMOS EVOLUTIVOS E GAMIFICAÇÃO: UMA PROPOSTA DE ATIVIDADE PARA A EDUCAÇÃO BÁSICA

Dissertação apresentada ao Centro Federal de Educação Tecnológica de Minas Gerais como parte das exigências do Programa de Pós-Graduação Mestrado Profissional em Matemática em Rede Nacional, para obter o título de Mestre.

Orientador
Dênis Emanuel da Costa Vargas

Coorientação
Jônathas Douglas Santos de Oliveira

Banca Examinadora
Afonso Celso de Castro Lemonge
Elizabeth Fialho Wanner
Luis Alberto D'Afonseca

Belo Horizonte
2023

Tavares, Lucas Amaral
T231a Algoritmos evolutivos e gamificação: uma proposta de atividade para a educação básica / Lucas Amaral Tavares. – 2023.
142 f.

Dissertação de mestrado apresentada ao Programa de Mestrado Profissional em Matemática em Rede Nacional.
Orientador: Dênis Emanuel da Costa Vargas.
Coordenador: Jônathas Douglas Santos de Oliveira.
Dissertação (mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais.

1. Educação básica – Teses. 2. Computação – Teses. 3. Otimização – Teses. 4. Algoritmos evolutivos – Teses. 5. Gamificação – Teses. I. Vargas, Dênis Emanuel da Costa. II. Oliveira, Jônathas Douglas Santos de. III. Centro Federal de Educação Tecnológica de Minas Gerais. IV. Título.

CDD 371.33

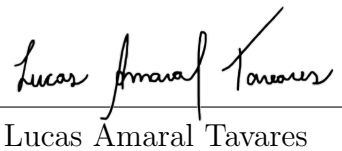
Elaboração da ficha catalográfica pela bibliotecária Jane Marangon Duarte, CRB 6º 1592 / Cefet/MG

Lucas Amaral Tavares

ALGORITMOS EVOLUTIVOS E GAMIFICAÇÃO: UMA PROPOSTA DE ATIVIDADE PARA A EDUCAÇÃO BÁSICA

Dissertação apresentada ao Centro Federal de Educação Tecnológica de Minas Gerais como parte das exigências do Programa de Pós-Graduação Mestrado Profissional em Matemática em Rede Nacional, para obter o título de Mestre.

Aprovada: 23 de agosto de 2023.



Lucas Amaral Tavares
(Autor)

Dênis Emanuel da Costa Vargas
(Orientador)

Belo Horizonte
2023

Agradecimentos

A Deus, pelas incontáveis graças, oportunidades e desafios a mim confiados.

Ao meu orientador, Dênis, pela paciência, pelo auxílio, pela motivação constante e por colaborar sempre para a qualidade do trabalho, em face de todas as dificuldades que tivemos ao longo deste tempo. Muitas vezes, foi a boa vontade dele que fez nosso trabalho avançar. Ao meu coorientador, Jonathas, pela dedicação e apoio ao nosso trabalho.

Aos meus pais, Cleusa e Márcio, e a minha irmã Carol, pelo apoio incondicional, pelos valores transmitidos e por todo esforço que sempre fizeram pela minha formação.

A minha namorada, Thais, pelo amor e compreensão, e por me dar as mãos nos dias mais difíceis. Obrigado por acreditar em mim e estar ao meu lado.

Aos professores e colegas de mestrado, com os quais dividi bons momentos e aprendi muito do que hoje posso dividir em sala de aula.

Aos meus familiares, amigos e colegas de trabalho, pela consideração e incentivo constante, em especial aos meus avós Alpha (*in memoriam*) e Afonso (*in memoriam*) que sempre foram fonte de inspiração e aprendizado.

A todos que tem me apoiado na busca dos meus sonhos que aos poucos se tornam realidade, especialmente aos que me incentivaram ao longo de minha trajetória.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Resumo

A Base Nacional Comum Curricular (BNCC) menciona a importância dos algoritmos no desenvolvimento do Pensamento Computacional com estudantes da Educação Básica, além de salientar que eles podem ser objetos de estudo nas aulas de Matemática. Duas habilidades da BNCC para o Ensino Médio estão diretamente relacionadas ao Pensamento Computacional: EM13MAT315, que propõe investigar um algoritmo que resolve um problema, e EM13MAT405, que propõe utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos. Entre as possibilidades de se trabalhar essas habilidades estão os Algoritmos Evolutivos (AEs), os quais são baseados em mecanismos da evolução biológica que resolvem problemas de otimização. Apesar desses problemas serem discutidos frequentemente em dissertações do PROFMAT, o uso de AEs ainda é pouco explorado. Nesta dissertação apresenta-se aspectos teóricos de otimização e AEs, além de uma revisão da literatura. A maior contribuição deste trabalho é um produto educacional onde AEs são usados para resolver problemas de otimização em diversos contextos, implementados na linguagem de programação Python e trabalhados através de sequências didáticas sob a óptica do Pensamento Computacional e da perspectiva da gamificação. A proposta inclui a organização de uma competição de AEs, inspirada em Maratonas de Programação, com foco nos Algoritmos Genéticos (AG) e na Evolução Diferencial (ED), visando aprimorar as soluções obtidas pelos competidores. Uma simulação de competição é realizada em que os resultados obtidos são discutidos. A ideia é fazer com que os competidores consigam desenvolver variações do AG e da ED que obtenham soluções cada vez mais próximas do ótimo dos problemas. A utilização de AEs para resolver problemas de otimização se mostrou promissora como uma excelente oportunidade para o desenvolvimento do Pensamento Computacional pelos estudantes do Ensino Médio. O presente material visa tanto oferecer ao professor um embasamento teórico consistente e ao mesmo tempo de fácil acesso, além dos próprios trechos de códigos já implementados para facilitar a replicação.

Palavras-chave: Educação Básica. Pensamento Computacional. Otimização. Algoritmos Evolutivos. Gamificação.

Abstract

The Brazilian National Common Curricular Base (BNCC) highlights the significance of algorithms in fostering Computational Thinking among students in Basic Education while emphasizing their potential as study subjects within Mathematics classes. Two specific skills outlined in the BNCC for High School education are directly linked to Computational Thinking: EM13MAT315, which involves investigating algorithms to solve problems, and EM13MAT405, which focuses on utilizing basic concepts of a programming language for algorithm implementation. Among the various possibilities for incorporating these skills, Evolutionary Algorithms (EAs) stand out. EAs employ techniques inspired by biological evolution to solve optimization problems. Despite being a frequently discussed topic in PROFMAT dissertations, the use of EAs remains relatively unexplored. This dissertation addresses theoretical aspects of optimization and EAs while also providing a literature review. The primary contribution of this work is an educational product that employs EAs to solve optimization problems across different contexts. The product uses Python and adopts a pedagogical approach that integrates Computational Thinking and gamification. The proposed educational product includes an EA competition inspired by Programming Contests focused on Genetic Algorithms (GA) and Differential Evolution (DE). The competition aims to enhance participants' problem-solving solutions by developing variations of GA and DE algorithms to achieve solutions that increasingly approximate the optimal solution. The results obtained with a performed simulation are analyzed and discussed. Solving optimization problems with EAs demonstrates a remarkable potential for fostering Computational Thinking among students in Secondary School. This material equips teachers with a solid theoretical foundation and provides effortless access to pre-implemented code snippets, simplifying replication.

Keywords: Basic education. Computational Thinking. Optimization. Evolutionary Algorithms. Gamification.

Sumário

1 Introdução

Pensamento Computacional é uma estratégia usada para solucionar problemas que se baseia, entre outros, na decomposição (dividir um problema complexo em pequenas partes) e no uso de algoritmos para sua solução. Algoritmos são sequências de passos executadas com o intuito de se resolver um problema. A ideia é reformular problemas que aparentam ser de difícil resolução analítica e transformá-los em algo capaz de ser compreendido, desejavelmente tendo a tecnologia como alicerce. Assim, além do pensamento algorítmico e da decomposição, o pensamento computacional também se baseia nas habilidades de abstração e reconhecimento de padrões [1].

A Base Nacional Comum Curricular (BNCC) [2] menciona a importância dos algoritmos no desenvolvimento do Pensamento Computacional dos estudantes da Educação Básica, além de salientar que eles podem ser objetos de estudo nas aulas de Matemática:

> Associado ao pensamento computacional, cumpre salientar a importância dos algoritmos e de seus fluxogramas, que podem ser objetos de estudo nas aulas de Matemática. Um algoritmo é uma sequência finita de procedimentos que permite resolver um determinado problema. Assim, o algoritmo é a decomposição de um procedimento complexo em suas partes mais simples, relacionando-as e ordenando-as, e pode ser representado graficamente por um fluxograma. (BNCC, 2018, p. 271).

É importante ressaltar que, segundo a Lei de Diretrizes e Bases da Educação Nacional [3], a Educação Básica compreende a Pré-Escola, o Ensino Fundamental e o Ensino Médio. Duas das habilidades da BNCC para a Matemática do Ensino Médio que estão diretamente relacionadas ao Pensamento Computacional são:

EM13MAT315, que propõe investigar e registrar, por meio de um fluxograma, quando possível, um algoritmo que resolve um problema, e

EM13MAT405, que propõe utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

Entre as possibilidades de se trabalhar algoritmos no Ensino Médio estão os Algoritmos Evolutivos (AEs), os quais são baseados em mecanismos da evolução biológica no intuito de resolver problemas de otimização. Tais problemas são discutidos frequentemente no contexto do Ensino Médio em dissertações do PROFMAT, como, por exemplo, Corrêa[4] e Nunes[5], os quais utilizam métodos determinísticos para resolução desses problemas.

Dados do Repositório de Dissertações do PROFMAT revelam uma escassez de trabalhos no contexto do programa que utilizam os AEs na resolução de problemas de otimização, o que traz um aspecto inovador à presente proposta. Ressalta-se ainda que o caráter interdisciplinar ao se utilizar AEs é potencializado, dado que eles podem traduzir elementos associados aos fenômenos que caracterizam à Teoria da Evolução para uma linguagem algorítmica. Nesse sentido, as atividades podem ser conectadas às aulas de biologia e química.

Uma utilização recorrente de algoritmos na resolução de problemas diversos se encontra na área de otimização. De forma simplificada, um problema de otimização consiste em obter os valores das variáveis independentes que minimizam uma função de custo dependente destas mesmas variáveis. Nesses problemas, as variáveis podem assumir uma representação real, inteira, binária ou mista.

A otimização é uma área da matemática com aplicações em problemas de engenharia, estatística, logística, biologia, economia, dentre outras [6, 7]. Métodos diversos podem ser utilizados no intuito de se resolver um problema de otimização que vão desde métodos baseados em derivadas até estratégias que envolvem populações. Nesse contexto, dentre as estratégias computacionais bem consolidadas despontam os AEs. Tais algoritmos são baseados em populações, isto é, existe um conjunto de vetores que codificam de alguma forma as variáveis independentes do problema. Exemplos bem conhecidos de AEs são os Algoritmos Genéticos (AGs) e a Evolução Diferencial (ED). Em AG, por exemplo, cada indivíduo da população está associado a uma função de custo que se deseja otimizar e recebe o nome de *fitness* (do inglês, adaptação). Essa população passa por processos de atualização (trocas de informações entre os indivíduos da população e variações individuais), que podem envolver cruzamentos, mutações e seleções [8]. De modo geral, cada uma dessas operações deve ser realizada considerando-se uma probabilidade de ocorrência, de forma que tais algoritmos estão intimamente associados a aspectos aleatórios. A execução de uma sequência dessas operações que foram mencionadas compõem uma geração ou iteração. O

algoritmo repete tal processo durante um certo número de gerações até que algum critério de parada seja satisfeito. Assim, o indivíduo com melhor valor da função fitness obtido durante o processo de busca é aquele que o AE considera como solução do problema de otimização. Tais abordagens computacionais são classificadas como heurísticas, ou seja, não existem garantias matemáticas de que a melhor solução encontrada pelo algoritmo seja a solução ótima do problema. Por outro lado, diferentemente de métodos determinísticos tradicionais, AEs não dependem de informações adicionais associadas às propriedades matemáticas da função de custo, como, por exemplo, suas derivadas.

Pelo fato de apresentarem caráter aleatório, os AEs também se constituem como oportunidade de se abordar conceitos de Estatística e Probabilidade em sala de aula. Como algumas operações, por exemplo, as mutações e as seleções, apresentam aspectos de aleatoriedade de modo intrínseco, as seguintes habilidades da BNCC se relacionam aos AEs:

EM13MAT406: Construir e interpretar tabelas e gráficos de frequências com base em dados obtidos em pesquisas por amostras estatísticas, incluindo ou não o uso de softwares que inter-relacionem estatística, geometria e álgebra.

EM13MAT407: Interpretar e comparar conjuntos de dados estatísticos por meio de diferentes diagramas e gráficos (histograma, de caixa (box-plot), de ramos e folhas, entre outros), reconhecendo os mais eficientes para sua análise.

EM13MAT311: Identificar e descrever o espaço amostral de eventos aleatórios, realizando contagem das possibilidades, para resolver e elaborar problemas que envolvem o cálculo da probabilidade.

EM13MAT316: Resolver e elaborar problemas, em diferentes contextos, que envolvem cálculo e interpretação das medidas de tendência central (média, moda, mediana) e das medidas de dispersão (amplitude, variância e desvio padrão).

Para implementação de AEs em um contexto computacional, utiliza-se alguma linguagem de programação. Dentre as diversas existentes, a linguagem Python desponta como uma das mais populares atualmente. Python é uma linguagem de programação de alto nível, interpretada e orientada a objetos, tendo sido criada no final dos anos 80 por Guido van Rossum. Tal linguagem apresenta uma sintaxe clara e concisa, o

que facilita o aprendizado e a utilização para iniciantes. Ademais, possui uma vasta biblioteca padrão que oferece funcionalidades que vão desde a manipulação de strings até a criação de interfaces gráficas. Outra vantagem é a sua portabilidade, pelo fato de ser compatível com diversos sistemas operacionais, como Windows, Linux, macOS, entre outros. Além disso, tal linguagem pode ser trabalhada em interpretadores online gratuitos diretamente do navegador, tais como o Google Colab[1], dispensando sua instalação em computadores e/ou dispositivos móveis. Também é importante mencionar que essa linguagem possui uma grande comunidade de entusiastas em diversas áreas, tais como ciência de dados, inteligência artificial, desenvolvimento web, automação de tarefas, jogos, etc. Tais comunidades tem atuado no sentido de produzir bibliotecas de boa qualidade, fato que também tem contribuído para o crescimento da quantidade de programadores que trabalham com Python.

A gamificação consiste na aplicação de técnicas, elementos e dinâmicas associadas a jogos para enriquecer outros contextos. No âmbito educacional, sabe-se que ela promove um processo de imersão do estudante em um ambiente lúdico ou ficcional na forma de narrativas, imagens e sons [9, 10]. Esse processo tende a aproximar aspectos da realidade (aleatoriedade, competição, resolução de problemas, etc.) seguindo determinadas regras que trazem certo grau de controle. Dessa forma, a gamificação permite relacionar a motivação intrínseca e extrínseca do estudante, aumentando o nível de engajamento no contexto de tarefas didáticas [11, 12, 13]. Outros benefícios relacionados à gamificação estão relacionados à facilitação quanto à compreensão de conceitos e conhecimentos complexos e a retenção do conhecimento [14, 15]. Assim, a gamificação será incluída nessa proposta de pesquisa em um contexto de competição similar ao que ocorre frequentemente em Maratonas de Programação, Olimpíadas de Informática e no IEEE *Congress on Evolutionary Computation* (CEC)[2], um evento bem conhecido da área de AEs.

Dessa forma, o objetivo geral desse trabalho foi tentar responder a seguinte questão de pesquisa:

Como construir um produto educacional para o Ensino Médio mediado principalmente pelo Pensamento Computacional e pela gamificação onde AEs são usados para resolver problemas de otimização em diversos contextos?

Visando respondê-la, os seguintes objetivos secundários foram traçados:

[1] https://colab.research.google.com/
[2] https://ieeexplore.ieee.org/xpl/conhome/1000284/all-proceedings

1. Estudar os principais AEs que representam o estado da arte da literatura;

2. Construir um material teórico de boa qualidade que alie tanto o rigor necessário quanto uma linguagem mais acessível, de modo a facilitar o entendimento e a aplicação em sala de aula pelo professor interessado em fazê-lo;

3. Elaborar um produto educacional com sugestões de atividades incluindo gamificação onde AEs implementados na linguagem de programação Python são programados para resolver problemas de otimização em diversos contextos.

Para o desenvolvimento deste trabalho, foram realizadas algumas etapas importantes. Primeiramente, foi feita uma pesquisa na literatura nas principais bases de dados relacionadas ao tema e repositórios de pesquisa bem conhecidos. O objetivo dessa pesquisa foi encontrar trabalhos relacionados aos temas de Otimização, AEs e gamificação no contexto da educação básica. Em seguida, foi realizada uma revisão bibliográfica sobre os fundamentos teóricos desses conteúdos, com o objetivo de formar uma base conceitual sólida sobre o assunto. Por fim, construiu-se um produto educacional de modo que algoritmos evolutivos implementados em uma linguagem de programação são utilizados para resolver diversos problemas de otimização em um contexto similar ao que ocorre em maratonas de programação, olimpíadas de informática e competições do CEC. É importante mencionar que embora neste trabalho utiliza-se a linguagem de programação Python, não existem impedimentos à aplicação do produto educacional em outras linguagens. Nesse sentido, o organizador da competição pode escolher utilizar aquela que lhe for mais familiar. Uma das principais vantagens associadas a se utilizar Python é que um conjunto de códigos já está previamente disponibilizado no Apêndice A e, portanto, pode ser reutilizado.

Por fim, resolver um problema de otimização via Algoritmos Evolutivos é uma oportunidade em potencial para desenvolver o Pensamento Computacional de estudantes da Educação Básica, especialmente do Ensino Médio. Além disso, a gamificação também tem ganhado espaço entre as metodologias ativas de aprendizagem. Sabe-se que muitos dos conceitos envolvidos nessa pesquisa são recentes e não são do contato cotidiano do professor do ensino básico. Portanto, este trabalho pretende fornecer um produto educacional de boa qualidade que alie tanto o rigor necessário quanto uma linguagem mais acessível, repleto de exemplos. Dessa forma, ao ter contato com o produto desenvolvido, espera-se que o professor tenha condições de entender os principais conceitos e aplicar a proposta em sala

de aula.

Devido a variedade de conteúdos envolvidos neste trabalho, sugere-se que o mesmo seja aplicado para estudantes que já tenham alguma familiaridade com a linguagem de programação a ser utilizada. De forma geral, estudantes de itinerários formativos do Novo Ensino Médio que estejam relacionados à programação e modalidades de Ensino Médio atreladas a cursos técnicos em que linguagens de programação são trabalhadas constituem o público que reúne as melhores condições para usufruir de forma plena do presente trabalho. Contudo, o produto educacional pode ser aplicado mesmo em turmas que não tenham conhecimentos prévios em programação, cabendo ao professor realizar as adaptações necessárias.

O restante do trabalho se constitui como descrito na sequência. O Capítulo 2 apresenta aspectos teóricos sobre Otimização, dando o leitor a oportunidade de revisitar os principais conceitos que serão fundamentais para o entendimento do restante do trabalho. No Capítulo 3 são apresentadas as ideias relacionadas à evolução biológica, além dos conceitos básicos de AEs, incluindo a descrição de alguns dos algoritmos mais importantes, como Algoritmos Genéticos e Evolução Diferencial. O Capítulo 4 apresenta trabalhos correlatos que abordam os principais aspectos matemáticos e educacionais tais como otimização, algoritmos evolutivos e gamificação. O Capítulo 5 constitui-se da descrição do produto educacional com as atividades propostas, enquanto o Capítulo 6 apresenta as conclusões desta dissertação.

2 Otimização

Neste capítulo, apresentamos os principais fundamentos teóricos relacionados à área da Matemática conhecida como Otimização. De forma simples, podemos dizer que o objetivo deste conhecimento é estudar métodos para obter os valores das variáveis independentes que levam a um valor de máximo ou de mínimo para uma função de custo dependente dessas mesmas variáveis. Inicialmente, apresentamos as principais definições e os conceitos fundamentais relacionadas ao tema para, em seguida, apresentarmos de forma resumida os aspectos computacionais que tipicamente são abordados neste contexto. O intuito deste capítulo é apresentar uma visão geral contendo os principais aspectos relacionados a esse saber matemático, de modo a preparar o leitor tanto em termos de linguagem e notação quanto em termos de conceitos que serão apresentados nos capítulos seguintes.

2.1 Introdução

A otimização é uma área da matemática com aplicações em problemas de engenharia [16, 17, 18, 19, 20], estatística [21, 22], logística [23, 24], biologia [25, 26, 27, 28], economia [29, 30, 31, 32], dentre outras áreas do conhecimento. No Ensino Médio, o aluno é apresentado a problemas simples de otimização envolvendo, por exemplo, cálculo do vértice de uma parábola ao se estudar a função quadrática.

Para definir matematicamente o problema de otimização no contexto deste trabalho, considere o vetor de $m \in \mathbb{N}$ variáveis independentes $\vec{x} \in D \subset \mathbb{R}^m$, $\vec{x} = (x_1, x_2, \ldots, x_m)$, e $f\colon D \subset \mathbb{R}^m \to \mathbb{R}$, uma função a ser otimizada, conhecida como função de custo ou função objetivo. Dizemos que este problema corresponde ao *contexto escalar* pelo fato de $f(\vec{x}) \in \mathbb{R}$, isto é, f é um *funcional*: uma função cuja imagem é um escalar, ou seja, um número real. Neste trabalho, os problemas de otimização estão restritos ao contexto escalar.

Ao otimizar (minimizar ou maximizar) o funcional f, classificamos a otimização

como *mono-objetivo*, isto é, deseja-se otimizar apenas uma função objetivo. Também é possível otimizar uma função vetorial que, quando possui duas ou três funções objetivo, tem-se uma otimização do tipo *multiobjetivo*. Para mais de três funções objetivo, utiliza-se o termo otimização de *muitos objetivos* (em inglês, *many-objective optimization*) [33]. A diferenciação desses termos se dá em função de existir a tendência de que os algoritmos projetados para o contexto *multiobjetivo*, em geral, perderem sua eficiência em problemas de otimização de *muitos objetivos*.

A seguir, são apresentadas definições e conceitos fundamentais para o entendimento de um problema de otimização. O leitor interessado em um estudo mais aprofundado pode encontrar maiores detalhes em referências como [17, 34, 35]. As definições aqui apresentadas foram extraídas ou adaptadas das fontes citadas para o aprofundamento do leitor. Mesmo quando adaptadas, as definições são equivalentes àquelas apresentadas nos materiais originais. As adaptações foram realizadas com o intuito de tornar o texto o mais didático possível, mantendo um nível de rigor que julgamos importante. Além disso, as adaptações visam manter um padrão de notação ao longo de todo o texto que foi inspirado em várias fontes e que utilizam notações distintas.

2.2 Definições e Conceitos Básicos

Lembremos que, para o contexto deste trabalho, iremos considerar o vetor de $m \in \mathbb{N}$ variáveis independentes $\vec{x} \in D \subset \mathbb{R}^m$, $\vec{x} = (x_1, x_2, \ldots, x_m)$. As importantes definições para o restante do trabalho são apresentadas a seguir.

Definição 2.1 (Conexidade): Seja o conjunto $D \subset \mathbb{R}^m$. Uma cisão de D é uma decomposição $D = A \cup B$, em que $A \cap B = \emptyset$ e os conjuntos A, B são ambos abertos em D. Todo conjunto $D \subset \mathbb{R}^m$ admite pelo menos a cisão trivial, $D = D \cup \emptyset$. Um conjunto D é conexo quando não admite outra cisão além da trivial. Quando existir uma cisão não trivial $D = A \cup B$, diremos que D é desconexo [36, p. 54].

De forma intuitiva, um conjunto é conexo, se for possível passar de um ponto qualquer dele para outro ponto distinto qualquer por meio de um movimento contínuo, sem sair deste conjunto.

Definição 2.2 (Convexidade): Dizemos que um conjunto $D \subset \mathbb{R}^m$ é convexo se para

quaisquer dois vetores $\vec{x}, \vec{y} \in D$, existe

$$\vec{z} = \alpha\vec{x} + (1-\alpha)\vec{y}, \tag{2.1}$$

sendo $\vec{z} \in D$ para qualquer $\alpha \in [0,1]$. A operação acima é conhecida como combinação convexa .

Geometricamente, um conjunto é conexo quando, dado dois pontos P,$Q \in D, P \neq Q$, tivermos o segmento $\overline{PQ} \subset D$.

Definição 2.3 (Continuidade): Dizemos que o funcional $f\colon D \subset \mathbb{R}^m \to \mathbb{R}$ é contínuo, se para todo $\vec{x}_0 \in D$, tem-se que $f(\vec{x})$ está definido e $\lim_{\vec{x}\to\vec{x}_0} f(\vec{x}) = f(\vec{x}_0)$.

Definição 2.4 (Diferenciabilidade): Dizemos que o funcional $f\colon D \subset \mathbb{R}^m \to \mathbb{R}$ é diferenciável, se para todo $\vec{x}_0 \in D$, existe o vetor gradiente avaliado neste ponto:

$$\nabla f(\vec{x})\Big|_{\vec{x}=\vec{x}_0} = \left(\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \cdots \quad \frac{\partial f}{\partial x_{m-1}} \quad \frac{\partial f}{\partial x_m}\right)\Bigg|_{\vec{x}=\vec{x}_0}, \tag{2.2}$$

em que operador $\partial[\cdot]/\partial x_m$ corresponde ao operador derivada parcial calculado com respeito a variável x_k, $k \in \{1, 2, \ldots, m\}$:

$$\frac{\partial f(x_1, \ldots, x_m)}{\partial x_k} = \lim_{h\to 0} \frac{f(x_1, \ldots, x_k + h, \ldots, x_n) - f(x_1, \ldots, x_k, \ldots, x_m)}{h}. \tag{2.3}$$

Dois importantes subconjuntos de $D \subset \mathbb{R}^m$ são definidos a seguir.

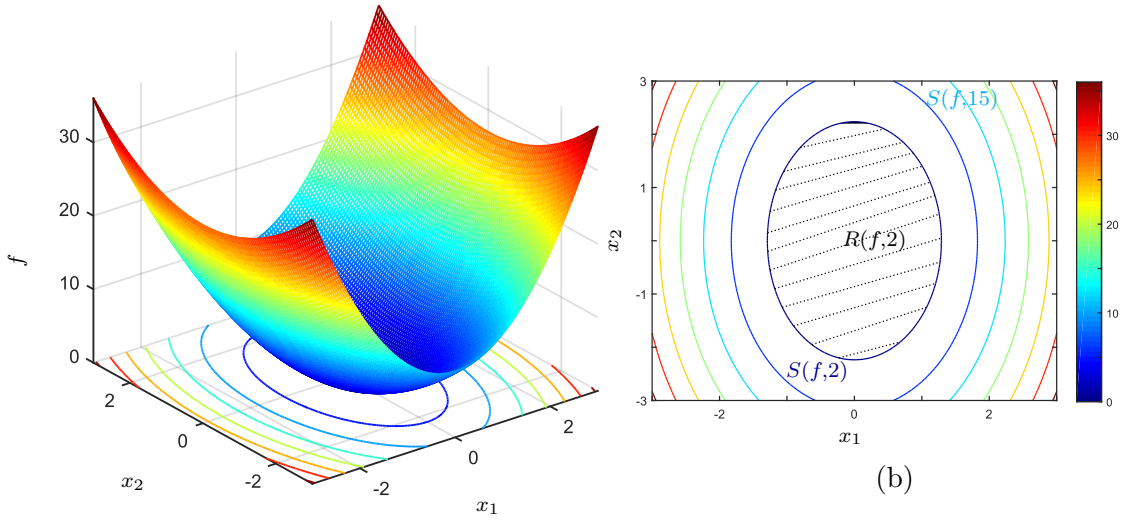
Definição 2.5 (Superfície de Nível): A superfície de nível $S(f, \alpha)$, associada ao nível α é definida como o conjunto: $S(f, \alpha) = \{x \in D \subset \mathbb{R}^m | f(\vec{x}) = \alpha\}$ [17, p. 30].

Definição 2.6 (Região de Sub-Nível): A região de sub-nível $R(f, \alpha)$, associada ao nível α é definida como o conjunto: $R(f, \alpha) = \{x \in D \subset \mathbb{R}^m \mid f(\vec{x}) \leq \alpha\}$ [17, p. 30].

Para ilustrar estes dois conceitos, apresentamos a Figura 2.1. Considerando $m = 2$, $\vec{x} = (x_1, x_2)$, e utilizou-se o funcional, $f\big((x_1, x_2)\big) = 3x_1^2 + x_2^2$. Note que, como o funcional retorna um escalar $f(\vec{x})$, tais valores são apresentados no eixo vertical do gráfico mostrado na Figura 2.1a. Para facilitar a visualização, optou-se por restringir x_1 e x_2, $D = \{(x_1, x_2) \mid x_1, x_2 \in \mathbb{R}, -3 \leq x_1, x_2 \leq 3\}$, de modo que $f(\vec{x}) \leq 36$ para todo $\vec{x} \in D$. Além disso, neste mesmo gráfico, tem-se as curvas de nível de f apresentadas no plano $x_1 \times x_2$. Note que

as curvas de nível correspondem aos pares (x_1, x_2) no plano citado, em que o funcional apresenta o mesmo valor escalar. Para explicitar esses pontos, tem-se na Figura 2.1b o plano $x_1 \times x_2$ em evidência, com as respectivas curvas de nível. Destaca-se a superfície de nível $S(f, 15)$, correspondente a todos os pares (x_1, x_2) em que $f\big((x_1, x_2)\big) = 15$. Também está em evidência a região de subnível $R(f, 2)$ que, por sua vez, corresponde a todos os pontos (x_1, x_2) em que $f\big((x_1, x_2)\big) \leq 2$, ou seja, incluem $S(f, 2)$ e seus pontos interiores.

Figura 2.1: Gráfico e curvas de nível para o funcional de duas variáveis, $f\big((x_1, x_2)\big) = 3x_1^2 + x_2^2$. Em (a), tem-se o gráfico em três dimensões de f, enquanto as curvas de nível de f são apresentadas no plano $x_1 \times x_2$. Em (b), tem-se o plano $x_1 \times x_2$, destacando-se as curvas de nível, em especial, $S(f, 15)$ $(-)$. Também em destaque, tem-se a região de subnível $R(f, 2)$ $(\cdots)$, limitada por $S(f, 2)$ $(-)$.



Fonte: Elaborada pelo autor.

Definição 2.7 (Unimodalidade): Dizemos que f é unimodal se $R(f, \alpha)$ é um conjunto conexo para todo $\alpha \in \mathbb{R}$ [17, p. 31].

Definição 2.8 (Multimodalidade): Dizemos que f é multimodal se existe $\alpha \in \mathbb{R}$ tal que $R(f, \alpha)$ é um conjunto não conexo [17, p. 31].

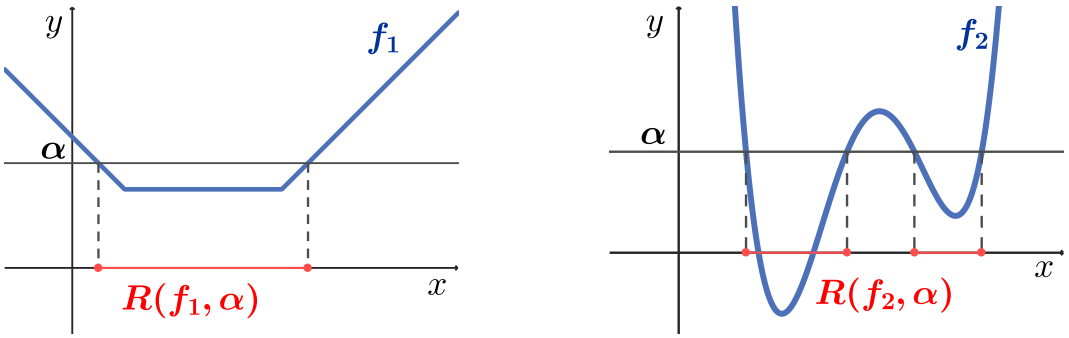
Com o intuito de exemplificar esses conceitos, considere:

$$f_1(x) = \frac{1}{2}(|x - 2| + |x - 8|), \tag{2.4}$$

$$f_2(x) = \frac{x^4}{10} - \frac{23x^3}{10} + \frac{37x^2}{2} - \frac{601x}{10} + 66. \tag{2.5}$$

O gráfico de $f_1(x)$ (2.4), um funcional unimodal, é apresentado na Figura 2.2a. Note que para $\alpha = 4$, o conjunto $R(f_1, 4)$ é conexo, o que também ocorreria para qualquer outro α que fosse arbitrariamente escolhido devido à unimodalidade. Por outro lado, o gráfico de $f_2(x)$ (2.5) é mostrado na Figura 2.2b. Este último é um exemplo de multimodalidade, dado que é possível encontrar algum α para o qual $R(f_2, \alpha)$ não é conexo. Como exemplo, ilustramos na figura o que ocorre para $\alpha = 3$, em que vemos que $R(f_2, 3)$ é um conjunto não conexo.

Figura 2.2: Exemplos de unimodalidade e multimodalidade. Em (a), tem-se o gráfico de $f_1(x)$ (2.4), unimodal, dado que para qualquer α escolhido $R(f_1,\alpha)$ é um conjunto conexo. Por outro lado, note que em (b), tem-se um exemplo de multimodalidade para $f_2(x)$ (2.5), dado que para $\alpha = 3$, $R(f_2,3)$ é um conjunto não conexo.



(a) (b)

Fonte: Elaborada pelo autor.

Definição 2.9 (Minimizador Global e Mínimo Global): Dizemos que $\vec{x}^*$ é minimizador global de f, se $f(\vec{x}^*) \leq f(\vec{x})$ para todo $\vec{x} \in D \subset \mathbb{R}^m$. Assim, $f(\vec{x}^*)$ é o valor de mínimo global.

Definição 2.10 (Maximizador Global e Máximo Global): Similarmente, dizemos que $\vec{x}^*$ é maximizador global de f, se $f(\vec{x}^*) \geq f(\vec{x})$ para todo $\vec{x} \in D \subset \mathbb{R}^m$. Assim, $f(\vec{x}^*)$ é o valor de máximo global.

Nas próximas definições, $\|\cdot\|$ representa a norma euclidiana.

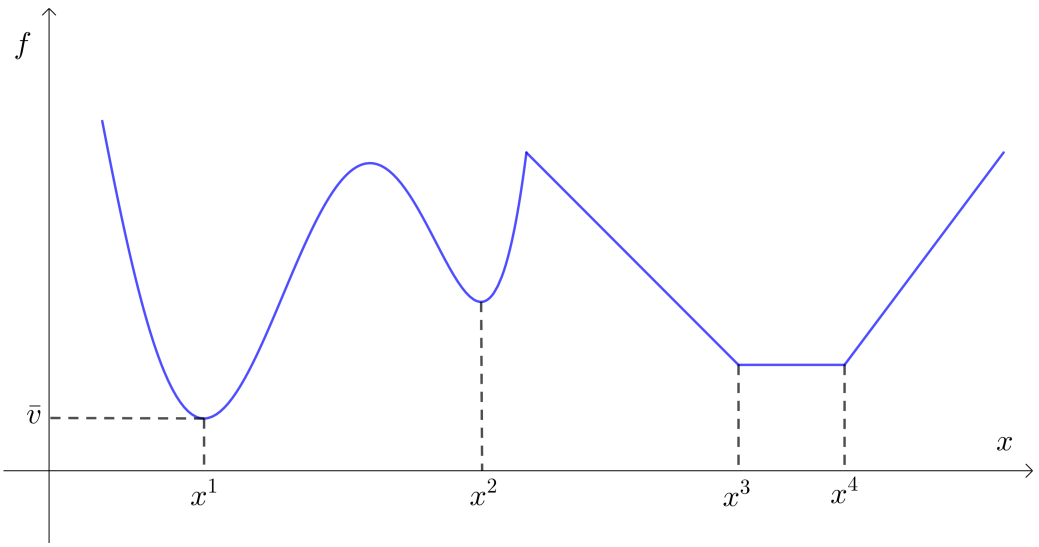
Definição 2.11 (Minimizador Local e Mínimo Local): Dizemos que $\vec{x}^*$ é minimizador local de f, se existir um $\epsilon > 0$ tal que $f(\vec{x}^*) \leq f(\vec{x})$ para todo $\vec{x} \in D \subset \mathbb{R}^m$ com $\|\vec{x} - \vec{x}^*\| < \epsilon$. Assim, $f(\vec{x}^*)$ é um valor de mínimo local.

Definição 2.12 (Maximizador Local e Máximo Local): De modo similar, dizemos que $\vec{x}^*$ é

maximizador local de f, se existir um $\epsilon > 0$ tal que $f(\vec{x}^*) \geq f(\vec{x})$ para todo $\vec{x} \in D \subset \mathbb{R}^m$ com $\|\vec{x} - \vec{x}^*\| < \epsilon$. Assim, $f(\vec{x}^*)$ é um valor de mínimo local.

No caso em que as desigualdades das Definições 2.9 e 2.11 são estritas, o conjunto de minimizadores é dito *estrito*. Isso significa que este conjunto, em contexto local ou global, possui apenas um elemento. Caso contrário, dizemos que o conjunto de minimizadores é *não estrito* e esses conceitos também apresentam seus análogos para os maximizadores. A Figura 2.3 ilustra os conceitos de minimizadores e mínimos, locais e globais, para uma função em particular. Além disso, note que um funcional *unimodal* pode possuir múltiplos mínimos locais ou globais desde que eles formem um conjunto conexo como podemos verificar nas Figuras 2.2a e 2.3.

Figura 2.3: x^1 é o minimizador global ($\bar{v}$ é o valor ótimo), x^2 é um minimizador local estrito, [x^3,x^4] é um conjunto de minimizadores locais não estritos.



Fonte: Ismailov e Solodov (2005) [34] (Adaptada).

Definição 2.13 (Região Conexa de Sub-Nível): Seja a região de sub-nível $R(f,\alpha)$ associada ao nível α e seja o ponto $\vec{x}_0 \in R(f,\alpha)$. A região conexa de sub-nível $R_c(f,\alpha,\vec{x}_0)$ é o maior subconjunto conexo de $R(f,\alpha)$ que contém $\vec{x}_0$. [17, p. 32].

Definição 2.14 (Bacia de Atração): Seja $\vec{x}_* \in D \subset \mathbb{R}^m$ um mínimo local de f. A bacia de atração de $\vec{x}_*$ é a maior região conexa de sub-nível associada a $\vec{x}_*$, em que α_* é o nível correspondente, de modo que a função restrita a essa região, $f_*\colon R_c(f,\alpha_*,\vec{x}_*) \to \mathbb{R}$, é unimodal [17, p. 32].

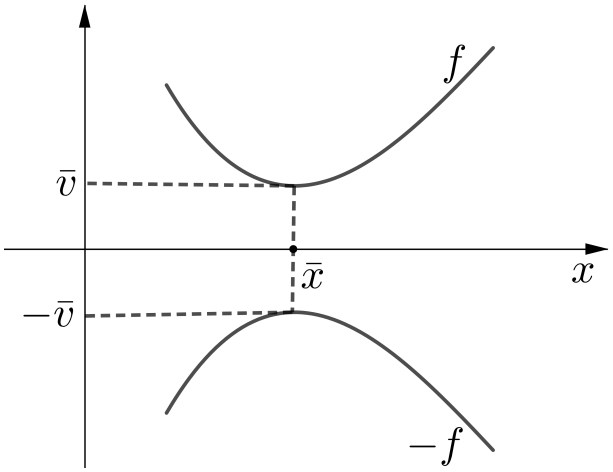
Assim, o problema de minimização *irrestrito* pode ser expresso como:

$$\vec{x}^* = \arg\min_{\vec{x}} f(\vec{x}), \tag{2.6}$$

consistindo em encontrar o minimizador global $\vec{x}^*$ do funcional f, sem que restrições sejam impostas, isto é: $\vec{x} \in D = \mathbb{R}^m$. Caso o objetivo seja resolver o problema de maximização irrestrito sobre f, este pode ser transformado em um problema de minimização como se segue (fato ilustrado pela Figura 2.4):

$$\vec{x}^* = \arg\min_{\vec{x}} f(\vec{x}) = \arg\max_{\vec{x}} \left[-f(\vec{x})\right]. \tag{2.7}$$

Figura 2.4: Na Figura, temos que o ponto $\bar{x}$ que maximiza $-f(x)$ é o mesmo que minimiza $f(x)$.



Fonte: Ismailov e Solodov (2005) [34] (Adaptada).

Dessa forma, como um problema de maximização pode ser transformado em um problema de minimização, a formulação do problema dada em (2.6) será utilizada de forma geral como a formulação de um problema de *otimização irrestrita* e, assim, diremos que $\vec{x}^*$ é a *solução do problema de otimização irrestrita*.

Ocorre que, em muitos cenários, além de minimizar o funcional objetivo, é necessário atender restrições. Essas restrições podem ser expressas por $r \in \mathbb{N}$ desigualdades, como $g_i(\vec{x}) \leq 0$, $\forall i = 1, \ldots, r$ e/ou $p \in \mathbb{N}$ igualdades descritas como $h_j(\vec{x}) = 0, \forall j = 1, \ldots, p$. Tal conjunto de restrições compõem um conjunto chamado de *conjunto viável* ou *região factível* D:

$$D = \left\{ \vec{x} \in \mathbb{R}^m \middle| \begin{array}{rcl} g_i(\vec{x}) & \leq & 0, \forall i = 1, \ldots, r \\ h_j(\vec{x}) & = & 0, \forall j = 1, \ldots, p. \end{array} \right\}. \tag{2.8}$$

Ao se impor as restrições dadas em (2.8) sobre o problema de otimização, tem-se que a busca por soluções se concentra apenas sobre o subconjunto formado pelos candidatos que atenderem a essas restrições, o conjunto viável D. Portanto, as restrições definem o espaço de valores factíveis para as variáveis independentes. Nesse sentido, as soluções do problema de otimização que atendem às restrições são chamadas de *soluções factíveis* do problema de otimização *com restrições*.

Matematicamente, o problema de otimização considerando as restrições pode ser escrito como

$$\vec{x}^* = \arg\min_{\vec{x}} f(\vec{x}) \text{ sujeito a } \vec{x} \in D, \tag{2.9}$$

ou seja, queremos obter o vetor $\vec{x}^*$, argumento do funcional de custo, que atende às restrições de D (2.8) e minimiza este funcional. É importante mencionar que a função de custo, usualmente, em problemas práticos, está associada a um modelo para um sistema físico [17]. Nesse sentido, as restrições podem estar ligadas à natureza física do problema como por exemplo, a existência de um valor de saturação para a transformação de energia no sistema; à impossibilidade de se utilizar valores de um dado subconjunto, como por exemplo, no caso em que não faz sentido utilizar valores negativos para algumas variáveis; ao desempenho, quando obrigatoriamente se quer impor uma condição de desempenho mínima para o sistema; dentre outras diversas razões [17, 18].

Posto o problema de otimização, existem diversas técnicas e algoritmos utilizados para encontrar as soluções do problema de otimização de forma metódica. Assim, neste trabalho, os termos *método* e *algoritmo* são utilizados para designar essas sequências de operações e passos que visam obter a solução do problema de otimização de forma sistemática. Em geral, os algoritmos de otimização são *iterativos*, isto é: são iniciados a partir de uma estimativa inicial, gerando uma sequência de aproximações até se encontrar uma aproximação satisfatória do minimizador. Assim, os algoritmos se distinguem pelas estratégias utilizadas para se mover de uma iteração para a seguinte. Nos métodos determinísticos clássicos esse processo é feito utilizando-se de informações advindas da própria função objetivo ou de suas propriedades matemáticas que podem incluir suas derivadas [17, 6, 8].

A seguir, apresenta-se um exemplo simples em que as definições desta seção são aplicadas. O objetivo do exemplo é mostrar como os conceitos abordados até aqui podem

ser aplicados em um problema. No exemplo, mostra-se como a partir de um problema que envolve uma aplicação prática, obtém-se o modelo matemático, nesse caso, a função objetivo. Em seguida, define-se o problema de otimização que é resolvido de forma analítica, utilizando resultados do Cálculo Diferencial. É importante frisar que o exemplo aqui abordado é uma situação simples, sendo que as técnicas para obtenção de modelos matemáticos para estudar fenômenos são estudadas em uma área de pesquisa conhecida como *Modelagem Matemática* que não faz parte dos objetivos deste trabalho.

Exemplo 2.2.1: Uma lata cilíndrica é feita para receber um litro de óleo. Encontre as dimensões que minimizarão o custo da lata [37, p. 333].

Sejam $h \in \mathbb{R}^+$ e $r \in \mathbb{R}^+$ a altura e o raio da base da lata cilíndrica em centímetros, respectivamente. O custo da lata é proporcional à área da superfície total de um cilindro de altura h e raio r. Portanto, ao minimizarmos tal área, estaremos também minimizando o custo. A área mencionada é expressa por

$$A = 2\pi r^2 + 2\pi r h, \tag{2.10}$$

em que a primeira parcela corresponde a soma das áreas das duas bases circulares de raio r, enquanto a segunda parcela corresponde a área da superfície lateral - um retângulo de base $2\pi r$ e altura h.
Como o volume da lata cilíndrica deve ser igual a 1000 cm^3, devemos ter

$$\pi r^2 h = 1000 \Rightarrow h = \frac{1000}{\pi r^2}. \tag{2.11}$$

Substituindo h (2.11) em A (2.10), conseguimos eliminar a dependência de h, obtendo

$$A = 2\pi r^2 + 2\pi r \left(\frac{1000}{\pi r^2}\right) = 2\pi r^2 + \frac{2000}{r}.$$

Portanto, a *função objetivo de uma variável* a ser minimizada é

$$A(r) = 2\pi r^2 + \frac{2000}{r} \quad \text{com} \quad r > 0. \tag{2.12}$$

Utilizando a formulação do problema conforme (2.9), temos o *problema de minimização com restrições* definido

$$r^* = \arg\min_r A(r) \text{ sujeito a } r \in \mathbb{R}^+. \tag{2.13}$$

Note que a função objetivo de uma variável é *contínua* e *diferenciável*, portanto, diferenciamos

$$\frac{d}{dr}A(r) = A'(r) = 4\pi r - \frac{2000}{r^2} = \frac{4(\pi r^3 - 500)}{r^2}. \tag{2.14}$$

Os pontos críticos de uma função de uma variável são pontos candidatos a ótimo desta função em que a derivada desta função é nula ou não existe:

- $A'(r) = 0$:

$$\frac{4(\pi r^3 - 500)}{r^2} = 0 \Rightarrow r = \sqrt[3]{\frac{500}{\pi}} \tag{2.15}$$

  Atente-se ao fato de que, devemos considerar apenas as soluções reais, dado que $r \in \mathbb{R}^+$, assim as soluções complexas conjugadas de (2.15) não são levadas em conta.

- $A'(r)$ não existe:

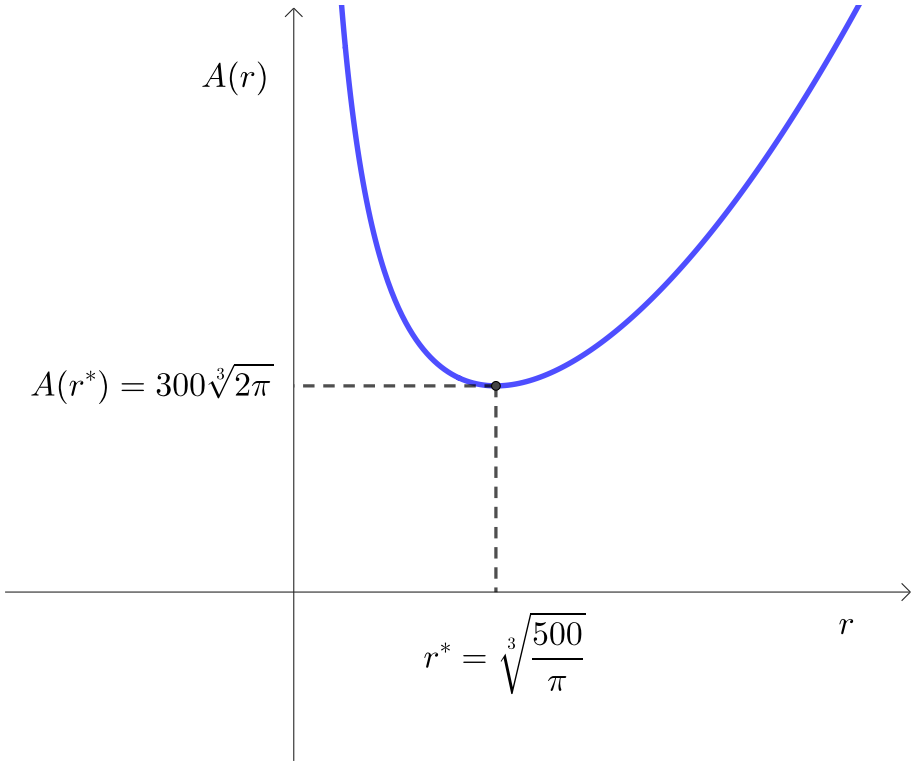
$$r^2 = 0 \Rightarrow r = 0 \tag{2.16}$$

  Note que, $r = 0 \notin \mathbb{R}^+$, portanto este não é um ponto dentro da *região factível*.

Concluímos que existe apenas um ponto crítico, $r = \sqrt[3]{\frac{500}{\pi}}$. Note que para $r < \sqrt[3]{\frac{500}{\pi}}$, tem-se que $A'(r) < 0$, de modo que $A(r)$ está decrescendo para todo r à esquerda do ponto crítico. Por outro lado, para $r > \sqrt[3]{\frac{500}{\pi}}$, tem-se que $A'(r) > 0$, sendo que $A(r)$ está crescendo para todo r à direita do ponto crítico. Assim, conclui-se que o ponto crítico $r^* = \sqrt[3]{\frac{500}{\pi}}$ é *o minimizador global* do problema, associado ao *valor mínimo global* $A(r^*) = 300\sqrt[3]{2\pi}$.

A Figura 2.5 mostra o gráfico da função objetivo (2.12) que é *convexa*, evidenciando o minimizador global e seu valor mínimo. Podemos perceber a *bacia de atração* única, de modo que tal função é *unimodal*.

Figura 2.5: Gráfico da função objetivo do exemplo da lata cilíndrica.



Fonte: Elaborada pelo autor.

2.3 Uma Visão Geral sobre os Métodos

O objetivo desta seção é apresentar noções básicas e uma visão geral sobre métodos computacionais (algoritmos) que resolvem problemas de otimização em geral. Esta seção se baseia no conteúdo apresentado em Ismailov e Solodov[35], de modo que o leitor interessado em maiores detalhes pode recorrer a tal referência.

Para resolver o problema de otimização formulado em (2.9), utilizaremos procedimentos computacionais. Nesse contexto, calcula-se ou avalia-se a função objetivo e suas restrições para um conjunto de pontos dentro do espaço de busca. Também é comum utilizar a informação das derivadas da função objetivo e das restrições. Como, nesses métodos, cada ponto é conhecido a partir da informação obtida nos pontos anteriores são conhecidos como *sequenciais* ou *iterativos*. Um método iterativo gera uma sequência de pontos em $\mathbb{R}^m$, $\{\vec{x}_k\} = \vec{x}_1, \ldots, \vec{x}_k, \ldots$, chamada de *trajetória*, em que cada ponto é conhecido como uma *aproximação* à solução do problema ou um *iterando* do método [35]. Uma iteração do método consiste em gerar uma nova aproximação, $\vec{x}_{k+1}$, a partir da anterior, $\vec{x}_k$. Na k-ésima iteração, $U_k \subset \mathbb{R}^m$ corresponde ao conjunto dos valores que $\vec{x}_k$ pode assumir. Assim, partindo-se de um ponto inicial $\vec{x}_0 \in U_0 \subset \mathbb{R}^m$, o método consiste

em um procedimento para obter a trajetória $\{\vec{x}_k\}$.

A ordem do método está associada a maior ordem de derivadas da função objetivo ou das restrições que são calculadas ao longo das gerações. Portanto, um método de ordem zero, não utiliza derivadas, mas apenas os valores das funções. Por outro lado, métodos de primeira ordem utilizam as derivadas primeiras dessas funções, de segunda ordem, as derivadas segundas e, assim, sucessivamente.

2.3.1 Convergência

No caso em que, partindo-se de qualquer $\vec{x}_0 \in U_0$, encontra-se $\vec{x}_n = \vec{x}^*$ que seja solução do problema de otimização, para algum n finito, diz-se que o método apresenta *convergência finita* [35]. Tipicamente, tem-se esse tipo de convergência para uma gama de problemas de maior simplicidade, como aqueles de programação linear e quadrática. Para problemas mais complexos, utiliza-se métodos de *convergência assintótica*. Neles, embora não seja possível garantir que $\vec{x}_n$ seja uma solução exata, para n suficiente grande, é possível dizer que $\vec{x}_n$ é uma boa aproximação para a solução do problema segundo algum critério [35]. De acordo com a natureza da aproximação, falamos dos seguintes tipos de convergência em relação:

- **Às Variáveis do Problema**: à medida que k se torna suficientemente grande, $\vec{x}_k$ converge para $\vec{x}^*$.
- **Ao Conjunto de Soluções**: à medida que k se torna suficientemente grande, a distância de $\vec{x}_k$ para o conjunto das soluções S converge para zero.
- **À Função Objetivo**: à medida que k se torna suficientemente grande, $f(\vec{x}_k)$ converge para $\bar{v}$, valor ótimo do problema.
- **Ao Gradiente**: à medida que k se torna suficientemente grande, $\{f'(\vec{x}_k)\}$ converge para zero.
- **Global**: para $U_0 = \mathbb{R}^m$, tem-se convergência partindo-se de qualquer $\vec{x}_0 \in \mathbb{R}^m$.
- **Local**: quando algum tipo de convergência pode ser garantido somente para pontos suficientemente próximos a uma solução.

2.3.2 Taxas de Convergência

Suponha que um método seja convergente a uma solução ótima $\vec{x}^*$ pelo menos em um contexto local: $\{\vec{x}_k\} \to \vec{x}^*$ para $k \to \infty$. O conceito de *taxa de convergência* está associado à velocidade de redução da distância $\|\vec{x}_{k+1} - \vec{x}^*\|$ em relação à redução da distância $\|\vec{x}_k - \vec{x}^*\|$, para k suficientemente grande. Considerando $\vec{x}_0 \in \mathbb{R}^m$, suficiente próximo de $\vec{x}^*$, $\vec{x}_k \neq \vec{x}^*$, para todo k, e que as constantes a seguir, q e C, não dependem de $\vec{x}_0$, tipicamente destacam-se na literatura as seguintes denominações para taxas de convergência [35]:

1. **Convergência linear:**

   Se existe $q \in (0,1)$ tal que

   $$\limsup_{k\to\infty} \frac{\|\vec{x}_{k+1} - \vec{x}^*\|}{\|\vec{x}_k - \vec{x}^*\|} = q, \tag{2.17}$$

   dizemos que esse tipo de convergência ocorre.

2. **Convergência superlinear:**

   Quando em (2.17), tem-se $q = 0$.

3. **Convergência sublinear:**

   Quando em (2.17), tem-se $q = 1$, de modo que esta é mais lenta do que a convergência linear.

4. **Convergência aritmética:**

   Ocorre quando, para uma constante fixa $C > 0$, a desigualdade

   $$\limsup_{k\to\infty} k\|\vec{x}_k - \vec{x}^*\| \leq C \tag{2.18}$$

   é satisfeita.

5. **Convergência geométrica:**

   Quando, para uma constante fixa $C > 0$, a desigualdade

   $$\limsup_{k\to\infty} \frac{\|\vec{x}_k - \vec{x}^*\|}{q^k} \leq C \tag{2.19}$$

   é verdadeira para $q \in (0,1)$.

6. **Convergência quadrática:**

   Se existir $q \in (0, 1)$ tal que:

$$\limsup_{k\to\infty} \frac{\|\vec{x}_{k+1} - \vec{x}^*\|}{\|\vec{x}_k - \vec{x}^*\|^2} = q, \tag{2.20}$$

   dizemos que a convergência é quadrática, um caso particular da taxa superlinear.

2.3.3 Propriedades Desejáveis para Métodos de Otimização

Um conceito importante para avaliar a boa qualidade de um método de otimização é conhecido como *custo computacional*. Tal conceito está relacionado tanto ao tempo necessário para o processamento quanto à quantidade de memória que deve ser utilizada para execução do método. Portanto, esse conceito está intimamente ligado ao gasto energético dos recursos computacionais que o método necessita para resolver o problema, sendo um aspecto relevante tanto na escolha de qual algoritmo será utilizado para resolução do problema quanto para se comparar algoritmos distintos que visam resolver um mesmo problema de otimização.

Nesse sentido, é importante mencionar que são desejáveis as seguintes propriedades para algoritmos de otimização [6]:

Robustez: capacidade de obter bom desempenho mesmo sujeito a: i) incertezas nas variáveis, ii) variações na função objetivo ou iii) variações nas inicializações do método, considerando diversos pontos de partida;

Eficiência: baixa exigência de tempo computacional e necessidade de armazenamento de dados;

Precisão: baixa sensibilidade a erros numéricos decorrentes da implementação em computadores.

2.3.4 Regras ou Critérios de Parada

Como todo processo computacional deve ser *finito*, é importante que seja definida alguma *regra de parada*, também conhecido como *critério de parada*. Tais critérios, definem condições que, quando satisfeitas, encerram o método, entregando uma aproximação para o otimizador, dado que, em geral, os métodos apresentam convergência assintótica. No

caso em que as taxas de convergência são conhecidas, as regras de parada mais confiáveis são baseadas nessas estimativas, contudo, as taxas de convergência tendem a apresentar natureza local além de serem, em geral, dependentes de constantes nem sempre conhecidas (como q e C em (2.17), (2.18), (2.19) e (2.20)) [35].

Embora as regras que tipicamente são utilizadas no contexto da prática computacional não apresentem a mesma confiabilidade teórica daquelas baseadas nas taxas de convergência, elas tendem a ser mais facilmente implementadas. Tais regras tendem a examinar o comportamento dos valores mais recentes de $\{\vec{x}_k\}$ e/ou de $\{f(\vec{x}_k)\}$ e/ou suas derivadas [35]. Utilizando um valor pequeno $\epsilon > 0$ definido pelo próprio projetista do algoritmo, tem-se como possíveis critérios para se encerrar o método na iteração $k+1$:

$$||\vec{x}_{k+1} - \vec{x}_k|| \leq \epsilon, \tag{2.21}$$

$$||f(\vec{x}_{k+1}) - f(\vec{x}_k)|| \leq \epsilon, \tag{2.22}$$

$$||f'(\vec{x}_{k+1})|| \leq \epsilon. \tag{2.23}$$

No caso do critério (2.21), o método é encerrado quando o passo de aproximação entre duas soluções fica curto, o que aponta para uma tendência de que a continuação do método não promova melhorias na qualidade da aproximação [35]. Para o critério (2.22), tem-se um caso análogo, em que o passo produzido não gerou aumento significativo na qualidade dos valores da função objetivo, indicando uma tendência de acomodação em torno de algum valor ótimo local [35]. Por fim, no caso de funções objetivo diferenciáveis em um contexto irrestrito, pode-se utilizar a informação da derivada conforme apresentado no critério (2.23) ao invés de (2.22), de modo que a tendência desses valores de se aproximarem de zero, podem indicar a existência de um ótimo local. É importante frisar que esses critérios de parada são frágeis do ponto de vista teórico pelo fato de não garantirem a proximidade do iterando $\vec{x}_{k+1}$ a uma solução do problema em qualquer sentido, contudo, eles são tipicamente utilizados pelo fato de não existirem alternativas melhores [35].

Como o algoritmo não pode funcionar por um tempo ilimitado, um critério de parada associado a quantidade de iterações ou ao número de avaliações da função objetivo está implícito a esses métodos computacionais e deve ser definido pelo projetista [35]. Nesse sentido, tipicamente algoritmos de otimização combinam diversas regras de parada organizadas hierarquicamente, sendo que estas escolhas muitas vezes são definidas por questões empíricas ao invés de estarem embasadas em fundamentos teóricos.

Portanto, tendo sido os aspectos e conceitos fundamentais relacionados à área de otimização que são importantes para uma boa compreensão do restante do texto, o próximo capítulo se dedica a apresentar os Algoritmos Evolutivos (AEs). Tais algoritmos são as estratégias utilizadas para resolver problemas de otimização, abordadas no contexto deste trabalho.

3 Algoritmos Evolutivos

Neste capítulo são apresentados os fundamentos teóricos associados aos algoritmos evolutivos que desempenham papel fundamental nesta dissertação. Inicialmente, tem-se uma apresentação da evolução biológica que é, em seguida, associada a área de Computação Evolucionária (CE) por meio da metáfora fundamental da CE. Na sequência são apresentados os aspectos teóricos fundamentais dos Algoritmos Evolutivos e, a partir de então, descreve-se de forma geral duas estratégias fundamentais: os Algoritmos Genéticos (AG) e a Evolução Diferencial (ED).

3.1 Introdução

Algoritmos Evolutivos são ferramentas computacionais que, baseados em conceitos de evolução dos seres vivos[1], visam resolver problemas de otimização [8]. Nesse sentido, existe uma população de indivíduos que carregam informações sobre as variáveis do problema e sobre a função de custo. Tais indivíduos passam por operações que envolvem seleções, mutações e cruzamentos, nas quais ocorrem comparações entre os indivíduos, a troca de informações entre eles e a inserção de elementos de forma meramente aleatória. Assim, ao percorrer o espaço de busca, a população visa convergir para encontrar o ótimo global. As bases teóricas para tais algoritmos foram apresentadas nas décadas de 1960 e 1970 [38, 39, 40, 41, 42] e, desde então, novos mecanismos, variações e aprimoramentos tem sido propostos.

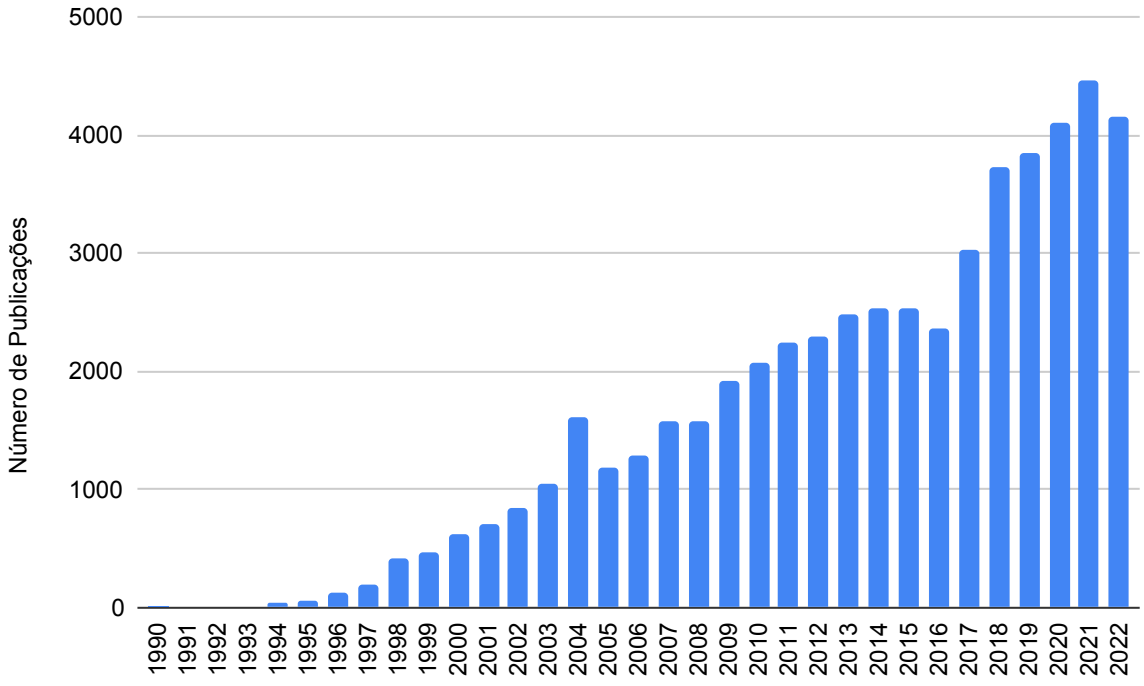
Esses algoritmos podem apresentar alto custo computacional, principalmente por se avaliar a função objetivo diversas vezes em uma dada população, ao longo das gerações. Atualmente, com o advento da computação de alto desempenho (processadores e *clusters* com cada vez mais núcleos) tem-se a computação paralela das funções objetivo, de modo que boa parte dessas dificuldades tem sido superadas em muitos casos [6, 8]. Obviamente,

[1] *"Não é o mais forte que sobrevive, nem o mais inteligente, mas o que melhor se adapta às mudanças."* – Leon C. Megginson.

para problemas de grande dimensão e alta complexidade, dificuldades relacionadas a essas questões sempre deverão ser avaliadas pelo projetista que está resolvendo o problema de otimização.

Por outro lado, o uso dessas ferramentas se justifica devido a algumas dificuldades numéricas e problemas de robustez muitas vezes encontrados por métodos determinísticos tais como: sensibilidade às condições iniciais, multimodalidade, problemas de continuidade e diferenciabilidade das funções a serem otimizadas e/ou de suas restrições, funções não convexas, existência de ruídos nas funções, necessidade de se trabalhar com valores discretos, dentre outras [6]. De forma geral, algoritmos evolutivos são capazes de lidar de forma efetiva com essas limitações, possuindo a capacidade de encontrar ótimos globais de funções com alto grau de complexidade, sem a exigência do cálculo de suas derivadas ou outras informações associadas às propriedades matemáticas dessas funções. A Figura 3.1 mostra o crescimento do número de publicações por ano encontradas no Portal Periódicos[2] ao se utilizar o termo de busca *"Evolutionary Algorithm"* considerando o período entre 1990 e 2022, evidenciando o interesse da comunidade científica pelo tema.

Figura 3.1: Número de publicações por ano encontradas no Portal Periódicos com o termo de busca *"Evolutionary Algorithm"*, entre 1990 e 2022.



Fonte: Elaborada pelo autor.

[2] https://www.periodicos.capes.gov.br/

3.2 A Evolução Biológica

Um ponto interessante ao se estudar AEs é a interdisciplinaridade com às Ciências Biológicas. Nesta seção, apresentaremos apenas as ideias mais básicas e fundamentais para a compreensão da evolução biológica. O leitor interessado em maiores detalhes históricos e conceituais sobre a descoberta desse importante fato pode se aprofundar em [8, 43, 44].

Em um ambiente ecológico, os seres vivos de uma mesma espécie competem entre si pela sobrevivência e pela reprodução. Esse conjunto de seres vivos que compartilham tal ambiente é chamado de *população*. No contexto da Biologia Moderna, a evolução biológica pode ser definida como a mudança gradual da composição genética média de uma determinada população de organismos, ao longo de sucessivas gerações [8]. Essas modificações ocorrem devido a combinação de operações evolutivas que consistem de: seleções, cruzamentos e mutações. De modo geral, entre duas gerações sucessivas, tais mudanças tendem a gerar pouco impacto. Contudo, ao longo de centenas ou milhares de gerações, ocorrem apreciáveis alterações morfológicas e metabólicas na população em questão e que podem resultar na diferenciação de uma nova espécie. A seguir, são apresentados de forma resumida e simplificada, aspectos fundamentais sobre a bioquímica da hereditariedade.

3.2.1 Codificando os Seres Vivos

Um ser vivo, indivíduo biológico, possui as informações relacionadas à hereditariedade armazenadas em um composto orgânico conhecido como ***á****cido* ***d****esoxirribo****n****ucleico* (ADN ou DNA, do inglês, ***d****eoxyribo****n****ucleic* ***a****cid*) [45]. O DNA se localiza no núcleo das células, sendo por essa razão um tipo de ácido nucleico, constituído por átomos de Hidrogênio, Oxigênio, Nitrogênio, Carbono e Fósforo, organizados em estruturas conhecidas como nucleotídeos. Assim, temos que os nucleotídeos do DNA possuem:

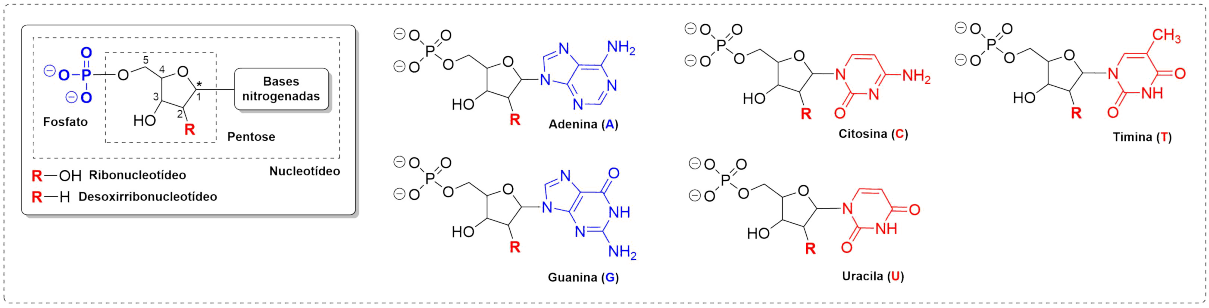
- uma base nitrogenada – pode ser **A**denina(**A**), **C**itosina(**C**), **G**uanina(**G**) ou **T**imina(**T**);
- um monossacarídeo (açúcar) constituído de cinco átomos de carbono (pentose) conhecido como desoxirribose;
- um grupo fosfato [46].

As bases nitrogenadas são assim denominadas devido à presença do átomo de Nitrogênio (N), sendo que as fórmulas químicas das cinco bases nitrogenadas existentes

estão mostradas na Figura 3.2. Dentre essas cinco bases, apenas a base **U**racila(**U**) não pode estar presente na estrutura do DNA. A **U**racila está presente em outro tipo de ácido nucléico conhecido como RNA (do inglês, ***ribo****nucleic* ***a****cid* ou ARN, ***á****cido* ***ribo****nucleico*) [45]. O RNA possui uma estrutura bastante similar à do DNA, com as seguintes diferenças principais:

- o monossacarídeo do RNA é a ribose, enquanto no DNA é a desoxirribose;
- as bases nitorgenadas do RNA são às mesmas do DNA, exceto que a **T**imina é substituída pela **U**racila, de modo que o RNA é codificado com **A**,**C**,**G** e **U** [46].

Figura 3.2: Fórmulas químicas das bases nitrogenadas.

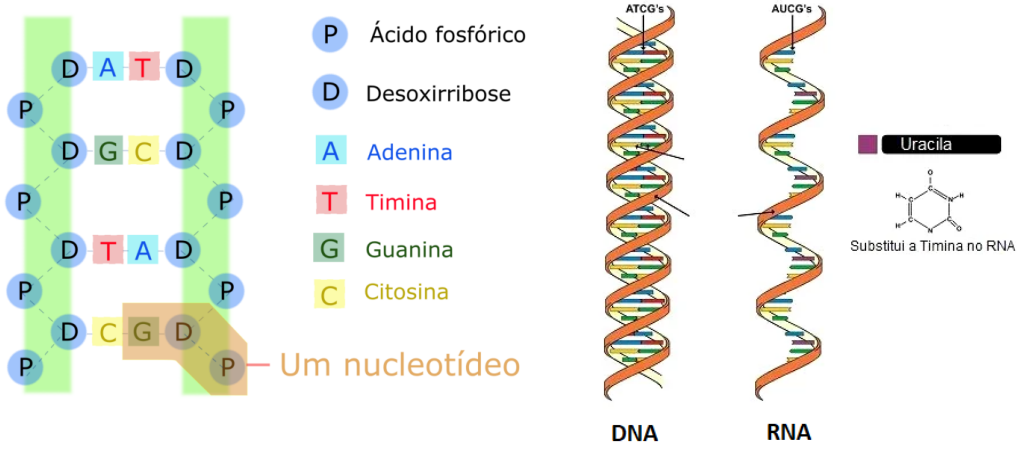


Fonte: https://pt.wikipedia.org/wiki/Base_nitrogenada.

Na estrutura do DNA, tem-se um esqueleto composto do grupo fosfato (P) e do açúcar desoxirribose (D) que se ligam e se alternam, de modo a formar uma dupla-hélice (ver Figura 3.3b). Portanto, o DNA é constituído por duas fitas ou duas cadeias que correspondem aos nucleotídeos pareados. Cada base nitrogenada (**A**,**C**,**G** ou **T**) está ligada a uma desoxirribose (D) que juntamente com um grupo fosfato formam, assim, cada nucleotídeo. As ligações entre os nucleotídeos estão relacionadas às bases nitrogenadas e vale destacar que o pareamento ocorre entre bases complementares, que devem respeitar a seguinte regra: **A** deve estar sempre pareada com **T** e **G** sempre ligada a **C**. Portanto, conhecendo uma das fitas, a outra estará determinada. No caso do RNA, o que modifica é que **A** deve estar sempre pareada com **U** [46].

Uma das importantes funções do DNA é produzir o RNA em um processo conhecido como transcrição. Nesse processo, a enzima RNA polimerase separa as duas fitas do DNA, quebrando as ligações químicas que as unem [45]. Assim, uma das fitas é utilizada como molde para a síntese de RNA. Portanto, diferentemente do DNA que possui fita dupla, o RNA possui apenas uma cadeia de nucleotídeos organizada em forma de fita simples,

Figura 3.3: Esquema representativos de trechos de DNA e RNA. Em (a), tem-se um trecho de DNA em dupla hélice e seus componentes, destacando-se um nucleotídeo, formado por uma base nitrogenada (neste caso, **G**uanina), um ácido fosfórico e a desoxirribose. Em (b), tem-se os formatos de dupla hélice do DNA e de fita simples do RNA, frisando-se a substituição de **U**racila por **T**imina.



(a) Fonte: https://www.todamateria.com.br/dna-e-rna/ (Adaptada).

(b) Fonte: https://www.todamateria.com.br/rna/.

conforme Figura 3.3b. Em um processo conhecido como síntese proteica, um conjunto de RNAs (mensageiro, transportador e ribossômico), inicialmente produzidos pelo DNA no processo de transcrição, são capazes de construir uma cadeia de aminoácidos que se dá o nome de cadeia polipeptídica ou proteínas [46]. As proteínas determinam a estrutura das células e controlam os processos metabólicos. Portanto, desde as diferentes possíveis cores da pele, dos olhos, tamanhos das partes do corpo, doenças, até o fato de algumas pessoas apesentarem a tendência de metabolismo mais acelerado estão relacionadas à produção de proteínas que são dependentes do DNA.

Na cadeia de RNA, uma sequência de três bases nitrogenadas específicas codificam um determinado aminoácido ou os pontos conhecidos como *start* e *stop*, termos *do inglês* que significam, respectivamente, início e fim de tradução - ou síntese - da cadeia proteica, processo por meio do qual o RNA produz as proteínas. Essa sequência é conhecida como códon [45]. Por exemplo, a glutamina ($C_5H_{10}N_2O_3$), um aminoácido bastante conhecido que atua como nutriente para células imunológicas e no crescimento muscular, é sintetizada no nosso organismo por meio dos códons **CAA** e **CAG**. Note que duas sequências distintas são responsáveis por produzir o mesmo aminoácido, o que ocorre também com outros aminoácidos. Um outro exemplo é o aminoácido serina que pode ser produzido por

quatro sequências distintas: **UCU**, **UCC**, **UCA** e **UCG**. Observe que caso a relação entre aminoácidos e códons fosse biunívoca, existiram 64 aminoácidos: 4 possibilidades para cada códon, o que pelo princípio fundamental de contagem, levaria a: $4 \cdot 4 \cdot 4 = 64$. Contudo, posto que algumas sequências geram os mesmos aminoácidos, na vida existente no planeta Terra que é baseada na química do carbono, temos apenas os 20 aminoácidos que são mencionados na Figura 3.4a. Assim, todos os seres vivos que conhecemos produzem e/ou se utilizam desses 20 aminoácidos que são codificados conforme os códons da Figura 3.4b. Por tal figura, note que *Metionina* **(AUG)** codifica o início da cadeia, enquanto os códons **UAA**, **UAG** e **UGA** são referidos como do tipo *stop*, ou seja, delimitam o fim da construção da cadeia polipetídica.

Figura 3.4: Aminoácidos e Códons. Em (a), tem-se a lista dos 20 aminoácidos existentes na natureza, enquanto em (b), apresenta-se uma tabela com a relação que codifica cada códon em seu respectivo aminoácido. O códon *Metionina* (**AUG**) sinaliza o início da tradução (*start*), enquanto os códons **UAA**, **UAG** e **UGA** são referidos como do tipo *stop*.

Arg – Arginina
Asn – Aspargina
Asp – Ácido aspártico
Cys – Cisteína
Gln – Glutamina
Glu – Ácido glutâmino
Gly – Glicina
His – Histidina
Ile – Isoleucina
Leu – Leucina
Lys – Lisina
Met – Metionina (códon de início)
Phe – Fenilalamina
Pro – Prolina
Ser – Serina
Stop – Códon de parada
Thr – Treonina
Tyr – Tirosina
Val – Valina

(a)

Segunda base do códon

Primeira base do códon	U	C	A	G	Terceira base do códon
U	UUU, UUC: Phe; UUA, UUG: Leu	UCU, UCC, UCA, UCG: Ser	UAU, UAC: Tyr; UAA stop; UAG stop	UGU, UGC: Cys; UGA stop; UGG Trp	U C A G
C	CUU, CUC, CUA, CUG: Leu	CCU, CCC, CCA, CCG: Pro	CAU, CAC: His; CAA, CAG: Gln	CGU, CGC, CGA, CGG: Arg	U C A G
A	AUU, AUC, AUA: Ile; AUG Met	ACU, ACC, ACA, ACG: Thr	AAU, AAC: Asn; AAA, AAG: Lys	AGU, AGC: Ser; AGA, AGG: Arg	U C A G
G	GUU, GUC, GUA, GUG: Val	GCU, GCC, GCA, GCG: Ala	GAU, GAC: Asp; GAA, GAG: Glu	GGU, GGC, GGA, GGG: Gly	U C A G

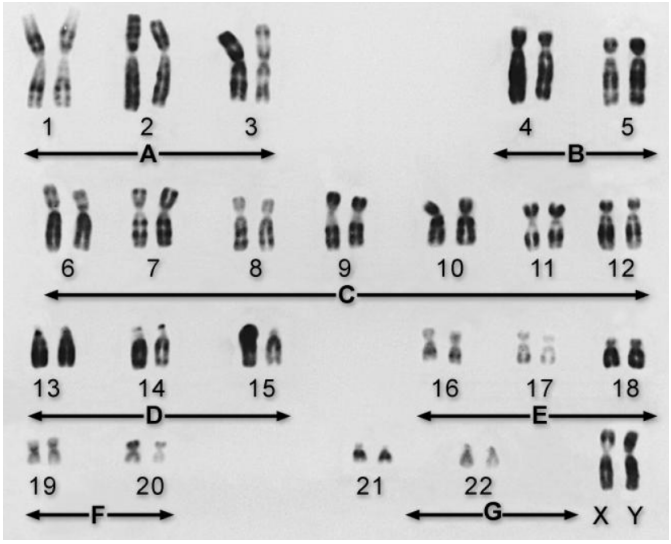
(b)

Fonte: https://www.todamateria.com.br/codigo-genetico/.

Na cadeia de DNA, uma sequência específica e ordenada de nucleotídeos que produz uma molécula de RNA é conhecida como *gene*, sendo este identificado por meio de sua sequência de bases nitrogenadas [45]. Como exemplo, suponha que a seguinte sequência de bases corresponda a um gene, **ACGTAC**. Portanto, um gene pode ser visto matematicamente como um vetor que utiliza elementos do conjunto {**A**,**C**,**G**,**T**}. Por sua vez, os genes são organizados em estruturas conhecidas como *cromossomos*. Em seres vivos complexos, um cromossomo tem de centenas a milhares de genes, sendo que a

localização fixa e específica de um gene no cromossomo é conhecida como *locus gênico* [46]. Assim, cada espécie possui um número específico de cromossomos, no caso das células haploides dos seres humanos, existem 23 cromossomos, enquanto nas diploides, tem-se 46 cromossomos como mostrado na Figura 3.5.

Figura 3.5: Exemplo de cariótipo – conjunto de cromossomos de uma célula. Nesse caso, tem-se uma célula diploide (23 pares - 46 cromossomos) de um homem adulto. O último par de cromossomos se relaciona ao sexo: XY corresponde ao sexo masculino, enquanto XX, ao feminino.



Fonte: https://phil.cdc.gov/Details.aspx?pid=14679.

Note que, na referida figura, os cromossomos estão pareados. Cromossomos que compõem o mesmo par são ditos *homólogos*, sendo que um deles veio da célula paterna (espermatozoide), enquanto o outro foi herdado da célula materna (ovócito) [45]. Nos seres humanos as células haploides correspondem as células reprodutivas (espermatozoide e ovócito), de modo que na fecundação, o zigoto recebe 23 cromossomos de cada uma das células dos pais, formando uma célula diploide de 46 cromossomos. As demais células diploides do organismo são formadas no processo de mitose, em que ocorre a replicação do DNA. Por outro lado, as células reprodutivas que são haploides se formam via meiose [46].

Nesse sentido, um cromossomo pode ser visto como um vetor em que posições específicas estão associadas a genes distintos. Em última instância, um indivíduo, tem seu *genótipo* ou *genoma*, o conjunto de todos os seus cromossomos [45], que também pode ser encarado como um vetor de bases nitrogenadas. Como vimos, os genes são trechos de DNA e, portanto, determinarão a produção de RNA que são os construtores de aminoácidos e, consequentemente, de proteínas. Da perspectiva de um determinado ser

vivo, um aminoácido pode ser classificado em essencial ou não essencial. Os aminoácidos são ditos essenciais quando são necessários para o funcionamento do organismo em questão mas não podem ser sintetizados por ele. Assim, tais aminoácidos devem ser obtidos de fontes externas ao organismo por meio da alimentação, por exemplo.

Dizemos que o conjunto de características morfológicas e funcionais que podem ser observadas em um organismo é o *fenótipo* [8]. O fenótipo é resultado da interação do genótipo com o ambiente em que o indivíduo vive. Na Figura 3.6, tem-se diferentes indivíduos que exibem variados fenótipos em aspectos de coloração, padrões e formato das conchas na espécie de molusco *Donax variabilis*.

Figura 3.6: Quinze indivíduos da espécie de molusco *Donax variabilis* com variações fenotípicas quanto a padrões de formato e coloração.

Fonte: https://en.wikipedia.org/wiki/Phenotype.

A dependência do ambiente ocorre, dentre outros fatores, pela necessidade de o organismo obter aminoácidos essenciais e fontes de energia que irão impactar em seu metabolismo e desenvolvimento [44]. Além disso, o indivíduo molda diversos aspectos comportamentais e até mesmo fisiológicos de acordo com suas reações às restrições e aos estímulos impostos pelo ambiente. Um exemplo disso é a cor da pele dos seres humanos. Obviamente, essa coloração está associado a genes específicos que contém as instruções para a produção dos aminoácidos que irão determinar tal cor. Contudo, a incidência de luz solar sobre a pela tende a estimular a produção de uma proteína conhecida como melanina que gera o escurecimento da pele e confere maior proteção contra os efeitos da radiação

ultravioleta. Portanto, essa coloração irá depender tanto dos genes vinculados quanto dos estímulos do ambiente, neste caso, a incidência de luz solar sobre a pele.

Um outro conceito importante corresponde ao que conhecemos como genes *alelos*. Eles são formas alternativas de um mesmo gene que estão expressas aos pares em cromossomos homólogos, possuindo a mesma localização na extensão cromossômica, ou seja mesmo locus gênico [45]. Alelos distintos podem resultar em diferentes características fenotípicas, como por exemplo, a pigmentação do cabelo. Os alelos podem ser classificados em recessivos e dominantes. Um alelo dominante, independente de seu par, determinará uma característica hereditária. Por outro lado, alelos recessivos determinam sua característica hereditária somente se formarem um par com um outro alelo idêntico a ele [46]. Para exemplificar, consideremos o fenótipo *presença de chifres* em bovinos. Identifica-se como M o alelo dominante associado a ausência de chifres, nesse caso o bovino é chamado mocho. Por outro lado, seja m o alelo recessivo associado a presença de chifres. A cada possível par genético, associa-se uma das possíveis características fenotípicas como mostrado na Tabela 3.1. Note que a presença de chifres só ocorre no par mm devido ao fato de o alelo m ser recessivo.

Tabela 3.1: Presença de chifres em bovinos de acordo com os pares genéticos.

Par Genético	**Fenótipo**
MM	mocho
Mm	mocho
mM	mocho
mm	chifre

A seguir, iremos introduzir o importante conceito de Seleção Natural que é creditado a Charles Robert Darwin, importante cientista do século XIX. Além disso, iremos apresentar a Teoria Moderna da Evolução das Espécies que envolve aspectos de pressão seletiva com aspectos geradores de diversidade relacionadas ao que foi visto até aqui.

3.2.2 A Seleção Natural

Os seres vivos possuem instintos básicos atrelados à reprodução de modo que se todos eles reproduzissem, as populações cresceriam exponencialmente [46]. Contudo, em qualquer ambiente em que uma população se desenvolve, temos que os recursos são limitados e, portanto, algum tipo de *pressão seletiva* é inevitável [8]. Em outras

palavras, os indivíduos de uma população irão competir para sobreviver e também para reproduzir, transmitindo seus genes para as próximas gerações. Cada indivíduo possui uma probabilidade de sobreviver e uma probabilidade de reproduzir, sendo que ambas estão relacionadas às suas características fenotípicas, tais como: agilidade, força, aparência física, capacidade de camuflagem, dentre outras [44]. Como os recursos são escassos, apenas um subconjunto formado por indivíduos mais adaptados ao ambiente, isto é, capazes de competir por esses recursos de forma mais efetiva, irá sobreviver e gerar descendentes para as próximas gerações. Como foi mencionado, a sobrevivência depende das características fenotípicas que determinam o grau ou a medida de *aptidão/adaptação* (*em inglês, fitness*) do indivíduo ao meio e, por sua vez, estão relacionadas ao genótipo [8].

No processo de reprodução, os filhos herdam os genes dos pais e, portanto, existe uma tendência de se preservar características favoráveis à sobrevivência naquele meio, visto que os mais aptos apresentam maior probabilidade de reproduzir [44]. Contudo, como explicar o surgimento de novas características? Isso ocorre como resultado das reproduções e mutações sucessivas, introduzindo o fator *diversidade* de forma persistente ao longo das gerações.

3.2.3 Diversidade

No contexto evolutivo, o conceito de *diversidade* está relacionado às variações no código genético que ocorrem ao longo das gerações e, assim, provocam modificações em características fenotípicas [8]. Existem, basicamente, três importantes processos que atuam como fontes de variabilidade genética que serão apresentados a seguir.

Cruzamento sexuado: nesse tipo de reprodução, tem-se a participação de dois gametas, um masculino e um feminino. Já na produção dos gametas, tem-se o aumento da variabilidade genética devido à recombinação gênica. Nesse processo, ocorre a quebra dos cromossomos e a troca de partes de material genético entre partes dos cromossomos conhecidas como cromátides [45]. A *recombinação gênica* também é conhecida como *crossing over* e, assim, os cromossomos originais se recombinam, de modo que cada um agora transporta uma parte dos genes do outro. Além disso, na fecundação, tem-se a união do gameta masculino com o gameta feminino, formando um novo indivíduo em que o código genético é o resultado de uma combinação dos genes do gameta masculino e do gameta feminino, gerando, portanto, uma nova

combinação de genes [46].

Mutação: ocorre ao acaso e permite o surgimento de novos alelos nos indivíduos. Tal surgimento está relacionado à inserção, troca ou até mesmo deleção de algum nucleotídeo do DNA, de modo que a sequência de bases é alterada [45]. Com a alteração da sequência, tem-se alteração na molécula de RNA e, consequentemente, alterações na proteína codificada. Como as proteínas estão intimamente relacionadas às características fenotípicas, pode ocorrer alterações que geram vantagens ou desvantagens para a sobrevivência naquele ambiente [8]. Note que, sendo este processo aleatório, a seleção natural irá atuar para preservar as características vantajosas para a sobrevivência ou para o cruzamento.

Fluxo gênico: consiste na movimentação de genes de uma população para a outra. Essa movimentação acaba por inserir um novo gene ou reintroduzir um gene em uma população, aumentando a variabilidade [45]. Um exemplo ocorre no contexto da migração de um determinado indivíduo para outro meio, de modo que, ao reproduzir com elementos da população desse novo meio, tem-se o aumento de variabilidade.

3.2.4 Síntese da Teoria Evolutiva

A evolução biológica é um processo que pode ser sintetizado como a combinação da seleção natural com os elementos responsáveis pela diversidade [8]. Mutações e cruzamentos tendem a aumentar a variabilidade, enquanto a pressão seletiva faz o papel de preservar os indivíduos mais adaptados à sobrevivência naquele meio. Assim, em uma determinada população, alguns indivíduos irão conseguir sobreviver e reproduzir, passando seus genes para as próximas gerações. Na reprodução, tem-se a combinação de gametas, aumentando a variabilidade. Além disso, a mutação que ocorre aleatoriamente pode gerar novas características, sejam elas vantajosas ou não para a sobrevivência do indivíduo. Como a sobrevivência e o fato de ele reproduzir ou não estão associados a aspectos fenotípicos que, por sua vez, dependem do genótipo do indivíduo, aqueles indivíduos que possuem características genéticas vantajosas terão maior probabilidade de sobreviver e reproduzir [44]. Com o passar das gerações, tem-se a tendência de que as características genéticas mais vantajosas para a sobrevivência naquele meio sejam preservadas.

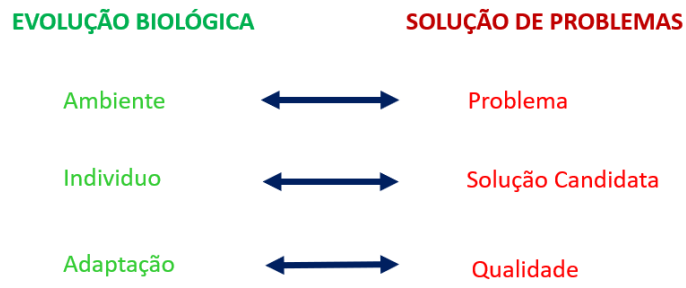
Os fatores de diversidade são muito importantes porque espécies muito especializadas a um determinado ambiente sofrerão maiores dificuldades de sobrevivência conforme,

naturalmente, existe a tendência de ocorrer alterações no meio. Tais alterações podem estar relacionadas à disponibilidade de alimentos, características físicas, químicas, biológicas e ecológicas que tipicamente acontecem com o passar do tempo.

3.3 Conceitos Básicos em Computação Evolucionária

Tendo apresentado as ideias fundamentais relativas à evolução biológica, o que se pretende nesta seção é traduzir os conceitos biológicos para o contexto computacional. A metáfora fundamental que relaciona a evolução biológica com a solução de problemas usando computação evolucionária está apresentada na Figura 3.7. No contexto de solução de problemas de otimização via estratégias evolutivas, deseja-se obter a solução candidata que apresente a maior qualidade ao resolver o problema. O conceito de qualidade se relaciona ao de adaptação no contexto da evolução biológica, à medida em que ambos estão associados a probabilidades. O primeiro concerne à probabilidade de uma solução ser mantida e usada como semente para a construção de novas soluções para o problema de otimização, enquanto o segundo consiste na probabilidade de um indivíduo sobreviver e reproduzir no ambiente em que vive [8].

Figura 3.7: Metáfora fundamental da Computação Evolucionária.



Fonte: Eiben e Smith[8] (Adaptada).

Ao longo das décadas de 1960 e 1970, três diferentes implementações inspiradas na evolução biológica foram desenvolvidas por diferentes grupos de pesquisa. Nos Estados Unidos, os pesquisadores Fogel, Owens e Walsh introduziram as ideias do que ficou conhecido como *Programação Evoulcionária* [38], enquanto John Holland propôs um método que ficou conhecido como *Algoritmo Genético* [39]. Por outro lado, na Alemanha, Rechenberg e Schwefel desenvolveram as bases do que conhecemos como *Estratégias Evolutivas* [41, 42, 47]. Até a década de 1990, essas três áreas se desenvolveram de forma independente, contudo, a partir de então, elas passaram a ser vistas pelos pesquisadores

como diferentes sub-áreas de um mesmo campo cientifico-tecnológico, a *Computação Evolucionária* [8]. Assim, os algoritmos desenvolvidos dentro deste campo de pesquisa são conhecidos como *Algoritmos Evolutivos* (AEs). Nesses algoritmos, uma população de indivíduos é inicializada e evolui sucessivamente para regiões melhores no espaço de busca por meio de um processo estocástico em que ocorrem seleções, mutações e, em alguns casos, recombinação ou cruzamento. As diferentes formas nas quais os AEs podem ser implementados resultam em diferentes paradigmas dentro da Computação Evolucionária, descritos a seguir:

Algoritmos Genéticos (AGs), que replicam a evolução genética em um contexto algorítmico. De forma simplificada, tem-se uma população de soluções candidatas (pontos do espaço de busca ou indivíduos), em que cada indivíduo é codificado utilizando-se uma representação conhecida como cromossomo. Os indivíduos são avaliados via função de adaptação (fitness) que incorpora matematicamente o problema de otimização a ser resolvido e associa os maiores valores aos indivíduos que representam melhores soluções do problema de otimização. Nesse sentido, é possível comparar indivíduos entre si. Indivíduos mais adaptados apresentam maior probabilidade de cruzamento, gerando filhos que recebem um cromossomo resultante da mistura de cromossomos dos pais. A população então sofre mutações, variações aleatórias em seus cromossomos, e os indivíduos mutantes são também avaliados. Ao final, ocorre a seleção de sobreviventes ou substituição, em que um subconjunto de indivíduos da geração atual é selecionado para se repetir o processo na próxima geração. Nesse contexto, os indivíduos mais adaptados tipicamente apresentam maior probabilidade de sobrevivência. Tais algoritmos serão tratados com mais detalhes na Seção 3.4.

Programação Genética, que é baseada em AGs, servindo como uma técnica para realizar o processo de programação de computadores de forma automática. Nesse sentido, os indivíduos são programas [48].

Programação Evolucionária , que é derivada da simulação do comportamento adaptativo na evolução para desenvolver uma forma automática de programação via *máquinas de estado finito*. A ideia inicial é obter a evolução de uma população de máquinas que seriam avaliadas por uma função de custo quanto à capacidade de, ao receber uma sequência de símbolos, predizer corretamente o próximo símbolo da sequência

[38]. As máquinas sofrem variações aleatórias (mutações) e são novamente avaliadas, ocorrendo a preservação de um subconjunto de máquinas mais adaptadas.

Estratégias de Evolução, que visam obter, a partir de mecanismos evolutivos, além da solução do problema de otimização, os parâmetros que controlam os mecanismos evolutivos do algoritmo, como por exemplo, a probabilidade de mutação [47]. Nesse sentido, ocorre o processo conhecido como auto-adaptação dos parâmetros [49].

Evolução Diferencial (ED), bastante similar aos algoritmos genéticos, diferindo-se nos mecanismos de atualização utilizados, que não são inspirados na recombinação genética [50]. Este algoritmo será tratado com mais detalhes na Seção 3.5

Evolução Cultural, em que se modela a evolução da cultura de uma população e como esta cultura influencia a evolução das características genotípicas e fenotípicas dos indivíduos [51].

Co–Evolução, nos quais a medida de adaptação de um indivíduo é subjetiva, isto é, os indivíduos são avaliados baseados em suas interações com outros indivíduos [52]. Nesse sentido, existem dois grupos principais de algoritmos, baseados em: i) cooperação ou ii) competição. No primeiro caso, indivíduos são premiados quando trabalham bem com outros indivíduos e punidos quando essas interações são menores. No caso competitivo, os indivíduos são recompensados as custas daqueles com os quais interagem. Em ambos os casos, a ideia básica é que os indivíduos interajam via cooperação ou competição, de modo que o grupo adquira as características necessárias para sobreviver [52].

Conforme já foi apresentado, os AEs se utilizam de uma *população* de indivíduos, em que cada indivíduo representa um ponto de busca no espaço de soluções potenciais de um problema de otimização. É importante mencionar que esses algoritmos funcionam de forma iterativa, sendo que as iterações são conhecida, no contexto de AEs, como *gerações*. Assim, definido um número de gerações $N_{\text{ger}} \in \mathbb{N}$, a população em uma dada geração $0 \leq k < N_{\text{ger}}$ produz descendentes que, submetidos a algum critério de elitismo, são selecionados para a geração $k+1$.

Matematicamente, cada indivíduo é um vetor $\vec{x}_{k,n}$ associado a uma geração ou iteração $0 \leq k < N_{\text{ger}}$, e que pertence a um conjunto de $N_{\text{pop}}(k) \in \mathbb{N}$ vetores, $1 \leq n \leq N_{\text{pop}}(k)$,

ou seja, $N_{\text{pop}}(k)$ é o tamanho da população na iteração k a qual é submetida aos processos de atualização inspirados na evolução biológica. Note que o tamanho da população $N_{\text{pop}}(k)$ pode, a princípio, variar de acordo com a geração k.

Seja um problema de otimização dado na forma de (2.9) e descrito por $m \in \mathbb{N}$ variáveis de decisão: $x_1, x_2, \ldots, x_{m-1}, x_m$. Dizemos que o ponto ou solução candidata do espaço de busca $\vec{x}_{k,n} = (x_1, x_2, \ldots, x_m)$ é o *indivíduo* que pode ser codificado no *cromossomo* ou *genoma* ou *genótipo*, $\bar{x}_{k,n} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_m)$. Note que existe uma relação biunívoca entre cromossomo e indivíduo, de modo que $\bar{x}_{k,n}$ está associado unicamente a $\vec{x}_{k,n}$. Nesse sentido, o cromossomo $\bar{x}_{k,n}$ possui a informação das variáveis do problema codificada nos $m \in \mathbb{N}$ *genes*, de modo que, cada gene $\bar{x}_j$, $1 \leq j \leq m$, está associado a variável de decisão x_j, tipicamente expressa em base dois ou na forma de um número real em base decimal. No primeiro caso, a codificação é dita binária e no segundo, real. A codificação binária será apresentada com detalhes na Subseção 3.4.1. A codificação real pode ser feita simplesmente considerando $\bar{x}_j = x_j$, $j \in \{1, \ldots, m\}$. Note que, nesse caso, cada gene corresponde diretamente a uma variável de decisão x_j do problema de otimização. Assim, no contexto da codificação real, por simplicidade pode-se tratar $\vec{x}_{k,n} = \bar{x}_{k,n}$.

O espaço ou conjunto que contém todas os possíveis $\bar{x}_{k,n}$ é o espaço genotípico. Cada cromossomo $\bar{x}_{k,n}$ pertence a um indivíduo $\vec{x}_{k,n}$ que está associado ao valor da função de custo neste ponto, $f(\vec{x}_{k,n})$. A função f a ser minimizada, no contexto de AEs, está associada a uma outra função que recebe o nome *fitness* (*do inglês*, adaptação), $\bar{f}$. O objetivo de minimizar f deve equivaler à maximização de $\bar{f}$. Ou seja, indivíduos que correspondem a valores de $\vec{x}_{k,n}$ que minimizam f devem apresentar alta pontuação de fitness, $\bar{f}(\bar{x}_{k,n})$. Portanto, a relação entre a fitness e a função a ser minimizada é que a função fitness é construída de modo a refletir o quão bem um determinado indivíduo apresenta desempenho na resolução do problema de otimização. Além disso, enquanto o argumento da função objetivo f é um indivíduo qualquer da população $\vec{x}_{k,n}$, o argumento da fitness é o cromossomo deste indivíduo, $\bar{x}_{k,n}$. Desse modo, a fitness é uma medida ou uma nota que podemos atribuir a um indivíduo, medindo sua adaptação à estrutura (curva, superfície, variedade, etc) que queremos otimizar. Às vezes, é possível usar diretamente a função que se deseja otimizar como função fitness, no entanto, em problemas mais complexos, pode ser necessário fazer ajustes. As definições de fitness apresentadas a seguir podem não ser apropriadas para alguns problemas, de modo que, nesses casos, o projetista

deverá realizar adaptações. Nesse sentido, é importante reiterar que a escolha da função fitness é uma etapa crucial para os algoritmos e depende da natureza do problema. Dado um problema de maximização da função objetivo f, podemos definir $\bar{f}(\bar{x}_{k,n}) = f(\vec{x}_{k,n}) - \beta(k)$, em que $\beta(k)$ corresponde ao menor valor de f dentro da população da geração atual k. Para o contexto de minimização, podemos definir a fitness como uma transformação simples de f como se segue

$$\bar{f}(\bar{x}_{k,n}) = -\big(f(\vec{x}_{k,n}) - \beta(k)\big) - \bar{\beta}(k), \tag{3.1}$$

sendo $\bar{\beta}(k)$ o menor valor de $-\big(f(\vec{x}_{k,n}) - \beta(k)\big)$ na geração k. Assim, em ambos os casos, tem-se que a fitness $\bar{f}$ é não negativa e maximizá-la equivale a resolver o problema de otimização para a função objetivo f. De forma similar ao espaço genotípico, o espaço ou conjunto que contém todos os possíveis valores de $\bar{f}(\bar{x}_{k,n})$ é conhecido como espaço fenotípico.

Ao longo de uma geração, dois processos básicos ocorrem e serão detalhados mais adiante, sendo eles: o *cruzamento* e a *mutação*. O cruzamento consiste na troca de informações genéticas entre indivíduos da população e a mutação consiste em uma perturbação que introduz alguma característica aleatória nos indivíduos. O número de cruzamentos, $N_c(k) \in \mathbb{N}$, pode ser determinado por meio da probabilidade de cruzamento $P_c \in [0,1]$ que é um meta-parâmetro do algoritmo e deve ser definido pelo usuário, $N_c(k) = P_c N_{\text{pop}}(k)$. Note que o número de cruzamentos $N_c(k)$ também pode depender da geração k. No contexto da codificação real, uma opção de combinação de material genético de dois pais distintos, $\vec{x}_{k,i}$ e $\vec{x}_{k,j}$, $i \neq j$, para gerar o p-ésimo filho $\vec{y}_{k,p}$, $p \in \{1,\ldots,N_c(k)\}$, é o *cruzamento aritmético* que consiste de uma combinação convexa (2.1) dos genes dos pais

$$\vec{y}_{k,p} = \alpha\vec{x}_{k,i} + (1-\alpha)\vec{x}_{k,j}, \tag{3.2}$$

sendo $\alpha \in [0,1]$ um parâmetro que determina a proximidade do filho a cada um dos pais no espaço genotípico, em que tal proximidade pode ser medida via norma euclidiana. Assim, $\alpha \approx 0$ produz um filho mais próximo do pai $\vec{x}_{k,i}$, isto é, $\|\vec{x}_{k,i} - \vec{y}_{k,p}\| < \|\vec{x}_{k,j} - \vec{y}_{k,p}\|$. Por outro lado, $\alpha \approx 1$ produz um filho mais próximo do pai $\vec{x}_{k,j}$, $\|\vec{x}_{k,j} - \vec{y}_{k,p}\| < \|\vec{x}_{k,i} - \vec{y}_{k,p}\|$. O parâmetro α pode ser definido de forma fixa ou sorteado a partir de uma distribuição de probabilidade, como por exemplo, a distribuição uniforme. Outra possível variação é

utilizar valores distintos de parâmetro para cada gene.

Na codificação real, a *mutação* simples pode ser vista como uma escolha de uma direção de busca aleatória, sendo dado um passo nessa direção. Como no cruzamento, o primeiro passo é definir o número de mutações $N_m(k) \in \mathbb{N}$ que é dependente do meta-parâmetro probabilidade de mutação $P_m \in [0,1]$, $N_m(k) = P_m N_{\text{pop}}(k)$. Note que o número de mutações $N_m(k)$ também pode depender da geração k. Assim, um indivíduo mutante $\vec{z}_{k,p}$, $p \in \{1,\ldots,N_m(k)\}$, resultado da mutação de um indivíduo $\vec{x}_{k,i}$ pode ser obtido como

$$\vec{z}_{k,p} = \vec{x}_{k,i} + \vec{t}_{k,p}, \tag{3.3}$$

em que $\vec{t}_{k,p} = (t_1(k), t_2(k), \ldots, t_m(k))$, sendo $t_j(k) \sim \mathcal{P}(0, \sigma_j(k)^2)$. Cada valor $t_j(k)$ provém de uma variável aleatória de distribuição de probabilidade $\mathcal{P}$ cuja média é nula e a variância é dada por $\sigma_j(k)^2$. Note que o vetor de meta-parâmetros que correspondem aos desvios-padrões da distribuição adotada, $\vec{\sigma}_k = (\sigma_1(k), \sigma_2(k), \ldots, \sigma_m(k))$, também devem ser inicializados e podem depender da geração k. Conforme foi mencionado, a mutação consiste em escolher uma direção aleatória e dar um passo nessa direção. Tipicamente, a distribuição $\mathcal{P}$ é escolhida como a distribuição gaussiana, também conhecida como normal, ($\mathcal{N}$) ou uniforme ($\mathcal{U}$) [8]. No paradigma de *Estratégias de Evolução*, os meta-parâmetros de mutação também fazem parte do cromossomo de um indivíduo, de modo que eles também evoluem ao longo das gerações [49].

Dados o conjunto de candidatos a pais $\mathrm{X}_k = \{\vec{x}_{k,i}\}_{i=1}^{N_{\text{pop}}(k)}$, de filhos $\mathrm{Y}_k = \{\vec{y}_{k,i}\}_{i=1}^{N_c(k)}$ e de indivíduos mutantes $\mathrm{Z}_k = \{\vec{z}_{k,i}\}_{i=1}^{N_m(k)}$, deverão ser selecionados aqueles indivíduos de $\mathrm{D}_k = \mathrm{X}_k \cup \mathrm{Y}_k \cup \mathrm{Z}_k$ que irão compor a próxima geração de pais, X_{k+1}. Essa fase é conhecida como *seleção dos sobreviventes*. Para selecionar os indivíduos sobreviventes algum critério de *elitismo* deve ser utilizado, isto é, a informação contida em alguns indivíduos mais bem adaptados deve ser preservada para a próxima geração. Isso é importante porque AEs não se constituem como uma busca meramente aleatória, sendo esta última tipicamente incapaz de resolver uma grande parte de problemas complexos [53]. Existem tanto critérios de elitismo fortes quanto critérios mais relaxados. O caso mais relaxado possível, ocorre quando apenas o indivíduo mais adaptado em D_k é garantidamente preservado em X_{k+1}, isto é

$$\vec{x}_{k+1,1} = \arg\min_{\vec{x}} f(\vec{x}) \text{ sujeito a } \vec{x} \in D_k, \tag{3.4}$$

sendo que os outros indivíduos que irão compor X_{k+1} serão selecionados aleatoriamente do conjunto $\mathrm{D}_k - \{\vec{x}_{k+1,1}\}$, de modo que X_{k+1} tenha uma população de tamanho $N_{\mathrm{pop}}(k+1)$. Por outro lado, o caso mais elitista ocorre quando todos os elementos de X_{k+1} são selecionados a partir da ordem crescente dos valores da função de adaptação, de modo, que são selecionados todos os mais bem adaptados. Embora, a primeira vista possa parecer razoável utilizar este elitismo forte, Eiben e Smith[8] argumentam que essas posturas podem levar a uma convergência prematura, isto é, o método acaba preso em um mínimo local, não explorando o espaço de busca de forma apropriada. Nesse sentido, deve-se determinar o número de indivíduos $N_{\mathrm{elt}}(k) \in \mathbb{N}$ que serão selecionados para a próxima geração X_{k+1} a partir de um critério elitista e compor o restante dos $N_{\mathrm{pop}}(k+1) - N_{\mathrm{elt}}(k)$ indivíduos da população de forma aleatória.

Outras formas de seleção de sobreviventes são destacadas a seguir [6, 8], considerando μ indivíduos pais de uma geração que produzem uma quantidade λ de descendentes por mutação ou cruzamento no contexto de Estratégias Evolutivas:

- Dois membros (1+1): esquema simplificado em que existe apenas um indivíduo gerador na k-ésima iteração, $\vec{x}_k$, o qual é submetido a um processo de mutação, e gera um novo descendente, $\vec{y}_k$. Ambos indivíduos são comparados em termos da adaptação, de modo que o mais adaptado é selecionado

$$\vec{x}_{k+1} = \begin{cases} \vec{x}_k & \text{se } f(\vec{x}_k) \leq f(\vec{y}_k) \\ \vec{y}_k & \text{se } f(\vec{y}_k) < f(\vec{x}_k) \end{cases}. \tag{3.5}$$

- Multi-membros ($\mu + \lambda$): nesse caso, tem-se, inicialmente, μ indivíduos em uma geração, sendo produzida uma quantidade λ de novos indivíduos por mutação ou cruzamento. Ao fim de cada geração, são selecionados μ indivíduos para a próxima geração a partir de uma quantidade de $\mu + \lambda$ indivíduos.

- Multi-membros (μ, λ): a partir de λ indivíduos, μ serão selecionados, sendo ($\lambda \geq \mu$).

Note que os procedimentos de selecionar os pais para cruzamento, realizar o cruzamento, realizar a mutação, selecionar os sobreviventes são repetidos durante um certo número de gerações até que algum critério de parada seja satisfeito. Os critérios de parada tipicamente utilizados estão tipicamente relacionados ao número de gerações e à estagnação de um índice de melhoria conforme está mencionado nas Subseções 2.3.4, 3.4.7 e 3.5.3.

Tendo o critério de parada sido satisfeito, o indivíduo com melhor valor da função fitness obtido ao longo do processo de busca é a solução do problema de otimização. É válido relembrar aqui que, embora sejam técnicas úteis e, por isso, bastante difundidas no meio científico-tecnológico para a resolução de problemas, Algoritmos Evolutivos não garantem que a solução ótima será encontrada. Dependendo da população inicial, dos parâmetros e dos próprios aspectos aleatórios embutidos no algoritmo, pode ser que a população não convirja para o ótimo global, encontrando, ao final do caminho, algum mínimo local. Tudo isso tem a ver com as características não determinísticas de algoritmos evolutivos que ficarão ainda mais evidentes conforme detalharemos os processos e os operadores na sequência do capítulo. Problemas em que a a função objetivo apresenta platôs consideravelmente grandes e muitos mínimos locais, tendem a evidenciar tais aspectos. Em geral, é importante permitir que o algoritmo excursione bem a função objetivo, de forma a se conhecer as várias bacias de atração. Somente após isso é recomendável a especialização da população convergindo para o ótimo local dessa bacia que se espera ser o ótimo global do problema. Para ilustrar o funcionamento de um algoritmo evolutivo, apresenta-se o seguinte exemplo.

Exemplo 3.3.1: Considere o problema de otimização com restrições que consiste em resolver o seguinte problema de duas variáveis $\vec{x} = (x_1,x_2)$:

$$\vec{x}^* = \arg\min_{\vec{x}} f(\vec{x}) = 20 + x_1^2 + x_2^2 - 10\cos(2\pi x_1) - 10\cos(2\pi x_2) \text{ sujeito a } \vec{x} \in D,$$

$$D = \left\{ x_m \in [-5{,}12;\ 5{,}12], \quad m = 1{,}2 \middle| \begin{array}{rcll} g_i(\vec{x}) = \text{sen}(2\pi x_i) + 0{,}5 & \leq & 0, & i = 1{,}2 \\ h_j(\vec{x}) = \cos(2\pi x_i) + 0{,}5 & = & 0, & j = 1{,}2. \end{array} \right\}.$$

Tal problema tem característica multimodal e apresenta um minimizador global em $(-1/3, -1/3)$. Este problema consiste da Função de Rastringin em um contexto restrito, sendo que essa função no contexto de problemas irrestritos será considerada no Capítulo 5.

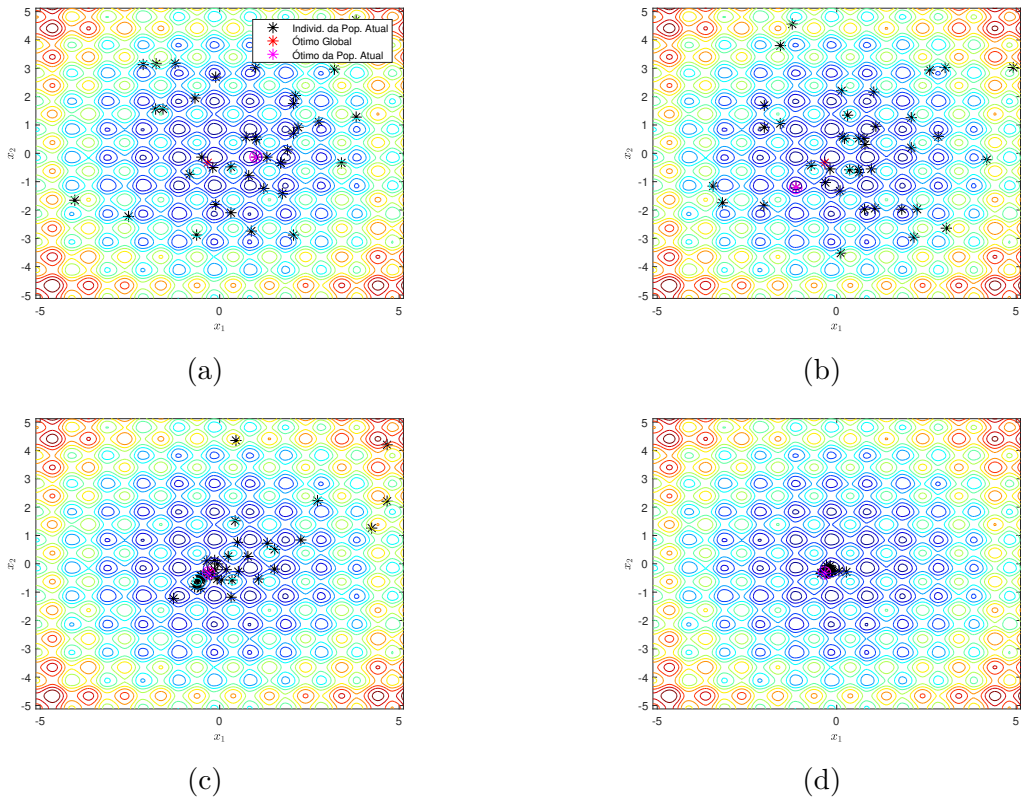
Um problema com restrições pode ser transformado em um problema irrestrito considerando o método dos multiplicadores de Lagrange [17, 34]:

$$\begin{aligned} \vec{x}^* = \arg\min_{\vec{x}} & f(\vec{x}) + \sum_{i=1}^{2} \mu_i \max(0, g_i(\vec{x})) + \sum_{i=1}^{2} \lambda_j |h_j(\vec{x})| \\ & \text{sujeito a } x_m \in [-5{,}12;\ 5{,}12], \quad m = 1{,}2, \end{aligned} \tag{3.6}$$

sendo que os parâmetros $\mu_i = 5$ e $\lambda_j = 5$ para i, $j = 1$, 2, são conhecidos como multi-

plicadores de Lagrange e penalizam, respectivamente, as restrições de desigualdade e de igualdade. Note que, quando as restrições são satisfeitas as parcelas relativas a tais penalizações têm contribuição nula. Por outro lado, quanto mais essas condições são violadas, maior se torna a contribuição dessas parcelas de penalização. Utilizou-se um AE com codificação real, cruzamento aritmético, mutação como uma perturbação via distribuição de probabilidade normal e preservação de um percentual dos mais bem adaptados. Além disso, os meta-parâmetros P_c, α, P_m, $\vec{\sigma}_k$, N_{elt} são adaptados dinamicamente. Na Figura 3.8, tem-se a evolução da população sobre as curvas de nível do problema transformado (3.6) ao longo de 100 gerações.

Figura 3.8: Evolução da população sobre as curvas de nível do problema transformado (3.6). Considera-se as 100 gerações de execução do AE utilizado neste exemplo. Cada imagem retrata a população em um dos estágios da evolução do algoritmo, especificamente, na k-ésima geração. Em (a), tem-se $k = 0$; (b), $k = 24$; (c), $k = 49$ e (d), $k = 99$.



Fonte: Elaborada pelo autor.

Neste AE, são realizadas 4000 avaliações da função objetivo, considerando uma população de 40 indivíduos por geração, ao longo de 100 gerações. Note, na Figura 3.8, que os asteriscos pretos representam a população de indivíduos que está distribuída sobre

a região de busca. Devido as seleções, cruzamentos e mutações, a população percorre a região de busca ao longo das gerações, com objetivo de encontrar o ótimo global do problema. Nas gerações iniciais, prevalecem os elementos geradores de diversidade de forma a varrer as diversas bacias de atração. Na fase final do algoritmo, busca-se o refinamento das soluções, assim, a população converge para o ótimo populacional que, neste caso, está muito próximo do ótimo global. Além disso, perceba que caso a região de busca não fosse adequadamente explorada na parte inicial do algoritmo, poderia se realizar o refinamento das soluções em uma das bacias de atração que não contém o ótimo global do problema, levando a convergência prematura para um ótimo local.

3.4 Algoritmos Genéticos

Algoritmos Genéticos (AGs) são algoritmos de otimização global pertencentes a classe dos AEs que, baseados na Metáfora Fundamental da Computação Evolucionária (Figura 3.7), buscam replicar os mecanismos da evolução biológica, sejam aqueles geradores de variabilidade, sejam os mecanismos que visam a seleção dos mais adaptados. As primeiras pesquisas que propuseram e utilizaram AGs para resolver problemas de otimização remetem a década de 1970 e são atribuídas a John Holland e seu grupo de pesquisa na Universidade de Michigan. O objetivo inicial era simular e estudar o comportamento adaptativo dos seres vivos. Após anos de pesquisa, os resultados e ideias desenvolvidos por Holland foram gradualmente refinados, sendo publicados no importante livro para a área, *Adaptation in Natural and Artificial Systems* [40]. Em 1989, David Goldberg, aluno de Holland, popularizou o método ao resolver um problema complexo de engenharia que envolvia o controle de transmissão de um gás em uma tubulação [54]. A partir de então, os algoritmos tem sido utilizados em diversos problemas de otimização. Para Golberg[54], existem quatro aspectos fundamentais que diferem os AGs dos métodos tradicionais de busca e otimização:

- AGs utilizam alguma codificação do conjunto de parâmetros, não trabalhando diretamente com os próprios parâmetros;
- Ao invés de utilizar um único ponto, AGs utilizam uma população de pontos;
- AGs utilizam informações de custo/recompensa ao invés de derivadas da função objetivo;

- AGs utilizam regras de transição baseadas em aspectos probabilísticos e não determinísticos.

De forma geral, as seguintes operações são necessárias para se utilizar AGs na resolução de problemas de otimização:

1. Codificação das variáveis do problema;
2. Geração da população inicial;
3. Avaliação da população;
4. Seleção dos indivíduos para o cruzamento;
5. Cruzamento dos indivíduos selecionados gerando uma população de filhos;
6. Mutação nestes indivíduos;
7. Seleção dos sobreviventes para a próxima geração.

O AG proposto por Holland[40] ficou conhecido como AG Simples (AGS). Este algoritmo possui as características que estão sintetizadas na Tabela 3.2. A seguir, detalha-se tanto cada um dos itens apresentados na tabela para o AGS, quanto formas alternativas importantes. Um comentário relevante é que o fato de AGs utilizarem versões codificadas das variáveis dos problemas e não as próprias variáveis do problema é visto por alguns pesquisadores da área como um purismo histórico, dado que alguns deles não necessariamente aderem a tal regra. Nesse sentido, em muitos trabalhos recentes, utiliza-se codificação real em algoritmos que são classificados como genéticos. Neste trabalho, opta-se por usar a definição que tem raiz histórica, associando os AGs aos cromossomos dados em binário.

Tabela 3.2: Características do AGS

Forma de Representação	Codificação Binária
Operador de Recombinação	*Crossover* de 1-ponto
Operador de Mutação	*Bit-Flip* com baixa probabilidade
Seleção dos Pais	Proporcional à *Fitness* (Roleta)
Seleção dos Sobreviventes	Geracional

3.4.1 Codificação das Variáveis do Problema

Em um algoritmo genético, cada indivíduo é também chamado de *cromossomo* possuindo uma codificação binária. Outra forma de codificação é conhecida como codificação real e foi apresentada na Seção 3.3. O cromossomo é constituído por um conjunto de genes, sendo uma forma codificada de uma solução (ponto) do problema no espaço de busca. A codificação é binária porque, em cada gene, os alelos podem apresentar somente os valores 1 ou 0. Como mencionado na Seção 3.3, para um problema descrito por $m \in \mathbb{N}$ variáveis, um cromossomo é dado por $\vec{x}_{k,n} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_m)$, em que $\bar{x}_j$ é um gene que codifica a j-ésima variável do problema, x_j. Também é importante mencionar que $\vec{x}_{k,n}$ refere-se ao n-ésimo cromossomo da população, presente na k-ésima geração. O gene $\bar{x}_j$ é composto por $v_j \in \mathbb{N}$ alelos (bits), em que, nesta codificação, cada alelo pode apresentar apenas valores binários, $a_l \in \{0,1\}$, $l \in \{0,1,\ldots,v_j\}$, assim $\bar{x}_j$ é um número binário

$$\bar{x}_j = (a_{v_{j-1}} a_{v_{j-2}} \ldots a_2 a_1 a_0)_2, \tag{3.7}$$

que pode ser representado em decimal a partir da conversão a seguir, com a qual tem-se a equivalência relativa à j-ésima variável de decisão

$$\begin{aligned} x_j &= a_{v_{j-1}} \cdot 2^{v_{j-1}} + a_{v_{j-2}} \cdot 2^{v_{j-2}} + \cdots + a_1 \cdot 2^1 + a_0 \\ &= \sum_{l=0}^{v_j - 1} a_l \cdot 2^l. \end{aligned} \tag{3.8}$$

Para realizar o mapeamento do número binário $\bar{x}_j$ (3.7), codificado com v_j bits, para a variável real $L_j \leq x_j \leq U_j$, procedemos como se segue:

- Calculamos a distância entre dois valores adjacentes (ou precisão): $\Delta x_j = \dfrac{U_j - L_j}{2^{v_j} - 1}$;
- O valor x_j que corresponde ao binário $\bar{x}_j$ é dado por:

$$x_j = L_j + \Delta x_j \sum_{l=0}^{v_j - 1} a_l \cdot 2^l. \tag{3.9}$$

Para exemplificar, considere o seguinte caso de 3 variáveis limitadas dentro do conjunto dos reais: $0 \leq x_1, x_2, x_3 \leq 2$ e usaremos $v_1 = v_2 = v_3 = 5$ bits para os três genes. Assim, temos que:

- $(00000)_2$ mapeia 0;
- $(11111)_2$ mapeia 2;
- Tem-se que a precisão é: $\Delta x_1 = \Delta x_2 = \Delta x_3 = \dfrac{2-0}{2^5-1} = \dfrac{2}{31} \approx 0{,}0645$;
- Suponha que o primeiro indivíduo da geração inicial (geração 0) tenha sido gerado de forma aleatória e seja o cromossomo $\bar{x}_{0,1} = (10011{,}10000{,}00111)_2$, em que seus genes são mapeados como se segue e a vírgula foi utilizada para separar os genes:
  $\bar{x}_1 = (10011)_2$ mapeia $x_1 = 0 + \dfrac{2}{31}(1 \cdot 2^4 + 1 \cdot 2^1 + 1 \cdot 2^0) = \dfrac{2}{31} \cdot 19 = \dfrac{38}{31} \approx 1{,}226$,
  $\bar{x}_2 = (10000)_2$ mapeia $x_2 = 0 + \dfrac{2}{31}(1 \cdot 2^4) = \dfrac{2}{31} \cdot 16 = \dfrac{32}{31} \approx 1{,}032$,
  $\bar{x}_3 = (00111)_2$ mapeia $x_3 = 0 + \dfrac{2}{31}(1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = \dfrac{2}{31} \cdot 7 = \dfrac{14}{31} \approx 0{,}452$.
  Assim, concluímos que o cromossomo $\vec{x}_{0,1}$ corresponde ao ponto $\dfrac{1}{31}(38; 32; 14)$ no espaço de busca.

Tendo sido apresentada a codificação binária, é importante, neste ponto do texto, mencionar alguns problemas relacionados à esta forma de representação. Uma limitação bem evidente ocorre quando os limites das variáveis de decisão são desconhecidos *a priori*, assim, escolhas arbitrárias sobre tais limites deverão ser realizadas. Outra limitação está relacionada à precisão: obviamente quanto maior o número de bits, melhor a acurácia da representação o que, entretanto, irá ocasionar um aumento do custo computacional. Além disso, a fixação de uma determinada precisão tende a ser uma escolha arbitrária e que pode gerar impactos nos resultados obtidos pelo algoritmo. Por fim, um outro problema é que uma pequena diferença entre valores no espaço fenotípico pode significar uma grande diferença entre esses valores no espaço genotípico. Para exemplificar esse fato, considere os valores 6, 7 e 8 no espaço fenotípico. A distância entre os valores extremos e o 7 são iguais: $|8-7| = |6-7| = 1$. Contudo, as representações binárias desses valores quando codificados são: $6 = (0110)_2$, $7 = (0111)_2$ e $8 = (1000)_2$. A medida de distância tipicamente utilizada para números no contexto binário em que ocorrem operações de trocas de bits é conhecida como *Distância de Hamming* (dH), definida como a quantidade de bits diferentes entre duas *strings* ou palavras binárias. Assim, note que $\text{dH}[(0110)_2,(0111)_2] = 1$ e $\text{dH}[(1000)_2,(0111)_2] = 4$. Portanto, embora a distância entre 6 e 7 seja igual a distância entre 7 e 8 no espaço de fenótipos, o mesmo não ocorre no espaço

genotípico. Nesse sentido, a probabilidade de um operador de mutação que selecione aleatoriamente alguns bits e os troque não é a mesma de, ao se partir de $7[(0111)_2]$ chegar em $6[(0110)_2]$, quando se compara a troca de bits partindo de $7[(0111)_2]$ para se chegar em $8[(1000)_2]$. Uma alternativa capaz de aliviar bastante este ponto fraco é o código de Gray. Para mais detalhes, o leitor interessado é convidado a consultar Mathias e Whitley[55].

3.4.2 Seleção da População Inicial

Definida a codificação, é necessário selecionar os indivíduos que irão compor a população inicial. De forma genérica, a população que dá início a k-ésima geração é representada como $\mathrm{X}_k = \{\vec{x}_{k,i}\}_{i=1}^{N_{\mathrm{pop}}(k)}$, onde $0 \leq k < N_{\mathrm{ger}}$. Tipicamente, a população inicial X_0 é constituída de $N_{\mathrm{pop}}(0)$ vetores escolhidos aleatoriamente sobre o espaço de busca, amostrando-se uma distribuição de probabilidade uniforme. Este tipo de seleção é conhecido como *seleção aleatória*. Em se tratando de AGS, tem-se que a quantidade de cromossomos da população não varia ao longo das gerações, de modo que a quantidade inicial é preservada: $N_{\mathrm{pop}}(k) = N_{\mathrm{pop}}(0), \forall k$. Portanto, definir a quantidade $N_{\mathrm{pop}}(0)$ é importante, visto que quando esta quantidade é pequena, existe uma tendência de não se explorar de forma apropriada a região de busca, podendo o algoritmo ficar preso em algum mínimo local. Entretanto, grandes tamanhos de população podem tornar o algoritmo excessivamente lento devido ao aumento do custo computacional [56]. Por simplicidade, no restante do capítulo, para os casos em que $N_{\mathrm{pop}}(k)$ for constante ao longo de todas as gerações, trataremos esse parâmetro simplesmente como N_{pop}. Tipicamente, este tamanho é definido de forma empírica a partir de pré-simulações do algoritmo.

3.4.3 Seleção dos Pais

Nesse ponto, o objetivo é selecionar aqueles pais que irão participar do cruzamento. A partir da população definida, é importante avaliar os indivíduos de modo a compará-los segundo algum critério de desempenho. Como em um problema de otimização mono-objetivo temos a fitness $\bar{f}$, podemos ordenar os indivíduos hierarquicamente medindo a adaptação deles à esta função de custo. Assim, dados dois indivíduos $\vec{x}_{k,i}$ e $\vec{x}_{k,j}$, cujos cromossomos são $\bar{x}_{k,i}$ e $\bar{x}_{k,j}$, podemos calcular $\bar{f}(\bar{x}_{k,i})$ e $\bar{f}(\bar{x}_{k,j})$. Independentemente se o problema de otimização consiste de minimização ou maximização, devemos lembrar que a fitness é máxima para o indivíduo mais bem adaptado, refletindo, portanto, um problema de maximização. Assim, se $\bar{f}(\bar{x}_{k,i}) \geq \bar{f}(\bar{x}_{k,j})$, classificamos $\vec{x}_{k,i}$ como mais ou igualmente

adaptado em comparação a $\vec{x}_{k,j}$. Caso contrário, dizemos que $\vec{x}_{k,j}$ está mais adaptado. Nesse sentido, é importante que AGs repliquem o fato presente na evolução biológica de que indivíduos mais adaptados, apresentam maior probabilidade de reproduzirem. Lembrando que esta maior probabilidade, não exclui os indivíduos menos adaptados deste processo, o que garante a presença do aspecto de diversidade no algoritmo. Na sequência, são apresentadas algumas técnicas utilizadas nesses procedimentos.

Seleção Proporcional a Fitness

O AGS utiliza o operador conhecido como *Seleção Proporcional a Fitness* (SPF). Dado o conjunto de pais candidatos $\mathrm{X}_k = \{\vec{x}_{k,i}\}_{i=1}^{N_{\mathrm{pop}}(k)}$, a probabilidade de seleção do j-ésimo indivíduo para o cruzamento, $P_S(\bar{x}_{k,j})$, é dada por:

$$P_S(\bar{x}_{k,j}) = \frac{\bar{f}(\bar{x}_{k,j})}{\sum_{\ell=1}^{N_{\mathrm{pop}}(k)} \bar{f}(\bar{x}_{k,\ell})}, \tag{3.10}$$

em que é importante revisitar o fato que a fitness $\bar{f}$ (3.1) é definida como não negativa e o processo de minimizar a função objetivo f equivale a maximização de $\bar{f}$. Um problema que pode surgir relacionado a este operador é a perda de pressão seletiva em um contexto de indivíduos que apresentam valores de fitness muito próximos. Neste cenário, existe a tendência da seleção tornar-se meramente aleatória, a partir de uma distribuição uniforme. Outro problema é a variação brusca de valores de probabilidade para versões deslocadas da função de adaptação, obtidas ao se somar uma constante. A formulação dada em (3.1) visa resolver os dois problemas, na medida em que as probabilidades foram calculadas a partir de uma versão deslocada da função objetivo, por meio das subtrações dos termos $\beta(k)$ e $\bar{\beta}(k)$ (consultar Seção 3.3 para mais detalhes).

Por outro lado, um problema mais difícil de lidar envolvendo este operador é a convergência prematura. Isso pode ocorrer devido ao fato de indivíduos com altos valores de fitness tenderem a se reproduzir com frequência bastante elevada, dominando rapidamente toda a população. A técnica a seguir conhecida como Seleção por Roleta utiliza a ideia de SPF. Por outro lado, após ser apresentada tal técnica, mostramos a seleção conhecida como *Torneio*. Esta última técnica visa evitar o problema de convergência prematura que pode ocorrer com a SPF.

Seleção por Roleta

Uma forma de se utilizar a SPF ocorre no mecanismo conhecido como *Seleção por Roleta*. Nesse operador, indivíduos são mapeados para segmentos de reta contíguos no intervalo $[0, 1]$. O tamanho de um segmento associado ao indíviduo $\vec{x}_{k,j}$ será proporcional a probabilidade de seleção $P_S(\bar{x}_{k,j})$ (3.10). Assim, obtém-se os limites dos segmentos contíguos:

$$\begin{aligned} &a_0 = 0; \\ &a_i = P_S(\bar{x}_{k,i}) + \sum_{\ell=1}^{i-1} P_S(\bar{x}_{k,\ell}), \end{aligned} \tag{3.11}$$

de modo que para $j = 1, \ldots, N_{\text{pop}}(k) - 1$, o indivíduo $\vec{x}_{k,j}$ está associado ao segmento $[a_{j-1}, a_j[$ e o último indivíduo da população $\vec{x}_{k,N_{\text{pop}}(k)}$, está associado ao segmento $[a_{N_{\text{pop}}(k)-1}, a_{N_{\text{pop}}(k)}]$. Assim, sorteando-se um valor $r \in [0, 1]$, o indivíduo cujo segmento contém r será selecionado para ser um pai. Esse processo é análogo a girar uma roleta de um cassino com um único ponto de seleção. Roda-se a roleta, então, uma quantidade de vezes igual a quantidade de pais que deverão ser selecionados.

Seleção por Torneio

Esta estratégia tenta replicar o que ocorre no contexto de algumas espécies biológicas, de modo que indivíduos estabelecem uma disputa em que apenas o vencedor irá reproduzir. Assim, um subconjunto de $p \in \mathbb{N}$ indivíduos, com $p \geq 2$, é selecionado aleatoriamente do conjunto de candidatos X_k, sendo que apenas um dos indivíduos será escolhido para o cruzamento. Nesse sentido, o parâmetro p determina o tamanho do torneio e quanto maior o valor de p, maior se torna a pressão seletiva, dado que isso aumenta a probabilidade de indivíduos mais adaptados serem sorteados. Basicamente, existem dois tipos de torneios: determinístico e probabilístico. No torneio determinístico, sempre o indivíduo mais adaptado é selecionado como vencedor dentre os p indivíduos que disputaram o torneio. Por outro lado, no torneio probabilístico, pode-se sortear um número t, a partir de uma distribuição de probabilidade, sendo $t \in [0, 1]$. A ideia é permitir que nem sempre o mais adaptado vença o torneio, dependendo do valor de t quando comparado a uma probabilidade de seleção do mais adaptado $P_s(k) \in [0, 1[$, sendo esta última definida pelo projetista. Nesse contexto, se $t < P_s(k)$, então se seleciona o pai mais adaptado. Caso

contrário, será escolhido aleatoriamente um outro indivíduo dentre os $p-1$ restantes para o cruzamento.

Essa estratégia apresenta a vantagem de ser computacionalmente eficiente, na medida que não exige nem a ordenação da população de forma hierárquica, nem o cálculo de medidas adicionais *a priori*, apenas o valor da fitness. Além disso, a seleção por torneio ajuda a manter a diversidade na população, pois indivíduos menos aptos também apresentam probabilidade de serem selecionados como pais, mesmo no caso determinístico.

3.4.4 Cruzamento

O processo de cruzamento ou *crossover* consiste em combinar os genes de indivíduos da população previamente selecionados, os pais, para gerar novos indivíduos, os filhos. Um meta-parâmetro importante nesse processo é a probabilidade de cruzamento $P_c(k) \in [0,1]$. Essa medida determina quão provável será para que um par de pais selecionados seja submetido à recombinação genética.

Basicamente, existem duas maneiras de utilizar a informação da probabilidade de cruzamento $P_c(k) \in [0,1]$ para geração dos filhos. Uma maneira consiste em determinar o número de cruzamentos $N_c(k) \in \mathbb{N}$, por meio desta probabilidade $N_c(k) = P_c N_{\text{pop}}(k)$. Assim, a partir deste número são selecionados $2N_c(k)$ pais para cruzamento segundo alguma técnica conforme as apresentadas na Seção (3.4.3). A segunda maneira consiste em se sortear um número t aleatoriamente no intervalo [0,1[. Se $t < P_c(k)$, então 2 filhos serão gerados via *crossover* dos pais. Caso contrário, os filhos serão simplesmente cópias dos pais.

Tipicamente, AGs utilizam valores altos para a probabilidade de cruzamentos, $P_c(k) \in [0{,}6; 0{,}9]$. Cada uma das possíveis formas de realizar a combinação dos genes caracteriza um tipo de cruzamento, sendo os principais apresentados a seguir.

Crossover de 1 ponto de corte

Este tipo de cruzamento é realizado pelo AGS, sendo portanto aplicado para a codificação binária. Na k-ésima iteração, são selecionados $2N_c(k)$ pais, dado que o cruzamento de 2 pais gera 2 filhos. Consideremos o *crossover* de 1 ponto de corte para os pais previamente selecionados, $\vec{x}_{k,r}$ e $\vec{x}_{k,s}$. Cada indivíduo está associado a um cromossomo da forma $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_m)$, sendo que para para cada gene $\bar{x}_j$ composto por v_j bits, considere

os seguintes genes dos pais $\vec{x}_{k,r}$ e $\vec{x}_{k,s}$, respectivamente

$$\begin{aligned}\bar{x}_{j,r} &= (a_{j,v_{j-1}}a_{j,v_{j-2}}\ldots a_{j,2}a_{j,1}a_{j,0})_2,\\ \bar{x}_{j,s} &= (b_{j,v_{j-1}}b_{j,v_{j-2}}\ldots b_{j,2}b_{j,1}b_{j,0})_2,\end{aligned} \tag{3.12}$$

em que $a_{j,\ell}$,$b_{j,\ell} \in \{0,1\}$, $\ell \in \{0,1,\ldots,v_j\}$. Este método consiste em sortear aleatoriamente um número natural q no intervalo $[1, v_1 + v_2 + \cdots + v_m - 1]$ que será um ponto de corte nos cromossomos dos pais. Sendo o número q a posição no cromossomo, é possível mapeá-la para uma posição p no gene x_j. Assim, os cromossomos dos filhos serão gerados por meio de trocas das caudas entre os cromossomos pais considerando aquele ponto sorteado, de modo que os genes dos cromossomos dos filhos $\vec{y}_{k,r}$ e $\vec{y}_{k,s}$ serão determinados, respectivamente, como se segue

$$\begin{aligned}&(\bar{x}_{1,r}, \bar{x}_{2,r}, \ldots, \bar{x}_{j-1,r}, \bar{x}_{j,r'}, \bar{x}_{j+1,s}, \ldots, \bar{x}_{m,s})\\ &(\bar{x}_{1,s}, \bar{x}_{2,s}, \ldots, \bar{x}_{j-1,s}, \bar{x}_{j,s'}, \bar{x}_{j+1,r}, \ldots, \bar{x}_{m,r})\end{aligned}$$

de modo que os genes $\bar{x}_{j,r'}$ e $\bar{x}_{j,s'}$ são produzidos a partir das misturas dos códigos genéticos de ambos os indivíduos

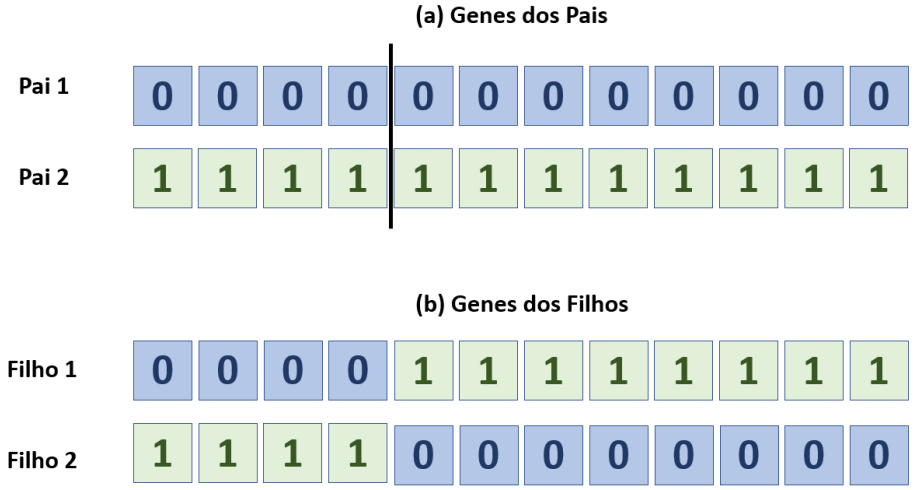
$$\begin{aligned}\bar{x}_{j,r'} &= (a_{v_{j-1}}a_{v_{j-2}}\ldots a_{p+1}a_p b_{p-1}\ldots b_2 b_1 b_0)_2,\\ \bar{x}_{j,s'} &= (b_{v_{j-1}}b_{v_{j-2}}\ldots b_{p+1}b_p a_{p-1}\ldots a_2 a_1 a_0)_2.\end{aligned} \tag{3.13}$$

Para tornar mais claro como ocorre este cruzamento, um exemplo é apresentado na Figura 3.9. Note que os genes de cada um dos filhos possuem partes do código genético de ambos os pais, sendo que o ponto de corte determina as partes que virão de um dos pais e aquelas que virão do outro.

Crossover de n pontos de corte

Este tipo de cruzamento é uma forma generalizada do cruzamento anterior. O primeiro passo é definir o número $n \in \mathbb{N}$ de pontos de corte. A seguir, são selecionados uma quantidade de n naturais no intervalo $[1,\ v_j - 1]$, sendo eles: p_1, p_2, $\ldots$, p_{n-1}, p_n. Assim, de forma similar ao que ocorre para 1 ponto de corte, são estabelecidos cortes nos genes dos pais sorteados separando-os em partes, de modo que os filhos são gerados a partir de trocas alternadas entre as partes.

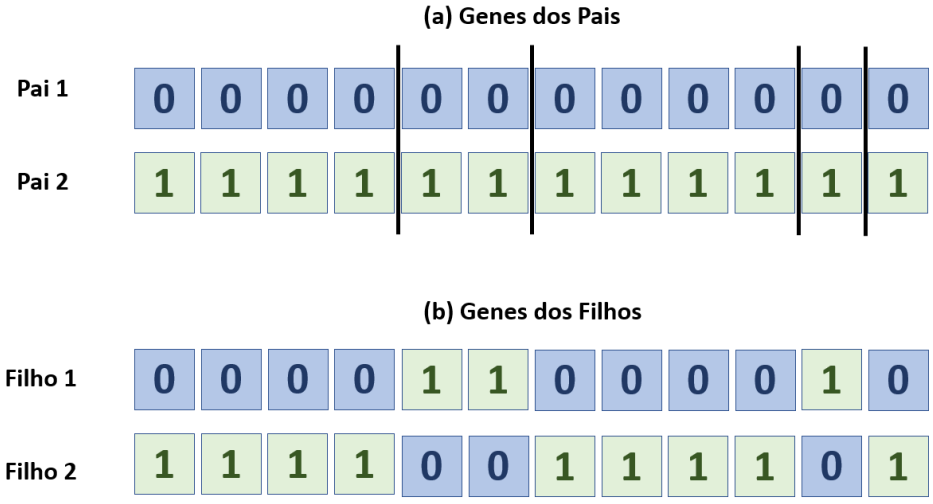
Figura 3.9: Exemplo de Crossover com 1 ponto de corte.



Fonte: Elaborada pelo Autor.

A Figura 3.10 apresenta um exemplo para $n = 4$ pontos de corte. Note que os pontos de corte determinam as alternâncias das partes dos códigos de cada pai que serão recebidos pelos filhos.

Figura 3.10: Exemplo de Crossover com $n = 4$ pontos de corte.



Fonte: Elaborada pelo Autor.

Crossover uniforme

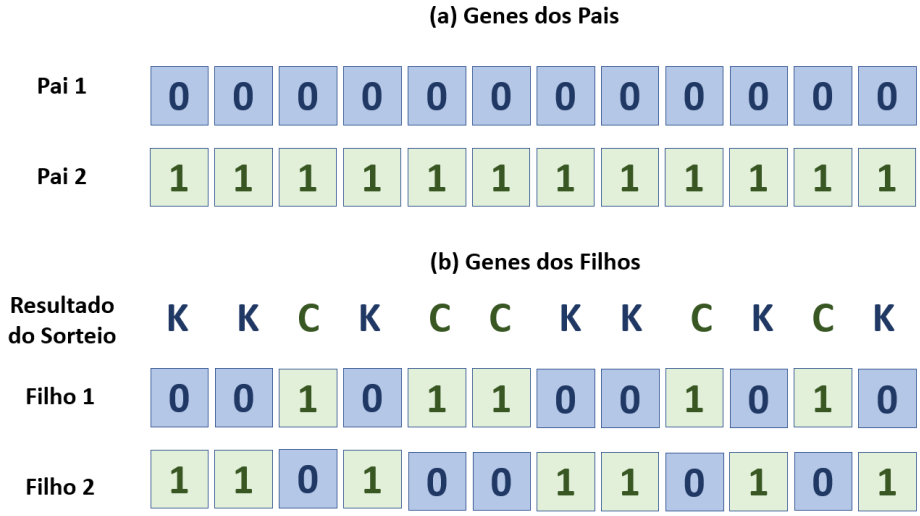
Este tipo de cruzamento funciona como se lançássemos uma moeda para cada alelo do genótipo e, caso o resultado seja "cara" (K) será preservado o alelo do primeiro pai para o genótipo do primeiro filho. Por outro lado, caso o resultado seja "coroa" (C) será preservado o alelo do segundo pai para composição do genótipo do primeiro filho. Por fim,

o segundo filho é obtido como uma cópia inversa dos alelos do primeiro filho. Matematicamente, sejam os pais $\vec{x}_{k,r}$ e $\vec{x}_{k,s}$ cujos cromossomos são dados por $(\bar{x}_{r,1}, \bar{x}_{r,2}, \ldots, \bar{x}_{r,m})$ e $(\bar{x}_{s,1}, \bar{x}_{s,2}, \ldots, \bar{x}_{s,m})$, em que r e s são valores distintos, $r, s \in \{1, \ldots, N_{\text{pop}}(k)\}$, e irão gerar o t-ésimo descendente $\vec{y}_{k,t}$ cujo cromossomo é $(\bar{y}_{t,1}, \bar{y}_{t,2}, \ldots, \bar{y}_{t,m})$, $t \in \{1, \ldots, N_c(k)\}$, de modo que o i-ésimo alelo é obtido como se segue

$$\bar{y}_{t,i} = \begin{cases} \bar{x}_{s,i} & \text{se } r_i \leq 0{,}5 \\ \bar{x}_{r,i} & \text{se } r_i > 0{,}5 \end{cases}, \tag{3.14}$$

sendo que $r_i \in [0,1]$ é sorteado a partir de uma distribuição uniforme. A Figura 3.11 apresenta um exemplo deste tipo de procedimento.

Figura 3.11: Exemplo de Crossover Uniforme.



Fonte: Elaborada pelo Autor.

Crossover por variável

No crossover por variável, aplica-se o cruzamento de 1 ponto de corte limitado a cada gene em específico que mapeia cada uma das variáveis, ao invés de se realizar o procedimento considerando o genótipo por completo. Sendo assim, atribui-se um ponto de corte para cada variável de decisão. Consideremos os pais previamente selecionados, $\vec{x}_{k,r}$ e $\vec{x}_{k,s}$. Para cada gene $\bar{x}_j$ composto por v_j bits, considere os seguintes genes dos pais $\vec{x}_{k,r}$ e

$\vec{x}_{k,s}$, respectivamente

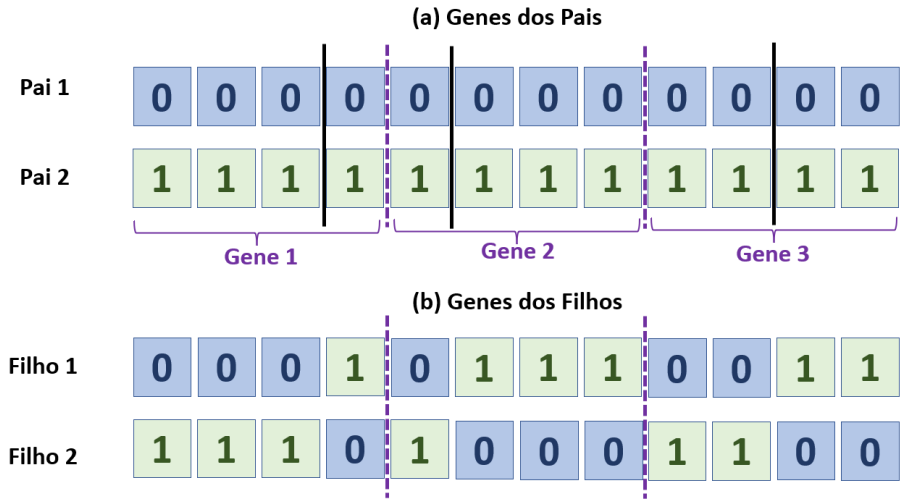
$$\begin{aligned} \bar{x}_{j,r} &= (a_{v_{j-1}} a_{v_{j-2}} \ldots a_2 a_1 a_0)_2, \\ \bar{x}_{j,s} &= (b_{v_{j-1}} b_{v_{j-2}} \ldots b_2 b_1 b_0)_2, \end{aligned} \tag{3.15}$$

em que a_ℓ,$b_\ell \in \{0,1\}$, $\ell \in \{0,1,\ldots,v_j\}$. Este método consiste em sortear aleatoriamente um número natural p no intervalo $[1, v_j - 1]$ que será um ponto de corte nos cromossomos dos pais. Os genes dos filhos serão gerados por meio de trocas das caudas entre os dois genes pais considerando aquele ponto sorteado, de modo que os genes dos filhos $\vec{y}_{k,r}$ e $\vec{y}_{k,s}$ serão determinados como se segue

$$\begin{aligned} \bar{y}_{j,r} &= (a_{v_{j-1}} a_{v_{j-2}} \ldots a_{p+1} a_p b_{p-1} \ldots b_2 b_1 b_0)_2, \\ \bar{y}_{j,s} &= (b_{v_{j-1}} b_{v_{j-2}} \ldots b_{p+1} b_p a_{p-1} \ldots a_2 a_1 a_0)_2. \end{aligned} \tag{3.16}$$

Para tornar mais claro como ocorre este cruzamento, apresenta-se um exemplo na Figura 3.12. Note que os genes de cada um dos filhos possuem partes do código genético de ambos os pais, sendo que o ponto de corte determina as partes que virão de um dos pais e aquelas que virão do outro.

Figura 3.12: Exemplo de Crossover por Variável considerando um indivíduo com 3 genes de 4 alelos. Considere que os pontos de corte sorteados sejam 3, 1 e 2, respectivavemnte para os genes 1, 2 e 3.



Fonte: Elaborada pelo Autor.

3.4.5 Mutação

O processo de mutação consiste em inserir novas informações de forma aleatória no código genético de indivíduos da população previamente selecionados, para gerar novos indivíduos, os filhos mutantes. Um meta-parâmetro importante nesse processo é a probabilidade de mutação $P_m(k) \in [0,1]$. Essa medida determina quão provável será para que um indivíduo selecionado seja submetido à mutação genética. Tipicamente,

$$\frac{1}{N_{\text{pop}}(k)} \leq P_m(k) \leq \frac{1}{\ell}$$

em que $\ell = v_1 + v_2 + \cdots + v_m$ é o tamanho do cromossomo de m genes, dado que v_j é o tamanho do j-ésimo gene.

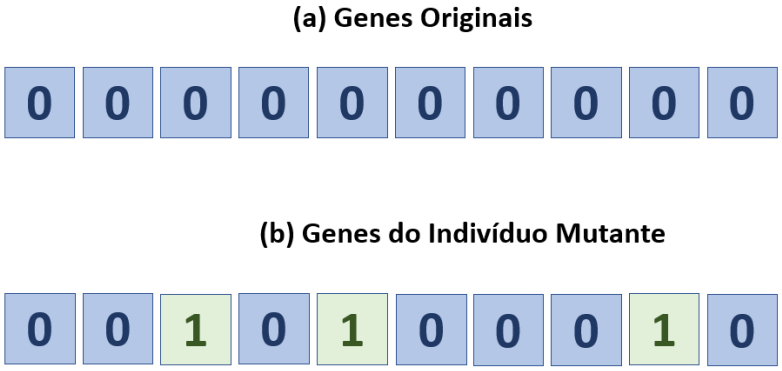
Basicamente, existem três maneiras de utilizar a informação da probabilidade de mutação $P_m(k) \in [0,1]$ para geração dos filhos mutantes. Uma maneira consiste em determinar o número de mutações $N_m(k) \in \mathbb{N}$, por meio desta probabilidade: $N_m(k) = P_m N_{\text{pop}}(k)$. Assim, a partir deste número são selecionados aleatoriamente segundo alguma distribuição de probabilidade $N_m(k)$ pais para mutação. Em seguida, para cada pai, sorteia-se o bit a ser alterado. A segunda maneira consiste em se sortear um número t aleatoriamente no intervalo [0,1[para cada indivíduo da população. Se $t < P_c(k)$, então o indivíduo sofrerá mutação sendo um de seus bits escolhidos aleatoriamente; em caso contrário, o indivíduo não sofrerá mutação. A outra forma consiste em se determinar o número médio de mutações para cada indivíduo como ℓP_m e assim selecionar este número de bits que serão alterados.

A forma de mutação típica da codificação binária é conhecida como *Bit-Flip*, em que se troca cada alelo segundo a probabilidade $P_m(k)$ considerando alguma das formas de seleção apresentadas no parágrafo anterior. A alteração do alelo é simplesmente uma inversão de valor binário: se o alelo vale 0, então ele é trocado para 1; e caso o valor seja 1, este será trocado para 0. A Figura 3.13 apresenta um exemplo de mutação considerando um indivíduo de 10 bits e $P_m = 30\%$.

3.4.6 Seleção dos Sobreviventes

Ao final de uma geração, é importante selecionar os indivíduos que irão sobreviver para a geração seguinte. Este processo é conhecido como seleção dos sobreviventes ou substituição. Sendo μ, o tamanho da população e λ, o número de descendentes, tem-se

Figura 3.13: Exemplo de Mutação *Bit-Flip* considerando um indivíduo com um total de 10 bits. Considerando $P_m = 30\%$, em média 3 de cada 10 bits serão selecionados para mutação. Neste exemplo, após um sorteio aleatório, os alelos das posições 3, 5 e 9 sofreram mutação.



Fonte: Elaborada pelo Autor.

como modelos de população:

Modelo Geracional ($\mu = \lambda$): cada indivíduo vive por exatamente uma geração, sendo todos os pais substituídos pelos descendentes.

Modelo de Estado Estacionário (*Steady-State*) ($\mu > \lambda$): apenas uma parte dos pais é substituída pelos λ descendentes, seguindo algum critério baseada em idade ou adaptação.

Assim, as duas principais abordagens de substuição são:

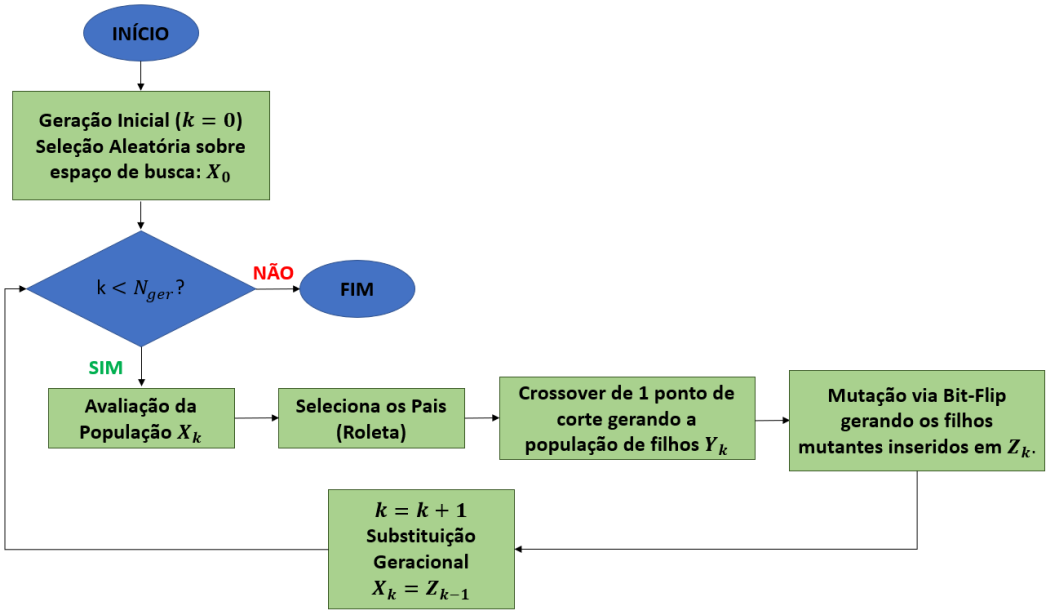
Substituição Baseada em Idade: os indivíduos são substituídos conforme a sua idade. No modelo geracional, todos os pais são substituídos pelos descendentes e no modelo de Steady-State a parte dos pais que será substituída pode ser escolhida de forma aleatória ou o processo pode envolver uma fila do tipo *First-in-First-Out* (Primeiro a entrar, primeiro a sair).

Substituição Baseada em Fitness: todos os λ descendentes são incluídos, de modo que a medida de adaptação é utilizada para decidir quais dos μ pais serão substituídos. A decisão pode ser baseada em métodos de SPF, Torneio ou mantendo μ melhores indivíduos considerando toda a população de pais e descendentes.

3.4.7 O Algoritmo Genético Simples

Considerando os principais aspectos mencionados de forma geral para AGs, apresenta-se a estrutura do Algoritmo Genético Simples na Figura 3.14.

Figura 3.14: Fluxograma do AGS.



Fonte: Elaborada pelo Autor.

A população inicial da geração $k = 0$, X_0, é gerada de forma aleatória sobre o espaço de busca (Subseção 3.4.2). Em seguida, para cada geração k, a população X_k é avaliada por meio da fitness, de modo que um subconjunto de pais é selecionado via roleta considerando a informação dada pela probabilidade de cruzamento P_c (Subseção 3.4.4). Sobre os pais selecionados, aplica-se o cruzamento de 1 ponto de corte (Subseção 3.4.4), de modo que a população de filhos Y_k é produzida. Em seguida, aplica-se a mutação sobre o conjunto Y_k, considerando o processo de *Bit-Flip* (Subseção 3.4.5), produzindo o conjunto de filhos que incluem indivíduos mutantes e não mutantes, Z_k. Por fim, avança-se a geração, fazendo $k = k + 1$, e substituímos os pais pelos descendentes, $X_k = Z_{k-1}$ (Subseção 3.4.6).

3.5 Evolução Diferencial

Algoritmos de Evolução Diferencial (ED) são AEs que, ao invés de replicar o processo da evolução biológica em todos os passos, inserem formas alternativas de cruzamento e mutação. A ideia fundamental presente neste algoritmo é gerar cada novo indivíduo selecionando-se um indivíduo da população ao qual será adicionado a diferença entre outros

indivíduos da população ponderada por um fator positivo [6].

Visando tornar a notação aqui apresentada o mais próxima possível da notação previamente utilizada neste trabalho, optou-se por usar uma notação própria neste ponto do texto que reaproveita o que já foi apresentado no restante do capítulo. A ideia é evitar a inserção de uma nova notação para o leitor. É importante mencionar que, como as diversas sub-áreas da Computação Evolucionária desenvolveram-se de forma independente, tipicamente as notações também se diferenciam bastante.

Diferentemente de AGs, eles utilizam a codificação real, de modo que cada indivíduo chamado de *vetor*, mapeia diretamente as variáveis do problema. Nesse sentido, consideremos no vetor $\vec{x}_{k,n} = (x_1, x_2, \ldots, x_m)$, tem-se que cada uma das variáveis de decisão x_j, $j \in \{1, \ldots, m\}$ estão mapeadas. O algoritmo é iniciado de modo que a população de N_{pop} indivíduos seja escolhida aleatoriamente sobre o espaço de busca conforme descrito na Subseção 3.4.2.

3.5.1 Mutação Diferencial

Na k-ésima iteração, $0 \leq k < N_{\text{ger}}$, serão produzidos N_{pop} indivíduos via mutação diferencial, um processo fundamental para geração de diversidade em algoritmos do tipo DE. Assim, para produzir o p-ésimo indivíduo, $p \in \{1, \ldots, N_{\text{pop}}\}$ via mutação, consideremos: o indivíduo mais bem adaptado da população $\vec{x}_{k,o}$ e sejam os indivíduos escolhidos aleatoriamente de modo a serem distintos entre si, $\vec{x}_{k,r}$, $\vec{x}_{k,s}$ e $\vec{x}_{k,t}$, sendo que os índices $r, s, t, o \in \{1, \ldots, N_{\text{pop}}\}$. Dois deles serão utilizados para se obter uma diferença $\vec{x}_{k,s} - \vec{x}_{k,t}$ que, multiplicada por um *fator de escala* $F > 0$, tem-se o termo denotado como *diferença ponderada*, $F(\vec{x}_{k,s} - \vec{x}_{k,t})$. Tal diferença ao ser adicionada ao terceiro vetor selecionado ou ao melhor vetor da população, produz um novo vetor conhecido como vetor *doador* ou *modificado* $\vec{z}_{k+1}$ que será utilizado na iteração $k + 1$ [6]:

$$\vec{z}_{k+1,p} = \vec{x}_{k,r} + F(\vec{x}_{k,s} - \vec{x}_{k,t})$$

ou

$$\vec{z}_{k+1,p} = \vec{x}_{k,o} + F(\vec{x}_{k,s} - \vec{x}_{k,t}). \tag{3.17}$$

Assim, é importante que $N_{\text{pop}} \geq 4$ e $F \in [0, 2]$, permitindo-se controlar a amplitude das perturbações que serão aplicadas. Uma abordagem alternativa é utilizar duas diferenças ponderadas, aumentando o aspecto de diversidade, desse modo, deverão ser selecionados

os indivíduos distintos entre si $\vec{x}_{k,r}$, $\vec{x}_{k,s}$, $\vec{x}_{k,t}$, $\vec{x}_{k,u}$ e $\vec{x}_{k,v}$:

$$\vec{z}_{k+1,p} = \vec{x}_{k,r} + F \cdot (\vec{x}_{k,s} - \vec{x}_{k,t} + \vec{x}_{k,u} - \vec{x}_{k,v})$$

ou

$$\vec{z}_{k+1,p} = \vec{x}_{k,o} + F \cdot (\vec{x}_{k,s} - \vec{x}_{k,t} + \vec{x}_{k,u} - \vec{x}_{k,v}), \tag{3.18}$$

em que $r, s, t, u, v, o \in \{1, \ldots, N_{\text{pop}}\}$, de modo que, exceto o que denota o índice do melhor indivíduo, os outros são escolhidos aleatoriamente, sendo $N_{\text{pop}} \geq 6$.

3.5.2 Cruzamento

No cruzamento, tem-se como objetivo aumentar a diversidade dos indivíduos que sofreram mutação. Utiliza-se neste processo um procedimento bastante similar ao Cruzamento Uniforme apresentado na Subseção 3.4.4. Para cada $p \in \{1, \ldots, N_{\text{pop}}\}$, o vetor doador $\vec{z}_{k+1,p} = (z_{p,1}, z_{p,2}, \ldots, z_{p,m})$ obtido da mutação diferencial é misturado com as componentes de outro vetor que é selecionado aleatoriamente e é chamado *vetor alvo* $\vec{x}_{k+1,d} = (x_{d,1}, x_{d,2}, \ldots, x_{d,m})$, $d \in \{1, \ldots, N_{\text{pop}}\}$, para gerar um novo vetor conhecido como *vetor tentativa* ou *vetor experimental* de acordo com a probabilidade de cruzamento P_c, $\vec{y}_{k+1,p} = (y_{p,1}, y_{p,2}, \ldots, y_{p,m})$ em que o i-ésimo componente do vetor é obtido como se segue

$$y_{p,i} = \begin{cases} z_{p,i} & \text{se } r_{p,i} \leq P_c \\ x_{d,i} & \text{se } r_{p,i} > P_c \end{cases}, \tag{3.19}$$

sendo que $r_{p,i} \in [0, 1]$ é obtido amostrando-se uma distribuição uniforme. Após o cruzamento, caso a i-ésima componente do vetor experimental esteja fora da região de busca, fazemos as correções reinserindo-as nos limites da região de busca

$$y_{p,i} = \begin{cases} U_i & \text{se } y_{p,i} > U_i \\ L_i & \text{se } y_{p,i} < L_i \end{cases}, \tag{3.20}$$

lembrando que cada variável x_j está limitada em um intervalo dado por $L_j \leq x_j \leq U_j$.

O cruzamento (3.19) é conhecido na literatura de ED como *cruzamento binomial*. Uma alternativa também utilizada em algoritmos ED é o que se entende por *cruzamento exponencial*. Nesse caso, as componentes do vetor experimental são dadas pelas componentes do vetor doador enquanto o número randômico for menor ou igual a P_c. Para o

número randômico de valor maior que P_c, as componentes seguintes virão do vetor alvo. Assim, suponha que para $1 \leq i < q$, em que $q \in \{1, \ldots, m\}$, obteve-se em q um número randômico de valor superior a probabilidade de cruzamento, $r_{q,i} > P_c$, e, assim, o vetor experimental deverá ser dado por

$$\vec{y}_{k+1,p} = (z_{p,1}, z_{p,2}, \ldots, z_{p,q-1}, z_{p,q}, x_{d,q+1}, x_{d,q+2}, \ldots, x_{d,m-1}, x_{d,m}). \tag{3.21}$$

3.5.3 Seleção

Neste procedimento, objetiva-se selecionar os sobreviventes para a próxima geração. Note que, neste ponto, partiu-se de uma população de N_{pop} pais dos quais foram produzidos N_{pop} filhos, após os operadores de mutação e cruzamento. Assim, para cada $p \in \{1, \ldots, N_{\text{pop}}\}$, tem-se a seleção conforme se segue, considerando o problema de minimização

$$\vec{x}_{k+1,p} = \begin{cases} \vec{y}_{k+1,p} & \text{se } f(\vec{y}_{k+1,p}) < f(\vec{x}_{k,p}) \\ \vec{x}_{k,p} & \text{se } f(\vec{y}_{k+1,p}) \geq f(\vec{x}_{k,p}) \end{cases}, \tag{3.22}$$

ou seja, caso caso o custo do vetor experimental gere algum benefício em relação ao vetor alvo, tem-se a substituição do vetor alvo pelo experimental na próxima geração. Caso contrário, é permitido ao vetor alvo avançar a próxima geração. O processo iterativo continua até que algum critério de parada seja alcançado. Tipicamente, um dos critérios de parada que é utilizado se refere ao número de gerações máximo ($0 \leq k < N_{\text{ger}}$) para execução do algoritmo. Outro critério também bastante utilizado relaciona-se à estagnação de um índice de melhoria dado por $\left| f_{\min}^{k+1} - f_{\min}^{k} \right| < \epsilon$. Neste índice, $f_{\min}^{k}$ indica o menor valor da função objetivo obtido na k-ésima geração e o parâmetro ϵ é utilizado para determinar o limite de estagnação que será utilizado como critério de parada.

3.5.4 Parâmetros em Evolução Diferencial

De forma geral, um algoritmo do tipo ED apresenta três parâmetros fundamentais: o fator de escala F, o tamanho da população N_{pop} e a probabilidade de cruzamento P_c. Orientações e recomendações de caráter empírico propostas para esses algoritmos incluem [6, 8]:

- A população inicial deve ser gerada de forma a se espalhar bem ao longo da região

de busca.

- A princípio usar $P_c < 0{,}5$ e caso existam dificuldades de convergência adotar $P_c \in [0{,}8; 1]$.
- De forma geral, tende-se a escolher $N_p = 10m$, em que m é o número de variáveis do problema.
- Normalmente, escolhe-se $F \in [0{,}5; 1]$.
- Quanto maior for o valor de N_{pop} menor deverá ser o valor F.

3.5.5 Estratégias em Evolução Diferencial

Como uma família de algoritmos, existem diversas variações que podem estar presentes em algoritmos caracterizados como ED. Basicamente, utiliza-se se a notação **ED/a/b/c** para descrever tais possíveis variações, de modo que:

- **a**: especifica qual vetor será perturbado, podendo ser *rand*, em que um vetor é escolhido de forma aleatória, ou *best*, escolhe-se o vetor mais adaptado da população. Existem variações como *current-to-best* e *current-to-pbest* que serão apresentadas adiante.
- **b**: especifica o número de diferenças ponderadas que não consideram o indivíduo atual e, portanto, são usadas para a perturbação – uma conforme (3.17) ou duas, (3.18).
- **c**: denota o tipo de cruzamento que pode ser binomial (*bin*) ou exponencial (*exp*).

Em geral, o tipo de cruzamento mais utilizado é o binomial, sendo que as principais estratégias, desconsiderando o cruzamento são apresentadas a seguir [57, 58, 59]. A notação e os termos utilizados foram detalhados na Subseção 3.5.1.

1. Estratégia **ED/rand/1**, em que os índices r, s, t denotam elementos aleatoriamente selecionados dentro da população e $F > 0$ é o fator de escala

$$\vec{z}_{k+1,p} = \vec{x}_{k,r} + F(\vec{x}_{k,s} - \vec{x}_{k,t}).$$

2. Estratégia **ED/rand/2**, em que ocorrem duas diferenças ponderadas, sendo que os índices r, s, t, u, v novamente denotam elementos aleatoriamente selecionados dentro da população

$$\vec{z}_{k+1,p} = \vec{x}_{k,r} + F(\vec{x}_{k,s} - \vec{x}_{k,t} + \vec{x}_{k,u} - \vec{x}_{k,v}).$$

3. Estratégia **ED/best/1** em que o índividuo ótimo de índice o é perturbado por apenas uma diferença

$$\vec{z}_{k+1,p} = \vec{x}_{k,o} + F(\vec{x}_{k,s} - \vec{x}_{k,t}).$$

4. Estratégia **ED/best/2** em que o índividuo ótimo de índice o é perturbado considerando duas diferenças ponderadas

$$\vec{z}_{k+1,p} = \vec{x}_{k,o} + F(\vec{x}_{k,s} - \vec{x}_{k,t} + \vec{x}_{k,u} - \vec{x}_{k,v}).$$

5. Estratégia **ED/current-to-best/1**, em que i denota o indivíduo atual da população, sendo a estratégia realizada para todos os indivíduos da população

$$\vec{z}_{k+1,p} = \vec{x}_{k,i} + F(\vec{x}_{k,o} - \vec{x}_{k,i} + \vec{x}_{k,s} - \vec{x}_{k,t}).$$

6. Estratégia **ED/current-to-best/2**: em que são utilizadas duas diferenças ponderadas, desconsiderando a diferença relativa ao indíviduo atual

$$\vec{z}_{k+1,p} = \vec{x}_{k,i} + F(\vec{x}_{k,o} - \vec{x}_{k,i} + \vec{x}_{k,s} - \vec{x}_{k,t} + \vec{x}_{k,u} - \vec{x}_{k,v}).$$

7. Estratégia **ED/current-to-pbest/1**, em que **pbest** denota um indivíduo aleatoriamente escolhido entre os P% mais adaptados da população, onde P é um parâmetro definido pelo usuário

$$\vec{z}_{k+1,p} = \vec{x}_{k,i} + F(\vec{x}_{k,\text{pbest}} - \vec{x}_{k,i} + \vec{x}_{k,s} - \vec{x}_{k,t}).$$

Portanto, este capítulo apresentou as bases sobre computação evolucionária e AEs,

de modo a proporcionar ao leitor uma visão geral e resumida sobre o assunto, devido ao fato de esse campo de estudo ser bastante extenso e apresentar uma quantidade considerável de variações em seus algoritmos. O leitor interessado em mais detalhes e profundidade é convidado a estudar as seguintes referências: [6, 8, 50, 56, 57, 58, 59, 60]. O capítulo seguinte situa o presente trabalho no contexto de trabalhos correlatos já desenvolvidos e aspectos educacionais inerentes ao produto educacional que será apresentado no Capítulo 5.

4 Trabalhos Correlatos e Aspectos Educacionais

No Capítulo 1, buscamos discutir a relevância de se fazer pesquisa considerando os eixos principais de intersecção deste trabalho: *Otimização, Algoritmos, AEs* e *gamificação*, para o ensino básico. O presente capítulo visa aprofundar essa discussão em sintonia com o arcabouço literário de pesquisas, sejam elas desenvolvidas no PROFMAT ou fora dele.

Na sequência do capítulo, tem-se a divisão em seções, de modo que cada seção se refere a um eixo principal em que esta pesquisa está inserida. Quanto à parte referente ao PROFMAT, as pesquisas foram realizadas considerando o Banco de Dissertações do programa[1], utilizando o título de cada seção como palavra de busca para a pesquisa ou algum termo similar, conforme será mencionado adiante.

4.1 Otimização

Como AEs são ferramentas dedicadas à resolução de problemas de otimização, a primeira parte dos trabalhos correlatos que serão apresentados neste capítulo referem-se a tal termo. As pesquisas foram feitas no dia 12/06/2022, considerando os trabalhos que possuem o termo *otimização* no título e estão presentes no Banco de Dissertações do PROFMAT. No momento mencionado, foram encontrados 64 registros. Nesse sentido, percebemos que, nos últimos anos, a otimização tem sido objeto recorrente de pesquisa no âmbito da educação básica dentro do programa, sendo que alguns dos trabalhos têm apresentado estratégias didáticas envolvendo o assunto. Devido a quantidade considerável de trabalhos, utilizamos como critério para selecionar os trabalhos a serem aqui tratados, os oito trabalhos mais recentes que deixassem claro o foco no ensino básico, dado que esse é um aspecto relevante da presente pesquisa. Além disso, buscamos também por trabalhos que realizassem abordagens que se distinguissem seja pelos métodos, pela formulação, pelo

[1] https://profmat-sbm.org.br/dissertacoes/

uso de ferramentas como calculadora, ou apenas papel e lápis na execução de algoritmos ou ainda softwares variados. O objetivo foi conseguir capturar trabalhos recentes com maior diversidade de abordagens, de modo a se perceber um contexto geral de pesquisa sobre o tema da otimização. Nesse contexto, percebemos o uso massivo do software GeoGebra, de modo que selecionamos apenas dois trabalhos que focaram no uso dessa ferramenta.

Na dissertação proposta por Castro[61], mostra-se que problemas de otimização são comuns no cotidiano, aparecendo, por exemplo, na minimização de custos. O trabalho teve como foco a elaboração de uma sequência didática para os alunos da terceira série do Ensino Médio. Embora a proposta não tenha sido aplicada, essa questão foi colocada como uma possibilidade futura de trabalho. Argumentou-se que a abordagem desenvolvida no corpo do trabalho das estratégias para a resolução de problemas são essenciais e devem ser ministradas aos alunos do Ensino Médio para que eles possam consolidar as habilidades específicas do ensino, conforme mencionado na BNCC. Além disso, também é mencionado que formular, resolver problemas e criar soluções com base nos conhecimentos das diferentes áreas é uma competência essencial para a formação dos alunos no intuito de despertar o interesse pela área de Matemática. Por fim, o autor se dedicou a fazer considerações sobre como o PROFMAT impactou positivamente sua atuação profissional. O produto educacional foi uma sequência didática composta de 4 atividades no contexto de otimização em que se utilizou apenas a função quadrática como base.

Fernandes[62] trabalha em sua dissertação a Resolução de Problemas, a partir da Modelagem Matemática, que é uma ferramenta eficaz na tomada de decisão. Em particular, o trabalho considera a abordagem de situações-problema com dados incertos, a partir de técnicas de Otimização sob Incerteza. O objetivo é ressaltar a relevância do ensino de estatística, probabilidades, álgebra e geometria na Educação Básica. Em tal trabalho, propõe-se duas partes: a abordagem teórica, sempre utilizando problemas como ponto de partida para a generalização, com foco no aprofundamento dos conteúdos necessários para a otimização com dados incertos e o desenvolvimento do estudo voltado aos alunos do ensino básico, revendo e ampliando conceitos abordados no Ensino Médio. No decorrer da dissertação houve uma preocupação de abordar problemáticas que discorressem sobre questões aplicadas, agregando competências que enfatizassem os conceitos probabilísticos os quais permeiam o cotidiano. Nesse sentido, visou-se distanciar dos jogos de azar, que podem dar uma interpretação enviesada dos conceitos de probabilidade e sua aplicabilidade.

Pretende-se como proposta futura preparar um material a partir das propostas apresentadas, com o objetivo de realizar atividades com os alunos. Sobre o produto educacional, tem-se que a própria dissertação apresenta diversas situações-problema de otimização envolvendo o contexto aleatório. É importante reiterar que não foi apresentada uma proposta didática.

O trabalho de Saldanha[63] tem como objetivo apresentar três estratégias para a resolução de problemas de otimização, envolvendo alguns conceitos geométricos como áreas de figuras planas e volume de sólidos geométricos. São abordadas duas estratégias de resolução, uma com o uso de valores de máximos e mínimos de uma função quadrática e, posteriormente, outra, através do uso da desigualdade das médias. A terceira estratégia, abordando problemas de caráter geométrico, utiliza os teoremas do Cálculo que consideram máximos e mínimos. Para o trabalho, utilizou-se como metodologia, a pesquisa bibliográfica. Por meio dessa pesquisa, concluiu-se que, apesar dos três artifícios apresentados, o cálculo toma grande importância para a resolução desse tipo de problema, já que existe uma grande quantidade de situações que não podem ser resolvidas através da desigualdade das médias, nem com o uso da função quadrática. Ademais, as estratégias apresentadas podem ser abordadas no Ensino Médio, visto que formular e resolver problemas e criar soluções com base nos conhecimentos das diferentes áreas é uma das competências gerais da educação básica de acordo com a BNCC. Nesse sentido, o estudo dos problemas de otimização trata de conteúdos importantes próprios da educação como básica, tais como o estudo de função quadrática, os conceitos de média aritmética, média geométrica e noções geométricas. Além disso, aborda-se também conceitos que vão além do ensino básico relacionados ao cálculo de limites e derivadas, que apesar de não serem incluídos por muitos professores, no Ensino Médio, poderiam ser apresentados de forma simples e modesta no contexto do estudo das funções. Conclui-se que, apesar dos três artifícios apresentados, o cálculo toma grande importância para a resolução desse tipo de problema, já que existem várias situações-problema que não podem ser resolvidas através da desigualdade das médias, nem com o uso dos valores de máximo e mínimo de uma função quadrática. O autor almeja que o trabalho possa servir de apoio a professores que desejem ampliar os seus conhecimentos pela área abordada ou utilizar em seu planejamento didático para as aulas de Matemática a fim de que possam desenvolver em seus alunos a capacidade de resolver problemas de otimização. Sobre o produto educacional tem-se a apresentação de diversas situações-problema de otimização, mesmo que não tenha sido elaborada uma proposta

didática propriamente dita.

A dissertação de Sena[64] apresenta uma proposta de resolução de problemas de otimização linear através do método geométrico com a utilização do GeoGebra. Inicialmente, realiza-se uma fundamentação teórica, apresentando tópicos de otimização linear e mostrando características da formulação matemática e teoremas que fundamentam o método geométrico. O desafio de reinventar e contribuir de forma significativa para o ensino da matemática formaram o ponto inicial do trabalho. Foram dados subsídios para professores com a resolução desses problemas através do método geométrico, com diversas aplicações e conteúdos que podem ser explorados no Ensino Médio. Argumenta-se que contemplar a resolução de problemas com modelos de otimização através da resolução geométrica provoca o caráter formativo e o status como ciência que a Matemática possui. Segundo o autor, o auxílio do GeoGebra foi fundamental para se trabalhar com problemas em três dimensões. O autor conclui que o software vem facilitar a visualização geométrica e ser aliado no propósito de despertar o caráter investigativo e desafiador através da resolução de problemas. Por fim, é feita uma sequência de resolução e discussão passo a passo de problemas propostos. O produto educacional consiste de uma sequência de exercícios de programação linear, incluindo a resolução detalhada, com o uso do GeoGebra para fins didáticos.

Nascimento[65] produziu um catálogo de sequências didáticas envolvendo otimização no contexto geométrico, desde problemas clássicos até um jogo criado pelo autor, visando alcançar estudantes do Ensino Básico, para estimular um primeiro contato com a otimização por meio do ensino da Geometria do Ensino Básico. Para instigar o interesse sobre otimização, o autor utiliza-se do ensino da geometria, introduzindo as ideias através de propostas de sequências didáticas que envolvem a confecção de um jogo e também uma história em quadrinhos. Sobre as sequências didáticas, o autor argumenta que elas são adaptáveis a diferentes níveis do ciclo básico, interativas, variadas, propositivas, contextualizadas e acessíveis em todos os sentidos, com estrutura processual completa de aplicação e avaliação, prontas para serem executadas. Um dos maiores empecilhos destacado foi não encontrar registros de tentativas anteriores de uma conexão tão contextualizada e facilitadora entre a aplicação da otimização no contexto geométrico e a Educação Básica. O autor também apresenta a pretensão tanto em aplicar as sequências didáticas em trabalhos futuros quanto continuar explorando outros problemas clássicos e formas de

introduzi-los em séries do Ensino Fundamental anos finais e Ensino Médio da melhor maneira para criar e aplicar novas sequências didáticas. É importante mencionar que o produto educacional produzido por ele constitui-se tanto do catálogo de sequências didáticas envolvendo otimização quanto do jogo e da história em quadrinhos desenvolvida para ser trabalhada em sala de aula com o título "Otimizando Caminhos".

Taveira[66] apresenta em seu trabalho um pouco sobre a evolução da educação brasileira, as habilidades e competências propostas pela BNCC, conceitos importantes sobre otimização e por fim como se pode conciliar, de maneira atrativa, problemas de otimização no ensino básico. No trabalho, explora-se, por meio da resolução de problemas e da ludicidade, metodologias de aulas motivadas pelas versões mais simples de alguns problemas de otimização combinatória, como: O Problema do caixeiro Viajante, O Problema do Carteiro Chinês, O Algoritmo de Dijkstra, O Problema da Mochila e o Problema do Dimensionamento de Lotes. Durante a aplicação das atividades, foram relatadas as seguintes dificuldades: os próprios modelos matemáticos dos problemas de otimização, que utilizam uma notação pouco habitual no contexto do ensino básico; o cronograma, devido ao fato que as aulas demandaram mais tempo do que o inicialmente planejado. O autor discorre que essas dificuldades foram superadas, de modo que, conciliando material lúdico e recursos computacionais, percebeu-se maior interação, interesse e motivação dos alunos. O produto educacional apresentado é a elaboração de atividades juntamente com seus planos de aula sobre diversos problemas de otimização combinatória.

No trabalho de Lima[67] discute-se uma série de problemas clássicos de otimização ligados ao cálculo variacional, com ênfase na braquistócrona. O trabalho procurou apresentar um conteúdo complexo, o Cálculo Variacional, e suas diversas aplicações no dia a dia, como por exemplo, a forma de uma pista de skate do tipo *half-pipe* ou o formato de uma bolha de sabão, entre outras. Os experimentos e problemas de otimização desenvolvidos com os alunos se mostraram como elos, que correlacionam a matemática com suas diversas aplicações. Segundo o autor, o público-alvo se mostrou mais interessado em apreciar a matemática após as primeiras atividades experimentais. O autor também menciona que o problema da braquistócrona e o da "bolha de sabão" foram os experimentos que mais produziram comentários, uma vez que, ambos causaram surpresa e encantamento nos estudantes. Na visão do autor, romper com esses paradigmas com relação a matemática, não necessariamente depende do conteúdo, mas está relacionado ao fato de o docente

preparar ou orientar algo que desperte a curiosidade e o espanto nos alunos. O produto educacional consiste na elaboração de atividades sobre diversos experimentos no contexto de otimização atrelada ao Cálculo Variacional.

4.2 Algoritmos em Diversos Contextos

Nesta parte do trabalho, a palavra de busca nos títulos das dissertações do PROFMAT foi "algoritmo", em que essas buscas foram realizadas no dia 16/06/2022. Foram obtidos 48 registros. Naquele momento, foram selecionados os seis trabalhos mais recentes em que algoritmos são utilizados nos diversos contextos, desde operações feitas com papel e lápis pelo estudante até a programação de computadores, de modo a tentar prover uma visão mais geral do que tem sido feito recentemente no PROFMAT. Além disso, todos os trabalhos selecionados estão diretamente ligados à aplicações na Educação Básica, o que embora seja um objetivo do PROFMAT, nem sempre ocorre.

Na dissertação de Pereira[68] apresentou-se a viabilidade de se abordar tanto a Teoria dos Grafos quanto à programação em linguagem Pascal no contexto dos itinerários formativos relacionados à Matemática e suas Tecnologias conforme evidenciado pela BNCC. Procurando conciliar teoria e prática, foram propostos problemas envolvendo o cotidiano dos alunos, para serem resolvidos em grupo, pretendendo-se, ao final da etapa, o domínio dos conceitos básicos de Pascal e da Teoria dos Grafos, o entendimento dos algoritmos e dos programas em Pascal relativos a caminhos mínimos com o fim de aplicá-los a casos práticos. O autor ressaltou também a dificuldade em encontrar em livros, aulas, trabalhos acadêmicos, etc. um único exemplo prático relacionado ao Algoritmo de Bellman-Ford. Para trabalhos futuros, pretende-se realizar trabalhos similares no contexto de outros temas correlacionados à Teoria dos Grafos, como Árvore Geradora Mínima e Problemas de Coloração, relacionados a problemas como o planejamento de rotas de voos e o controle de tráfego viário. O produto educacional consistiu na elaboração de uma proposta de ensino voltada para o problema de caminhos mínimos, com a respectiva implementação em linguagem Pascal.

Silva[69] apresenta uma discussão sobre os desafios relacionados ao ensino-aprendizagem de Matemática, tanto pelas dificuldades dos alunos quanto dos professores. Nesse contexto, o autor indica métodos, alternativos aos usuais, de efetuar tanto a multiplicação quanto a divisão que possuem, em sua maioria, um contexto histórico que podem representar

aos alunos uma forma de "liberdade" na qual há uma maior possibilidades de efetuar essas operações de formas distintas e de mostrar a eles que a matemática está mais próxima de seu cotidiano do que imaginam, pois cada cultura criou uma forma peculiar e mais familiar possível para suas realidades de modo a lidar com determinados tipos de situações, dessa forma, um mesmo problema pode ser resolvido de formas distintas. O autor também destaca que os métodos apresentados não são os únicos métodos alternativos aos usuais existentes, em que a escolha deles se deu pela possibilidade de serem apresentados aos alunos em um período de tempo não demasiadamente longo e permitirem a verificação de que realmente foram utilizados. Além disso, afirma que o trabalho não pretende em momento algum rejeitar os métodos tradicionais, mas apenas indicar alternativas quando aqueles tradicionais não são plenamente compreendidos. Quanto ao resultado da pesquisa, o autor nota que a diferença entre o volume de acertos das contas diretas relacionado à multiplicação, que, apesar de fatores externos possivelmente terem interferido no resultado, mostrou um avanço significativo dos acertos sendo a maioria pelo método árabe que, de certa forma, é o precursor do algoritmo usual da multiplicação e o método chinês. Apesar dos resultados da pesquisa não demostrar avanços relacionados a divisão, referindo-se ao volume ou tentativas de resoluções, e haver uma equivalência na resolução dos problemas, sendo que o resultado foi proporcionalmente melhor do que o preliminar com a aplicação do método árabe. O autor argumenta que os resultados mostram que houve uma contribuição positiva em relação aos possíveis meios de resolução das operações, principalmente a multiplicação, que permite uma melhor progressão dos assuntos que necessitam dessas bases, de forma, pode ser facilmente utilizado como uma estratégia, por parte dos professores, que vise sanar dificuldades dos alunos sem se tornar repetitivo em pontos que geraram dificuldades e mostrar que há várias possibilidades de se atacar um problema estimulando o raciocínio lógico. O autor também argumenta que em uma eventual contraprova dos resultados desta pesquisa em uma nova aplicação da metodologia proposta, mas com um melhor controle dos fatores externos que podem interferir no experimento, há grande potencial de que os resultados se mostrem bastante satisfatórios, não apenas na multiplicação como também na divisão. No caso específico da divisão, o autor menciona que a grande maioria dos estudantes afirmam possuir dificuldades relacionadas com essa operação. No que tange aos problemas propostos, identifica-se a existência de uma grande dificuldade, por parte dos discentes, em interpretar o enunciado

e elaborar uma estratégia de solução, como esse ponto também não estava definido nos objetivos do trabalho, não houve uma investigação e consequentemente uma proposta que vise minimizar essa problemática. Esse problema é apontado como uma questão para pesquisas futuras. O produto educacional produzido apresenta a indicação de métodos alternativos para as operações aritméticas, sendo que atividades propostas para avaliar o aprendizado com os novos métodos foram aplicadas e utilizadas para a avaliação.

No trabalho de Santos[70] se argumenta que os conhecimentos de algoritmos são fundamentais na formação de qualquer profissional, pois saber organizar os processos e ideias em passos que podem ser executados por pessoas ou computadores é uma habilidade importante na vida cotidiana. Assim, o autor situa seu trabalho como uma tentativa de contribuir para que esse conhecimento seja trabalhado na educação básica dentro dos itinerários formativos de Matemática. O produto educacional produzido consiste na elaboração de uma proposta didática constituída de vários algoritmos implementados em linguagem de programação Java.

Castro[71] apresenta uma visão em que o educador é responsável por tornar a matemática acessível para todos por meio de um processo de aprendizagem prazeroso e significativo. Nesse sentido, é preciso haver a imersão do professor no mundo dos alunos, para que se saiba a melhor maneira de abordar um conteúdo, algum modo de chamar a atenção e mostrar que aquele conhecimento é importante na formação social do mesmo. Além disso, argumenta-se que é de incumbência do professor levar a tecnologia para a sala de aula, mas também é seu papel mostrar que a máquina mais perfeita ainda é o cérebro humano quando atiçado ao conhecimento, pois é capaz de enxergar além. Foi utilizado o software GNU Octave aplicado em temas bastante abordados no ensino fundamental e médio através de exercícios que estimulam a reflexão sobre como chegar num objetivo final, buscando construir uma sequência lógica de passos para tal fim. A partir disso, o aluno também é capaz de correlacionar os algoritmos matemáticos com as atividades que realiza cotidianamente em sua vida. Nesse trabalho, o produto educacional consiste de uma proposta didática constituída de vários algoritmos implementados em linguagem de programação GNU Octave.

Na dissertação de Santos[72] mostrou-se a relação entre a matemática e os avanços tecnológicos, focando na importância de inserir o estudo de algoritmos no contexto da educação básica conforme a BNCC sugere, tratando de forma particular os Algoritmos de

Ordenação. Quanto ao conteúdo matemático utilizado para que a análise dos Algoritmos de Ordenação sugeridos seja realizada de uma maneira correta e sem perda de rigor, verificou-se que o tema favorece o estudo de Séries Aritméticas, bem como o estudo das Funções Afim e Quadrática. O autor propõe que a sequência didática seja um subsídio utilizado por professores do Ensino Médio no processo de ensino e aprendizagem. As competências abordadas na sequência estão relacionadas as ações de representar, comunicar, raciocinar e argumentar. Constatou-se que não existem propostas consistentes para que se utilizem os Algoritmos de Ordenação, com preceitos didáticos, no Ensino Médio e que o tema se restringe, principalmente, aos cursos de graduação da área de computação. De modo que tal sequência surge como alternativa. O produto educacional resultante de tal trabalho consistiu de uma proposta de atividades constituída de vários algoritmos de ordenação.

Pires[73] comparou o resultado do algoritmo utilizado com a rota atual feita pelo transporte escolar de Rio Verde - GO, encontrando uma economia de 2 km em uma das rotas que foi analisada. Existem 8 rotas no total. O algoritmo foi implementado em MATLAB e também se discutiu questões relacionadas à economia financeira, ao bem-estar dos alunos e ganhos de aprendizagem com o menor tempo gasto com o transporte. Ressaltou a necessidade de se considerar a conservação das estradas rurais. Observa-se também, que se fosse para fazer, cálculo por cálculo manualmente dessa rota, levar-se-ia muito tempo calculando algo que o computador faz em minutos. Nesse sentido, poderá ser apresentado para o aluno que a Matemática e a Computação caminham de braços dados e tem atuação importante no seu dia a dia. Percebe-se também que os métodos explicados no trabalho podem ser utilizados em diversos assuntos dentro da sala de aula, em diversos seguimentos do Ensino Básico, tais como: formulação do problema (parte da modelagem matemática), cálculos de distâncias, geometria, com o estudo de ponto, reta, aresta e vértice, além do conteúdo de análise combinatória e probabilidades. Foi ressaltado o grande potencial interdisciplinar de tal estudo, visto que é possível o desenvolvimento de um projeto que englobando Matemática (análise combinatória e probabilidade), Biologia (ecossistema e biodiversidade), Artes (figuras geométricas e assimétricas), Geografia (Vegetação), Português (produção de texto), Educação Física (caminhada); lembrando que, se for feita uma reflexão mais abrangente, provavelmente será possível entrar em conexões com outras disciplinas. O autor também fez uma reflexão sobre as dificuldades encontradas por alunos e professores no uso dessas tecnologias,

principalmente aquelas relacionadas à falta de infraestrutura adequada de muitas escolas. O produto educacional proposto consiste de uma proposta de atividade para resolver problemas de caminho mínimo por meio do Algoritmo da Colônia de Formigas.

Outros trabalhos interessantes que não estão situados dentro do PROFMAT mas merecem destaque nesse contexto de ensino de algoritmos são [74, 75, 76]. Em [74], é feita uma tentativa de analisar aspectos importantes de uma sequência de cursos sobre programação destinado a estudantes de áreas não relacionadas à Ciência da Computação. Embora o trabalho não esteja relacionado propriamente à Educação Básica, é interessante mencionar o fato de que mesmo estudantes do Ensino Superior podem apresentar dificuldades consideráveis no aprendizado de disciplinas relacionadas a algoritmos e programação. A análise realizada no trabalho é baseada em um questionário respondido por estudantes de forma voluntária. Os tópicos dos cursos de programação investigados incluem: a estratégia selecionada para a introdução à programação; a sequência das técnicas de programação e linguagens ensinadas; eventuais dificuldades dos estudantes; o design de ensino utilizado nos cursos; e o material didático utilizado. Baseando-se nas análises dos questionários respondidos e na experiência do professor no ensino dos cursos são apresentadas as conclusões. Os principais resultados apontam que o uso de uma pseudo-linguagem na introdução à programação pode ser uma estratégia promissora, além de se comentar maneiras consideradas mais eficientes de se realizar a transição da programação procedural para a orientada a objetos.

Santos et al.[75] menciona que para se aprender a programar diversas habilidades técnicas são fundamentais, estando elas relacionadas à lógica de algoritmos, à sintaxe de linguagens de programação e à utilização de plataformas computacionais. Diante desses desafios, metodologias de ensino inovadoras têm sido aplicadas no ensino de programação. O trabalho se situa no sentido de de compreender como essas metodologias estão sendo utilizadas. Assim, são apresentados os resultados de uma revisão sistemática da literatura, motivada pela seguinte pergunta de pesquisa: “Quais são as abordagens inovadoras de ensino e aprendizagem de programação, como elas são aplicadas e quais são os principais resultados de sua aplicação?”. A partir da busca em bases de pesquisa consolidadas, 24 estudos primários foram considerados e categorizados em 6 grupos. Os principais desafios destacados estão relacionados aos problemas abordados, ambiente de ensino, conteúdo, capital humano envolvido e processo de avaliação. Os estudos também mostraram

evidências de casos de sucesso, bem como caminhos abertos para novas pesquisas.

Outra revisão sistemática da literatura é apresentada por Silva et al.[76]. Nesse trabalho, a revisão está relacionada a abordagens de aprendizagem e ensino de programação. Os resultados observados mostram que os pesquisadores preferem desenvolver ferramentas para ensinar algoritmos e o foco de pesquisa está principalmente no ensino de programação para a educação superior em um contexto de sala de aula, o que novamente abrange uma certa carência de trabalhos voltados para a Educação Básica. Os resultados também indicam que os trabalhos de pesquisa se concentram no Brasil principalmente nas regiões Sul, Sudeste e Nordeste.

4.3 Algoritmos Evolutivos

Nesta parte do trabalho, as pesquisas foram feitas no dia 10/06/2022 e, inicialmente, não foram encontrados registros envolvendo dissertações cujos títulos apresentassem as palavras “evolutivo” ou “evolutivos”. Assim, optou-se por pesquisar títulos de dissertações que possuíssem a palavra “genético”, na medida em que algoritmos genéticos compõem uma pequena parte do que conhecemos como algoritmos evolutivos. Ao realizar essa busca, foram encontrados apenas dois títulos que estão apresentados a seguir. Isso demonstra uma escassez de trabalhos envolvendo o tema de algoritmos evolutivos no contexto do PROFMAT, o que mostra certo grau de ineditismo da presente dissertação.

O primeiro trabalho aqui mencionado é de autoria de Rosa[77] que se utiliza de técnicas para a resolução de problemas elementares de otimização que podem ser apresentados para alunos do Ensino Básico com conhecimento mínimo sobre funções. Assim, é feito um estudo acerca das técnicas Seção Áurea e Algoritmos Genéticos. São mostrados exemplos de possíveis aplicações das técnicas. No trabalho mencionado são comparados os resultados entre as duas técnicas de otimização para um problema simples: obter o mínimo de uma dada função quadrática. O método de seção áurea é mais rápido e eficiente devido à situação de incertezas menores. A autora aponta que o trabalho apresenta a possibilidade de se trazer novas perspectivas de aprendizagem para o ensino básico, dado que tais técnicas não são foco do ensino médio. Os exemplos de atividades propostas não foram aplicados em sala de aula. Os problemas de otimização utilizados consistiam de funções simples que não necessitavam de algoritmos complexos como algoritmos genéticos para a resolução. O produto educacional consiste de exemplos de atividades de forma detalhada em que se

pode utilizar ambas as técnicas em sala de aula com uso de calculadora. A autora não embasou as atividades em competências/habilidades o que pode estar relacionado ao fato de na época de publicação do trabalho a BNCC ainda estava em construção.

O segundo trabalho é a dissertação de Júnior[78] em que se realiza o ajuste linear para dados experimentais através de um AG. A proposta do trabalho é comparar os resultados de ajuste linear para alguns cenários de controle através dos dois métodos e certificar a qualidade dos ajustes obtidos pelo método aproximado. Assim, com a ajuda de um banco de dados, comparou-se a qualidade das estimativas do método de mínimos quadrados e o algoritmo genético, evidenciando as potencialidades e fraquezas desse método alternativo. Não foi possível identificar propriamente um produto educacional no trabalho.

4.4 Gamificação

A gamificação, a aplicação de elementos de jogos em contextos não relacionados a jogos, tem ganhado destaque como uma abordagem promissora para melhorar a aprendizagem em diversos domínios, incluindo a matemática. Resultados de pesquisas recentes indicam que a gamificação pode melhorar o engajamento dos estudantes, promover a motivação intrínseca, facilitar a compreensão de conceitos matemáticos complexos e aumentar a retenção do conhecimento [9, 10, 11, 12, 13].

Pelo fato de a matemática ser frequentemente percebida como uma disciplina desafiadora e abstrata, muitos estudantes a consideram difícil e desinteressante [11]. A gamificação surge como uma abordagem inovadora para tornar a aprendizagem de matemática mais envolvente, divertida e eficaz. Ao incorporar elementos de jogos, como desafios, recompensas e competição, a gamificação pode estimular o interesse e a motivação, além de promover a aplicação prática dos conceitos matemáticos [79].

Ademais, existe o potencial de aumentar o engajamento dos estudantes na aprendizagem de matemática. Ao transformar os conceitos matemáticos em desafios e atividades interativas, os alunos são incentivados a se envolverem de maneira ativa e a desenvolverem habilidades de resolução de problemas [12]. Além disso, a introdução de elementos de competição saudável, como placares de líderes e conquistas, pode estimular a motivação intrínseca dos estudantes, promovendo uma atitude positiva em relação à matemática.

Outro benefício que vem sendo descoberto é o fato deste tipo de atividade facilitar a compreensão de conceitos matemáticos complexos, fornecendo uma abordagem prática

e visualmente estimulante. Por meio de simulações, jogos e quebra-cabeças interativos, os alunos podem explorar e experimentar os princípios matemáticos de forma concreta, tornando-os mais tangíveis e acessíveis [15, 80]. Isso permite uma melhor compreensão da lógica por trás dos conceitos matemáticos e sua aplicação em situações do mundo real [79].

Estudos têm demonstrado que a gamificação pode melhorar a retenção do conhecimento matemático [14, 15]. Ao envolver os alunos em atividades lúdicas e desafiadoras, a gamificação cria um ambiente propício para a prática repetida e a consolidação dos conceitos aprendidos [14]. Além disso, o retorno imediato e personalizado fornecido pelos jogos permite que os alunos monitorem seu progresso e identifiquem áreas que precisam de aprimoramento, promovendo um aprendizado mais efetivo e duradouro [15].

Nesse contexto de gamificação, destaca-se a tese de Geronimo[81] que foi produzida no contexto do Doutorado em Educação Matemática do Pontifícia Universidade Católica de São Paulo (PUC-SP). Assim, construiu-se uma sequência didática articulada por meio da gamificação para se trabalhar o teorema de Tales. Visando avaliar a relevância para o contexto do ensino, desenvolveu-se uma sequência didática que incorporou a gamificação como estratégia pedagógica e implementou-se essa sequência junto a alunos do nono ano do ensino fundamental. A metodologia utilizada abarcou análises curriculares, revisão de pesquisas e exploração dos conceitos matemáticos. Em seguida, utilizou-se como base a Teoria das Situações Didáticas (TSD) e a Dialética Ferramenta Objeto (DFO) para elaborar e aplicar remotamente a sequência com três alunos do Nono Ano do Ensino Fundamental. Observou-se que mesmo as tarefas sendo realizadas de forma remota, os estudantes foram capazes de agir, formular e validar suas respostas. Eles utilizaram conceitos sobre polígonos e semelhança em diversas situações-problema e foram capazes de empregar seus conhecimentos prévios para compreender o teorema de Tales, indicando que as abordagens utilizadas foram bem-sucedidas na formulação dos desafios e interpretação das respostas apresentadas. Por fim, o autor argumenta que o procedimento foi validado no contexto de aplicação, já que os estudantes conseguiram construir seus conhecimentos sobre o teorema de Tales, enunciá-lo e aplicá-lo, mesmo antes da abordagem institucional. Além disso, enfrentaram as atividades propostas com relativa facilidade, mesmo considerando que eram situações didáticas que poderiam ser consideradas complexas.

Outro trabalho que vale a pena ser mencionado aqui é a dissertação de Boas[82]. O trabalho consiste em uma dissertação de Mestrado em Ciências da Computação produzido

na Universidade Estadual de Londrina em que desenvolveu-se uma Interface de Programação de Aplicativos, denominada GamAPI. A ferramenta tem o propósito,de implementar aspectos de gamificação a partir de premiações baseadas em recompensas, conquistas e níveis. Dois experimentos foram conduzidos para avaliar a eficácia dessa abordagem em um site de ensino. No primeiro experimento, não foi aplicada a Gamificação, enquanto o segundo experimento utilizou-se a interface mencionada. Segundo o autor, os resultados indicaram que a utilização da plataforma com gamificação incentivou os usuários a se engajarem mais em suas atividades.

No contexto do PROFMAT, a palavra de busca nos títulos das dissertações do PROFMAT foi "gamificação", tendo sido obtidos 9 registros no dia 05/07/2022. Naquele momento, foram selecionados dois trabalhos que foram escolhidos pelo fato de serem os dois trabalhos mais recentes no momento da busca que estava associados diretamente à realidade da sala de aula da Educação Básica.

O primeiro trabalho relacionado ao contexto de produções do PROFMAT é de autoria de Serra[83] sendo intitulado "Gamificação e Ensino de Matemática: Proposta de um Jogo para a Aprendizagem de Equações Polinomiais de Primeiro Grau". O objetivo da dissertação é compreender como utilizar jogos na forma de metodologia auxiliar para a aprendizagem, apresentando, assim, um produto educacional. Nesse sentido, o autor contextualiza historicamente o surgimento e o desenvolvimento dos jogos, ressaltando a importância que é atribuída a eles em diferentes culturas. Em seguida, discute-se os principais fatores associadas a motivação e a satisfação em se realizar uma tarefa, especialmente os elementos que tornam os jogos atrativos. A utilização de jogos que não são estritamente educacionais para fins pedagógicos é abordada, conjuntamente são tratados os aspectos relevantes associados a concepção de jogos - mecânicas, estratégias, dinâmicas, etc - e como esses elementos impactam a satisfação dos jogadores. A formação acadêmica dos professores de Matemática no Ensino Básico e as relações com os desafios associados de se ensinar Álgebra, especificamente Equações Polinomiais de 1º Grau, também são pontos avaliados na dissertação. Apresentou-se as razões relacionadas a alguns erros frequentemente cometidos pelos alunos no estudo dessas equações, assim como os princípios matemáticos envolvidos na resolução das mesmas, especialmente os conceitos relacionados ao estudo de Grupos, que fundamentam as técnicas para resolver uma equação. Introduziu-se o desenvolvimento de um jogo que se baseia em uma narrativa de mistério e

aventura, visando promover o desenvolvimento de habilidades essenciais para compreensão e resolução de equações polinomiais de primeiro grau. Por fim, o autor conclui e apresenta as propostas de continuidade, reconhecendo que os processos educacionais demandam formação e aprimoramento contínuos, o que não é diferente no contexto da gamificação voltada para a aprendizagem.

O segundo trabalho no contexto do PROFMAT é intitulado "Xadrez de Sociedade: Do Game à Gamificação", de autoria de Escobar[84]. O trabalho aborda a integração da gamificação na sala de aula, com objetivo de ensinar Matemática de forma contextualizada e interligada a outros conhecimentos. Para este fim, o autor propõe a resolução de problemas matemáticos por meio de games ou gamificações de atividades pedagógicas, estratégia intitulada *Xadrez de Sociedade*. Também se propõe que a atividade seja transformada em uma plataforma digital gamificada. Nesse sentido, são apresentadas sugestões visando adaptar as atividades para o contexto de jogo de modo a implementar o modelo digital gamificado na rotina escolar. Em tal modelo, o estudante recebe retornos sobre o próprio desenvolvimento ao longo dos desafios que lhes são apresentados. O autor enfatiza a distinção entre a gamificação da sala de aula e a simples utilização de um jogo como recurso pedagógico: enquanto empregar jogos como ferramenta representa uma abordagem, a gamificação implica em uma transformação mais abrangente do ambiente de aprendizado. Tais diferenças são bem ilustradas pelo jogo e pelo aplicativo gamificado. Conclui-se que é viável incorporar atividades como gincanas, olimpíadas, explorar recursos tecnológicos, incluir jogos e estruturas de gamificação como um suporte valioso para o ensino de Matemática. O autor finaliza a dissertação apresentando um quadro em que se compara as vantagens e desvantagens de se utilizar a gamificação em sala de aula. Também se ressalta a importância do aluno dominar as tecnologias digitais de forma crítica e reflexiva para participar ativamente dos diferentes contextos sociais, inclusive na escola. Além disso, o trabalho sugere que a utilização de jogos de tabuleiro e a aplicação da gamificação como atividades pedagógicas podem aumentar a motivação e o envolvimento dos alunos na resolução de problemas matemáticos.

Portanto, este capítulo apresentou os trabalhos correlatos e alguns aspectos educacionais inerentes ao objeto de pesquisa desta dissertação. O Capítulo 5 apresenta o produto educacional deste trabalho com detalhes.

5 O Produto Educacional

Este capítulo visa apresentar o produto educacional que envolve a competição de algoritmos evolutivos. Para comparar os algoritmos produzidos e, então, classificá-los, propõe-se uma suíte de testes conforme será apresentado a seguir. De forma simplificada, uma suíte de testes para uma competição de programação com foco em problemas de otimização visa apresentar o conjunto de problemas a serem solucionados, a partir de regras que envolverão os recursos disponibilizados. Nesse sentido, para as funções que deverão ser otimizadas, partiremos, inicialmente, de funções simples e, gradativamente, serão inseridas funções mais complexas. O objetivo é avaliar o desempenho dos algoritmos propostos nesses cenários.

Em geral, AEs são utilizados para lidar com problemas de otimização cada vez mais complexos, e consequentemente, as melhorias nos AEs são importantes. Essas melhorias podem ser verificadas inicialmente a partir do *feedback* sobre seu desempenho em testes conduzidos com funções *benchmark*.

Os participantes devem criar algoritmos, por exemplo, hibridizando elementos conhecidos ou inserindo novos, e testá-los na suíte de testes definida a seguir. Com base nos resultados, o professor apresentará uma análise comparativa que inclui testes estatísticos sobre desempenho de convergência para comparar algoritmos com soluções finais semelhantes. Os algoritmos devem ser implementados em uma mesma linguagem de programação e usando as mesmas bibliotecas como base. Apesar de indicarmos o Python neste trabalho, os organizadores podem adotar outras linguagens que se sintam mais seguros, por exemplo, C ou Matlab/Octave. Alguns códigos em Python estão descritos no Apêndice A. Eles podem ser disponibilizados aos estudantes como códigos-base para que eles próprios modifiquem, ampliem e combinem os trechos de códigos visando construir os seus próprios AEs.

5.1 Definições das Funções que compõe a Suíte de Testes

Nesta seção são apresentadas as funções sugeridas para avaliação do desempenho dos algoritmos que serão produzidos na competição. Tais funções a serem minimizadas são dividas em três grupos principais: funções de uma dimensão, funções bidimensionais e funções de dez dimensões. A escolha das funções que serão apresentadas a seguir foi inspirada nas competições de AEs que ocorrem em um dos eventos mais conhecidos e mundialmente importantes da área de Computação Evolucionária, o IEEE *Congress on Evolutionary Computation* (CEC)[1]. Como problemas de duas e dez dimensões exigem maior grau de abstração do estudante, ficará a cargo do organizador da competição incluir ou não esses problemas na competição, conforme comentaremos mais adiante. Também é importante mencionar que como o presente trabalho é voltado para a educação básica, tem-se problemas significativamente mais simples do que aqueles tipicamente utilizados no evento mencionado.

Antes de prosseguirmos, apresentando especificamente as funções que compõem o suíte de testes, daremos as seguintes definições que serão válidas para todos os problemas a seguir.

Definição do Problema: Todos os problemas serão de minimização conforme se segue

$$\vec{x}^* = \arg\min_{\vec{x}} f(\vec{x}) \text{ sujeito a } \vec{x} \in D_f, \tag{5.1}$$

em que $\vec{x} = (x_1, x_2, \ldots, x_{\mathrm{D}})^{\mathrm{T}}$, sendo que T indica a transposição matricial e $\mathrm{D} \in \mathbb{N}$ denota o número de dimensões do problema cujo domínio ou região de busca é D_f.

Região de Busca: A região de busca D_f apresenta limites inferiores e superiores para todas as D variáveis, ou seja D_f é definida de modo que $-L_i \leq x_j \leq L_s$, $j \in \{1, \ldots, \mathrm{D}\}$ e L_i, $L_s \in \mathbb{R}^+$. Essas restrições de limites superiores e inferiores na região de busca são conhecidas como restrições de caixa. De forma sintética, as restrições de caixa serão representadas como $D_f \subset [-L_i, L_s]^{\mathrm{D}}$, indicando os limites superiores e inferiores para as D dimensões. Como poderá ser visto, na maioria dos problemas $L_i = L_s = 100$, mas em alguns casos, L_i e/ou L_s apresentam valores menores do que 100.

[1] https://ieeexplore.ieee.org/xpl/conhome/1000284/all-proceedings

Ótimo Global: Todos os problemas apresentam um ótimo global $\vec{x}^*$ que está dentro da região de busca, $\vec{x}^* \in D_f$. Devido ao fato de que esses valores podem ser números irracionais, nesses casos, são apresentadas as aproximações considerando 5 casas decimais.

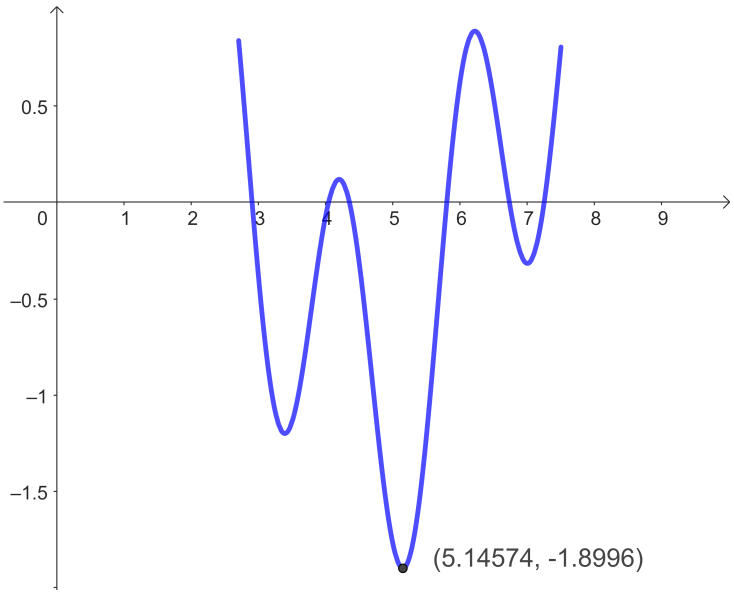
Funções de uma Dimensão

1. Soma de Senoides de Frequências Distintas

$$f_1(x) = \text{sen}(x) + \text{sen}\left(\frac{10}{3}x\right), \quad \text{para} \quad -2{,}7 \leq x \leq 7{,}5 \tag{5.2}$$

Valor ótimo: $x^* \approx 5{,}14574$ e $f_1(x^*) \approx -1{,}8996$.

Figura 5.1: Função soma de senos f_1 (5.2)



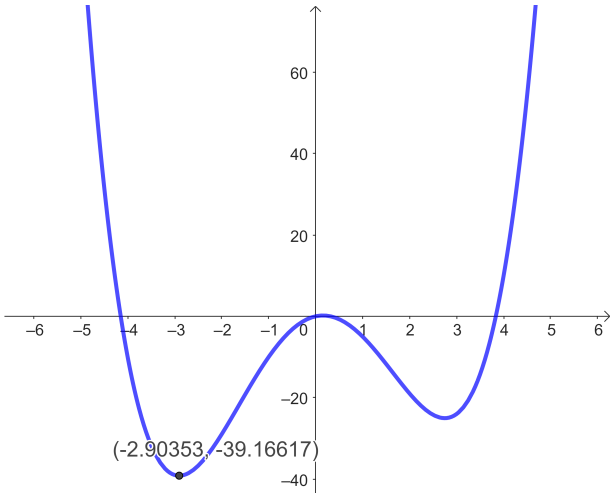
Fonte: Elaborada pelo autor.

2. Função de Styblinski–Tang de uma Dimensão

$$f_2(x) = \frac{x^4 - 16x^2 + 5x}{2}, \quad \text{para} \quad -5 \leq x \leq 5 \tag{5.3}$$

Valor ótimo: $x^* \approx -2{,}90353$ e $f_2(x^*) \approx 39{,}16617$.

Figura 5.2: Função Styblinski–Tang de uma dimensão f_2 (5.3)



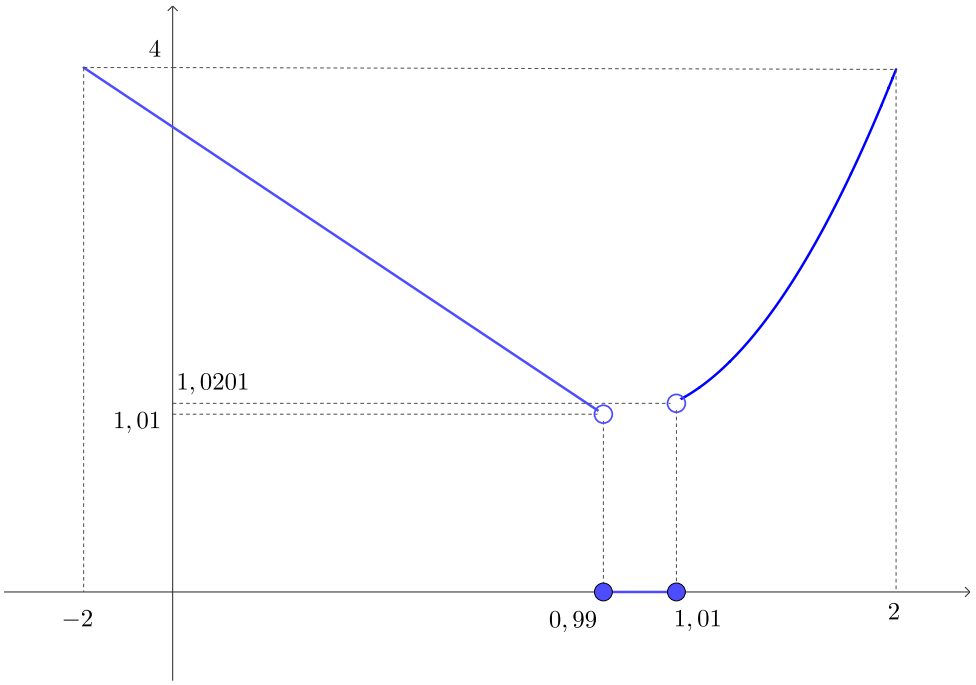
Fonte: Elaborada pelo autor.

3. Função Descontínua

$$f_3(x) = \begin{cases} 2 - x & \text{se } -2 \leq x < 0{,}99 \\ 0 & \text{se } 0{,}99 \leq x \leq 1{,}01 \\ x^2 & \text{se } 1{,}01 < x \leq 2 \end{cases} \tag{5.4}$$

Valor ótimo: $x^* \in [0{,}99; 1{,}01]$ e $f_3(x^*) = 0$.

Figura 5.3: Esboço do Gráfico da Função Descontínua f_3 (5.3)



Fonte: Elaborada pelo autor.

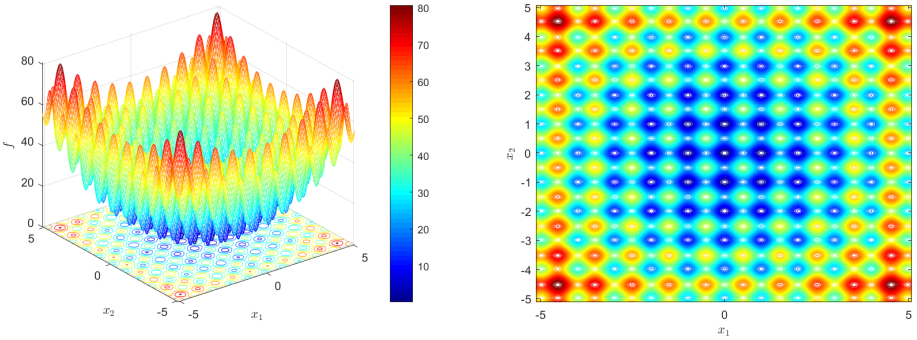
Funções Bidimensionais

4. Função Rastrigin

$$f_4(x_1,x_2) = 20 + x_1^2 - 10\cos(2\pi x_1) + x_2^2 - 10\cos(2\pi x_2)$$
$$\text{para } -5{,}12 \leq x_1,x_2 \leq 5{,}12. \tag{5.5}$$

Valor ótimo: $(x_1^*,x_2^*) = (0,0)$ e $f_4(x_1^*,x_2^*) = 0$.

Figura 5.4: Função Rastringin Bidimensional (5.5). Em (a), tem-se a representação de $f(x_1,x_2)$ e em (b), tem-se o mapa de calor.



(a) (b)

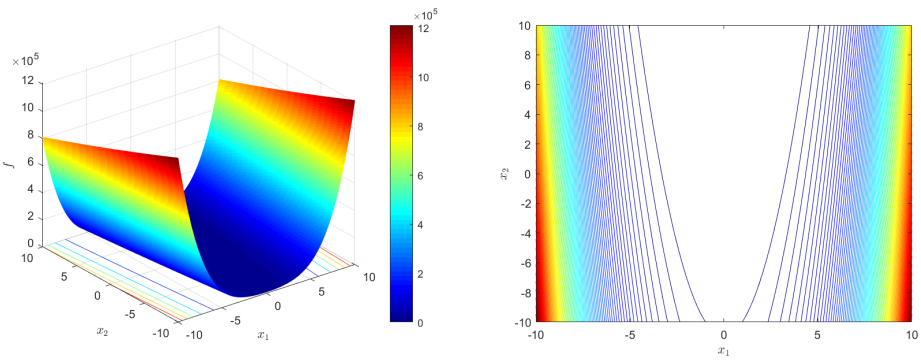
Fonte: Elaborada pelo autor.

5. Função Rosenbrock

$$f_5(x_1,x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad \text{para} \quad -100 < x_1,x_2 < 100 \tag{5.6}$$

Valor ótimo: $(x_1^*,x_2^*) = (1,1)$ e $f_5(x_1^*,x_2^*) = 0$.

Figura 5.5: Função Rosenbrock Bidimensional (5.6). Em (a), tem-se a representação de $f(x_1,x_2)$ e em (b), tem-se as curvas de nível.



(a) (b)

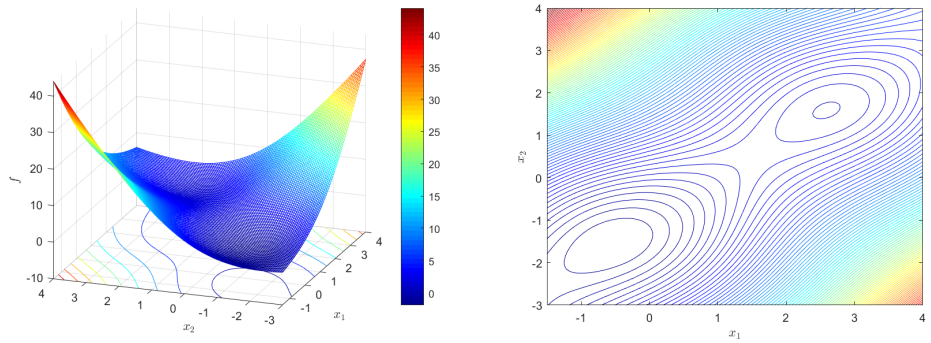
Fonte: Elaborada pelo autor.

6. Função McCormick

$$f_6(x_1,x_2) = \text{sen}(x_1 + x_2) + (x_1 - x_2)^2 - 1{,}5x_1 + 2{,}5x_2 + 1$$
$$\text{para } -1{,}5 \leq x_1 \leq 4; -3 \leq x_2 \leq 4. \tag{5.7}$$

Valor ótimo: $(x_1^*,x_2^*) \approx (-0{,}54719; -1{,}54719)$ e $f_6(x_1^*,x_2^*) \approx -1{,}9133$.

Figura 5.6: Função McCormick Bidimensional (5.7). Em (a), tem-se a representação de $f(x_1,x_2)$ e em (b), tem-se as curvas de nível.



(a) (b)

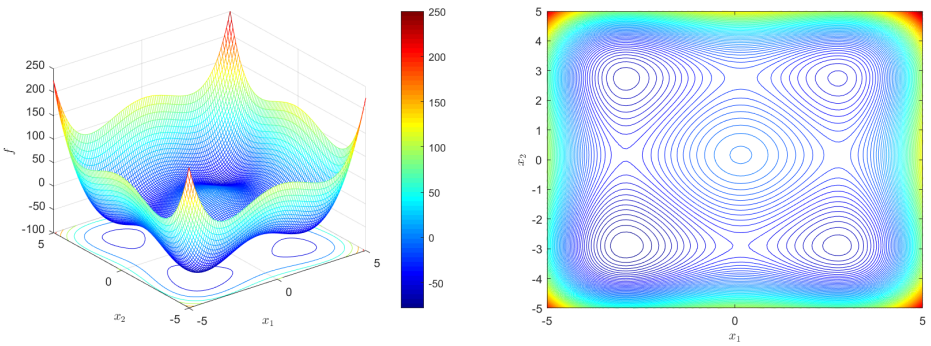
Fonte: Elaborada pelo autor.

7. Função de Styblinski–Tang de duas dimensões:

$$f_7(x) = \frac{x_1^4 - 16x_1^2 + 5x_1 + x_2^4 - 16x_2^2 + 5x_2}{2}, \quad \text{para} \quad -5 \leq x_1,x_2 \leq 5 \tag{5.8}$$

Valor ótimo: $(x_1^*,x_2^*) \approx (-2{,}90353; -2{,}90353)$ e $-78{,}33234 < f_7(x_1^*,x_2^*) < -78{,}33232$.

Figura 5.7: Função Styblinski–Tang Bidimensional (5.8). Em (a), tem-se a representação de $f(x_1,x_2)$ e em (b), tem-se as curvas de nível.



(a) (b)

Fonte: Elaborada pelo autor.

Funções de 10 Dimensões

8. Função Esfera:

$$f_8(x_1,\ldots,x_{10}) = \sum_{i=1}^{10} x_i^2 \quad \text{para} \quad -100 \leq x_1,\ldots,x_{10} \leq 100. \tag{5.9}$$

Valor ótimo: $(x_1^*,\ldots,x_{10}^*) = (0,\ldots,0)$ e $f_8(x_1^*,\ldots,x_{10}^*) = 0$.

9. Função Rastrigin:

$$f_9(x_1,\ldots,x_{10}) = 100 + \sum_{i=1}^{10} \left[x_i^2 - 10\cos(2\pi x_i)\right]$$
$$\text{para} \quad -5{,}12 \leq x_i \leq 5{,}12; \quad 1 \leq i \leq 10. \tag{5.10}$$

Valor ótimo: $(x_1^*,\ldots,x_{10}^*) = (0,\ldots,0)$ e $f_9(x_1^*,\ldots,x_{10}^*) = 0$.

10. Função Rosenbrock:

$$f_{10}(x_1,\ldots,x_{10}) = \sum_{i=1}^{9} \left[100(x_{i+1}^2 - x_i^2)^2 + (1-x_i)^2\right]$$
$$\text{para} \quad -100 \leq x_i \leq 100; \quad 1 \leq i \leq 10. \tag{5.11}$$

Valor ótimo: $(x_1^*,\ldots,x_{10}^*) = (1,\ldots,1)$ e $f_{10}(x_1^*,\ldots,x_{10}^*) = 0$.

5.2 Regras da Competição

Cada participante (ou equipe) deverá propor um único AE para resolver todos os 10 problemas de minimização da suíte de testes. Os AEs devem ser executados 30 vezes em cada problema. Cada execução terá uma semente geradora de números aleatórios definida pelos organizadores da competição. As populações iniciais devem ser geradas aleatoriamente com base nessa semente. É importante executar um AE várias vezes porque um mesmo AE pode evoluir para soluções distintas dependendo da população inicial. Para que a população inicial seja a mesma em uma dada execução de todos os algoritmos, propõe-se, portanto, a utilização da mesma semente geradora. Nesse sentido, para cada um dentre os 30 testes, tem-se uma população inicial distinta. Entretanto, todos os algoritmos usam uma mesma população inicial em cada teste.

O critério de parada de cada execução do AE será determinado pela número máximo de gerações N_{ger} (ou número de iterações) e pelo tamanho da população N_{pop} definidos na Tabela 5.1, fixado ao longo de todas as gerações. Além disso, as equipes deverão realizar, no máximo, o número de avaliações da função objetivo dado pelo produto do número de iterações pelo tamanho da população, valor que também está apresentado na referida tabela.

Tabela 5.1: Número Máximo de Gerações N_{ger}, Tamanho da População N_{pop} e Número Máximo de Avaliações da Função Objetivo.

Função	N_{ger}	N_{pop}	Máximo de Avaliações de f
f_1,f_2,f_3	4	8	32
f_4,f_5	30	30	900
f_6,f_7	10	10	100
f_8	50	30	1500
f_9,f_{10}	200	200	40000

Para definir a pontuação de cada equipe, os organizadores irão proceder como se segue:

1. Cada equipe entregará aos organizadores uma matriz 30×10 com o melhor valor da função objetivo obtido pelo AE utilizado nas 30 execuções de cada um dos 10 problemas. A maneira de entrega dos resultados é sugerida a seguir. Seja $\boldsymbol{R}_m = \{r_{i,j}^m\}_{30\times 10}$ a matriz entregue pela m-ésima equipe, em que o elemento $r_{i,j}^m$ refere-se ao resultado do mínimo obtido ao final da i-ésima execução do j-ésimo problema.

2. Os organizadores farão as diferenças absolutas entre os resultados da matriz $\boldsymbol{R}_m$ e o valor ótimo conhecido da função de cada problema. Seja $\vec{x}_j^* = (x_1^*, \ldots, x_D^*)$ o ponto de ótimo para o j-ésimo problema de dimensão D, cujo valor ótimo da função objetivo será denotado por $f_j(\vec{x}_j^*)$. Assim, a diferença ou erro absoluto para o valor ótimo do i-ésimo teste considerando o j-ésimo problema será calculada como

$$e_{i,j}^m = |f_j(\vec{x}_j^*) - r_{i,j}^m|, \tag{5.12}$$

organizados em uma matriz $\boldsymbol{E}_m = \{e_{i,j}^m\}_{30\times 10}$.

3. Para a m-ésima equipe, os organizadores farão as médias das colunas de $\boldsymbol{E}_m$ para o cálculo de SE_m (soma ponderada dos erros) dada por

$$SE_m = 0{,}2\left(\frac{\sum_{i=1}^{30}\sum_{j=1}^{3} e_{i,j}^m}{90}\right)+0{,}3\left(\frac{\sum_{i=1}^{30}\sum_{j=4}^{7} e_{i,j}^m}{120}\right)+0{,}5\left(\frac{\sum_{i=1}^{30}\sum_{j=8}^{10} e_{i,j}^m}{90}\right). \quad (5.13)$$

4. Em seguida, será calculada a primeira parte da nota para a m-ésima equipe, $N1_m$

$$N1_m = \left(1 - \frac{SE_m - SE_{\min}}{SE_m}\right) \times 50, \quad (5.14)$$

em que $SE_{\min} = \min_{m\in\{1,\ldots,t\}} SE_m$, supondo que existam $t \in \mathbb{N}$ equipes, ou seja, é o menor valor da soma ponderada dos erros considerando todas as equipes.

5. Tendo sido todos os testes executados para as t equipes, para cada execução i de cada problema j, as equipes serão ranqueadas, isto é, classificadas nas posições de 1 a t, de acordo com a ordem crescente de valores de erros absolutos. A m-ésima equipe será classificada na posição $p_{i,j}^m \in \{1,\ldots,t\}$ de acordo com o valor correspondente de erro absoluto $e_{i,j}^m$. Portanto, a melhor equipe terá como posição no ranqueamento o valor 1 e a pior, o valor t.

6. Assim, as posições da m-ésima equipe no ranqueamento serão utilizadas para o cálculo do índice SP_m que equivale a uma média ponderada das posições no ranqueamento

$$SP_m = 0{,}2\left(\frac{\sum_{i=1}^{30}\sum_{j=1}^{3} p_{i,j}^m}{90}\right)+0{,}3\left(\frac{\sum_{i=1}^{30}\sum_{j=4}^{7} p_{i,j}^m}{120}\right)+0{,}5\left(\frac{\sum_{i=1}^{30}\sum_{j=8}^{10} p_{i,j}^m}{90}\right). \quad (5.15)$$

7. A segunda parte da nota para a m-ésima equipe, $N2_m$, está relacionada as posições desta equipe no ranqueamento

$$N2_m = \left(1 - \frac{SP_m - SP_{\min}}{SP_m}\right) \times 50, \quad (5.16)$$

em que $SP_{\min} = \min_{m\in\{1,\ldots,t\}} SP_m$, é o menor valor da soma ponderada das posições no ranqueamento considerando todas as equipes.

8. Portanto, a nota final N_m da m-ésima equipe será dada por

$$N_m = N_1 + N_2. \quad (5.17)$$

9. Por fim, as equipes serão ordenadas em ordem decrescente de notas finais (5.17), sendo que a equipe campeã será aquele que apresentar a maior nota.

Os fatos de se:

- calcular os índices relacionados aos erros absolutos SE_m (5.13) e às posições no ranqueamento SP_m (5.15);
- utilizar esses índices de forma comparativa, considerando os valores mínimos obtidos em todas as execuções (ver cálculos de $N1_m$ (5.14), e $N2_m$ (5.16));
- dividir a nota N_m (5.17) em duas partes igualmente ponderadas $N1_m$ e $N2_m$;

são inspirados em procedimentos previamente utilizados pelas competições do CEC [85], realizadas neste trabalho com menor complexidade. É importante mencionar que o nível de complexidade de entrega de dados e avaliações conforme exige as competições do CEC tornaria a competição demasiadamente complexa para ser aplicada na Educação Básica, dado que essas competições são voltadas para um público de especialistas e estudiosos em Computação Evolucionária que estão na vanguarda deste conhecimento científico. Inclusive, os problemas são mais complexos dos que os propostos neste trabalho, gerando publicações e novos trabalhos científicos.

5.3 Simulando a Competição

Para fazer uma simulação da competição, suponha que 4 equipes estejam concorrendo: A, B, C e D. Todos os algoritmos geram a população inicial de forma aleatória considerando as fronteiras do problema.

A Equipe A escolheu usar uma versão do Algoritmo Genético Simples (AGS), com codificação binária. Uma diferença quanto ao AGS padrão é que, neste exemplo, a seleção de pais para cruzamento é realizada por torneio determinístico, ao invés de roleta. O cruzamento feito com um ponto de corte que é escolhido aleatoriamente. A mutação é realizada pela técnica de *bit-flip* em uma taxa equivalente ao inverso do número de bits, o qual foi definido pela Equipe A em 16. A substituição dos sobreviventes é realizada de forma geracional, sem critérios de elitismo. Mais detalhes sobre o AGS podem ser consultados na Subseção 3.4.7.

A Equipe B escolheu usar a versão mais simples da Evolução Diferencial (Seção 3.5), considerando a estratégia de mutação DE/Rand/1 e cruzamento binomial. Os parâmetros adotados foram $F = 0{,}5$ e $P_c = 0{,}7$.

A Equipe C utiliza um algoritmo que mescla elementos inspirados em ED e AG. Utiliza-se a codificação real e os pais são selecionados por torneio probabilístico, com probabilidade de o mais adaptado vencer o torneio fixada em $P_s = 50\%$. O cruzamento é do tipo aritmético com probabilidade de ocorrência fixada em $P_c = 50\%$. O parâmetro α (3.2) do cruzamento é amostrado de forma aleatória a partir de uma distribuição uniforme delimitada pelo intervalo [0,1]. A mutação é inspirada em DE com estratégia DE/Rand/1 e a probabilidade de ocorrência também fixada em $P_m = 50\%$. É importante ressaltar que tanto os pais quanto os filhos compõem o conjunto de elementos candidatos a sofrerem mutação. O elitismo não é muito forte, mas é realizado ao se preservar, para a próxima geração, o indivíduo mais adaptado da geração anterior, o filho mais bem adaptado e o mutante mais bem adaptado. Os indivíduos restantes são selecionados aleatoriamente dentre os filhos, os mutantes e a população da geração anterior.

A Equipe D também utiliza um algoritmo evolutivo que mistura elementos de ED e AG. Este algoritmo é baseado na codificação real, e realiza-se dois tipos de mutação: a mutação 1 que é do tipo ED/current-to-best/2 e a mutação 2 que consiste em perturbar um indivíduo a partir de uma distribuição normal multivariada. A mutação 1 possui probabilidade de ocorrência definida como P_m e a mutação 2, como $100\% - P_m$. Na mutação 2, $(100\% - P_m)N_{\text{pop}}$ indivíduos aleatoriamente escolhidos são perturbados considerando uma distribuição normal multivariada de média nula e matriz de covariância diagonal $C = \{c_{i,j}\}_{\text{D}\times\text{D}}$ que é construída como se segue para um problema de D dimensões:

$$c_{i,j} = \begin{cases} \sigma_i^2 & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases} \tag{5.18}$$

em que σ_i^2 é a variância da distribuição normal para a i-ésima variável. O desvio padrão σ_i é calculado como $\sigma_i = c|U_i - L_i|$, sendo $c \in [0,1]$, o meta-parâmetro a ser ajustado pelo usuário para permitir que a mutação aumente ou reduza seu potencial de excursionamento, U_i e L_i, são, respectivamente, os valores máximo e mínimo para a i-ésima variável. O procedimento aqui mencionado equivale ao que é descrito em (3.3). Ao final de cada geração, tem-se o seguinte critério de elitismo: o indivíduo mais adaptado da geração

anterior, o mutante do tipo 1 mais bem adaptado e o mutante do tipo 2 mais bem adaptado são preservados para a próxima geração. Além disso, os outros $N_{\text{elt}} - 3$ indivíduos mais bem adaptados são preservados, de modo que $N_{\text{elt}} = T_{\text{elt}} N_{\text{pop}}$ consiste no número de indivíduos mais bem adaptados que são preservados, sendo $0\% \leq T_{\text{elt}} \leq 100\%$, a taxa percentual de preservação via elitismo. Diferentemente dos algoritmos anteriores em que os meta-parâmetros são fixados ao longo de todas as gerações, neste algoritmo, tem-se que os meta-parâmetros podem variar ao longo das gerações como se segue:

- Até 50% das gerações ($k \leq 50\% N_{\text{ger}}$): $P_m(k) = 50\%$, $T_{\text{elt}}(k) = 40\%$, $c(k) = 30\%$;
- Entre 50% e 85% ($50\% < k < 85\% N_{\text{ger}}$): $P_m(k) = 80\%$, $T_{\text{sel}}(k) = 60\%$, $c(k) = 20\%$;
- A partir de 85% ($k \geq 85\% N_{\text{ger}}$): $P_m(k) = 95\%$, $T_{\text{sel}}(k) = 95\%$, $c(k) = 10\%$.

Os códigos dos algoritmos das Equipes A, B, C e D estão disponíveis no Apêndice A. Além disso, tais algoritmos e outras funções úteis para a competição estão disponíveis no link abaixo por meio da plataforma do Google Colab:
https://colab.research.google.com/drive/1n3DvD3aeA6rba4q7PMhAmmT1E6tCJN2W.
Conforme mencionamos no Capítulo 1, tem-se que para construir e manipular esses algoritmos, as equipes lidaram intimamente com as habilidades EM13MAT315 e EM13MAT405 da BNCC que versam sobre o pensamento computacional.

Assim, ao se realizar as 30 execuções dos algoritmos para cada problema, os dados das matrizes de erro $\boldsymbol{E}_A$, $\boldsymbol{E}_B$, $\boldsymbol{E}_C$ e $\boldsymbol{E}_D$ foram armazenados. Na Tabela 5.2, estes resultados são apresentados de forma sintética considerando a média e o desvio padrão das 30 execuções de cada equipe para cada problema. Em seguida, as notas utilizando os critérios descritos na Seção 5.2 são apresentadas na Tabela 5.3. Boxplots das 30 execuções de cada um dos problemas considerando os erros dados nas matrizes $\boldsymbol{E}_A$, $\boldsymbol{E}_B$, $\boldsymbol{E}_C$ e $\boldsymbol{E}_D$ são exibidos nas Figuras 5.8a e 5.8b.

Sobre os resultados obtidos, um ponto importante de se comentar é o fato de que tanto os algoritmos da Equipe C quanto o da Equipe D foram produzidas misturando-se elementos de algoritmos genéticos com elementos de evolução diferencial. Note que uma das misturas, o algoritmo da Equipe C, produziu resultados, de modo geral, inferiores que os algoritmos AG (Equipe A) e ED (Equipe B) simples, o que pode ser percebido através das médias exibidas na Tabela 5.2 e visualmente nos Boxplots (Figuras 5.8a e 5.8b). De modo especial, para os problemas de maior dimensão (f_8, f_9 e f_{10}), o algoritmo da Equipe

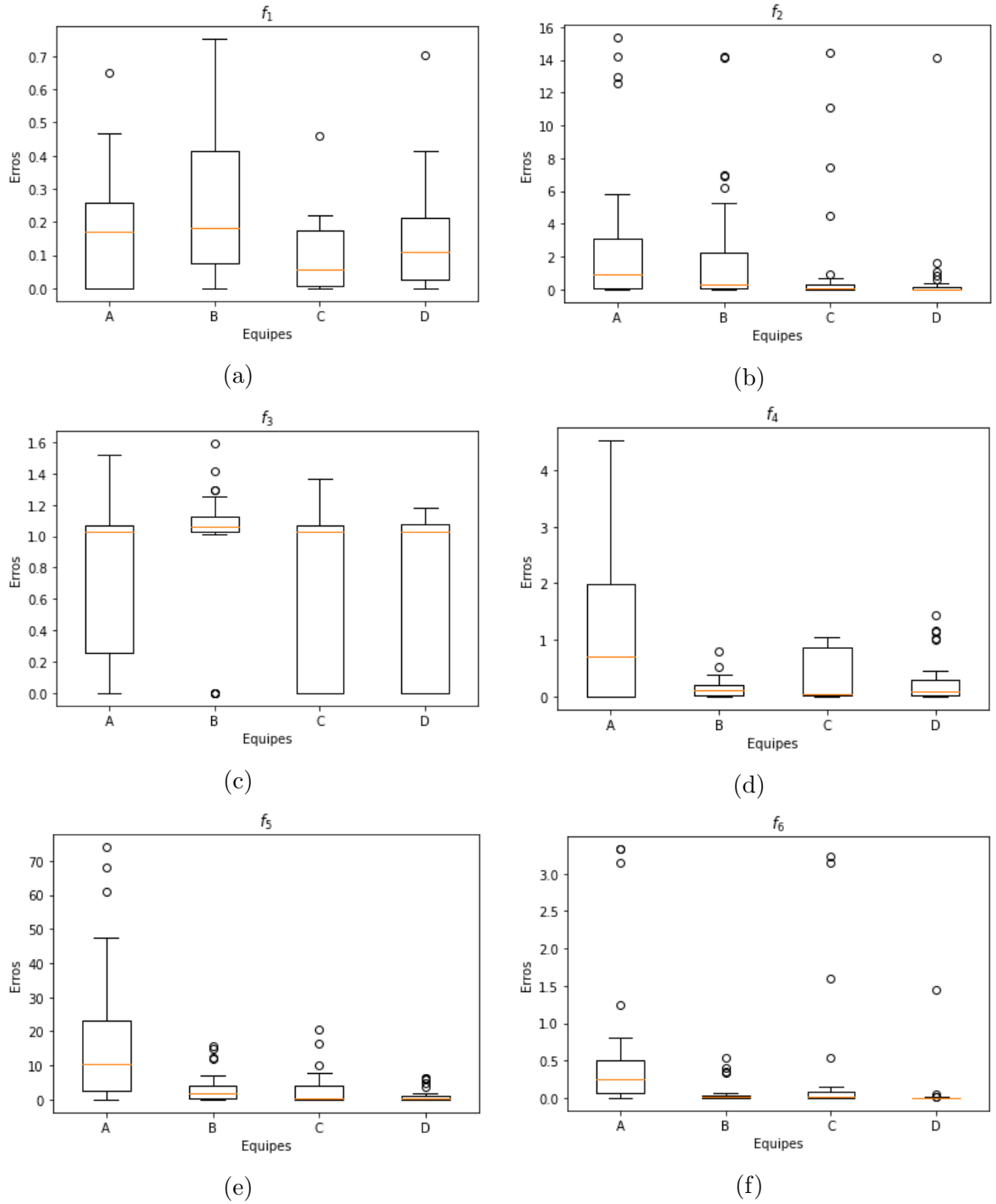
Tabela 5.2: Resultados obtidos para as 30 execuções de cada problema de otimização considerando os erros dados nas matrizes $\boldsymbol{E}_A$, $\boldsymbol{E}_B$, $\boldsymbol{E}_C$ e $\boldsymbol{E}_D$. Os valores estão apresentados na forma $\mu[\sigma]$ em que μ representa a média dos 30 testes para cada problema e σ o respectivo desvio padrão. Em negrito, tem-se o menor valor médio de erro obtido para cada problema.

Função	A	B	C	D
f_1	$1{,}7 \cdot 10^{-1}$ $[1{,}7 \cdot 10^{-1}]$	$2{,}6 \cdot 10^{-1}$ $[2{,}2 \cdot 10^{-1}]$	$\mathbf{9{,}7 \cdot 10^{-2}}$ $[1{,}1 \cdot 10^{-1}]$	$1{,}6 \cdot 10^{0}$ $[1{,}6 \cdot 10^{-1}]$
f_2	$3{,}0 \cdot 10^{0}$ $[4{,}5 \cdot 10^{0}]$	$2{,}3 \cdot 10^{0}$ $[3{,}8 \cdot 10^{0}]$	$1{,}4 \cdot 10^{0}$ $[3{,}4 \cdot 10^{0}]$	$\mathbf{6{,}6 \cdot 10^{-1}}$ $[2{,}5 \cdot 10^{0}]$
f_3	$8 \cdot 10^{-1}$ $[4{,}9 \cdot 10^{-1}]$	$\mathbf{9{,}4 \cdot 10^{-1}}$ $[4{,}4 \cdot 10^{-1}]$	$6{,}8 \cdot 10^{-1}$ $[5{,}2 \cdot 10^{-1}]$	$6{,}8 \cdot 10^{-1}$ $[5{,}2 \cdot 10^{-1}]$
f_4	$1{,}1 \cdot 10^{0}$ $[1{,}3 \cdot 10^{0}]$	$1{,}6 \cdot 10^{-1}$ $[1{,}8 \cdot 10^{-1}]$	$\mathbf{3{,}4 \cdot 10^{-1}}$ $[4{,}3 \cdot 10^{-1}]$	$2{,}8 \cdot 10^{-1}$ $[4{,}1 \cdot 10^{-1}]$
f_5	$1{,}8 \cdot 10^{+1}$ $[2{,}1 \cdot 10^{+1}]$	$3{,}4 \cdot 10^{0}$ $[4{,}5 \cdot 10^{0}]$	$2{,}9 \cdot 10^{0}$ $[4{,}9 \cdot 10^{0}]$	$\mathbf{1{,}4 \cdot 10^{0}}$ $[2{,}1 \cdot 10^{0}]$
f_6	$5{,}9 \cdot 10^{-1}$ $[9{,}4 \cdot 10^{-1}]$	$\mathbf{7{,}0 \cdot 10^{-2}}$ $[1{,}4 \cdot 10^{-1}]$	$3{,}1 \cdot 10^{-1}$ $[8{,}3 \cdot 10^{-1}]$	$5{,}4 \cdot 10^{-2}$ $[2{,}6 \cdot 10^{-1}]$
f_7	$5{,}0 \cdot 10^{0}$ $[4{,}6 \cdot 10^{0}]$	$4{,}2 \cdot 10^{0}$ $[5{,}1 \cdot 10^{0}]$	$6{,}6 \cdot 10^{0}$ $[7{,}9 \cdot 10^{0}]$	$\mathbf{1{,}7 \cdot 10^{0}}$ $[3{,}9 \cdot 10^{0}]$
f_8	$7{,}9 \cdot 10^{+1}$ $[8{,}3 \cdot 10^{+1}]$	$2{,}6 \cdot 10^{+1}$ $[1{,}5 \cdot 10^{+1}]$	$6{,}9 \cdot 10^{+2}$ $[3{,}6 \cdot 10^{+2}]$	$\mathbf{2{,}0 \cdot 10^{+1}}$ $[1{,}6 \cdot 10^{+1}]$
f_9	$7{,}0 \cdot 10^{0}$ $[4{,}5 \cdot 10^{0}]$	$3{,}3 \cdot 10^{+1}$ $[3{,}4 \cdot 10^{0}]$	$7{,}8 \cdot 10^{+1}$ $[1{,}9 \cdot 10^{+1}]$	$\mathbf{5{,}4 \cdot 10^{0}}$ $[2{,}4 \cdot 10^{0}]$
f_{10}	$5{,}0 \cdot 10^{+2}$ $[1{,}0 \cdot 10^{+3}]$	$\mathbf{6{,}3 \cdot 10^{0}}$ $[5{,}7 \cdot 10^{-1}]$	$1{,}1 \cdot 10^{+4}$ $[2{,}1 \cdot 10^{+4}]$	$2{,}2 \cdot 10^{+1}$ $[4{,}2 \cdot 10^{+1}]$

Tabela 5.3: Tabela de Notas para Simulação da Competição. A nota é divida em duas partes, N_1 (5.14) e N_2 (5.16), que somadas geram a nota final $\boldsymbol{N}$ (5.17). Em negrito, tem-se os melhores valores para cada linha da tabela.

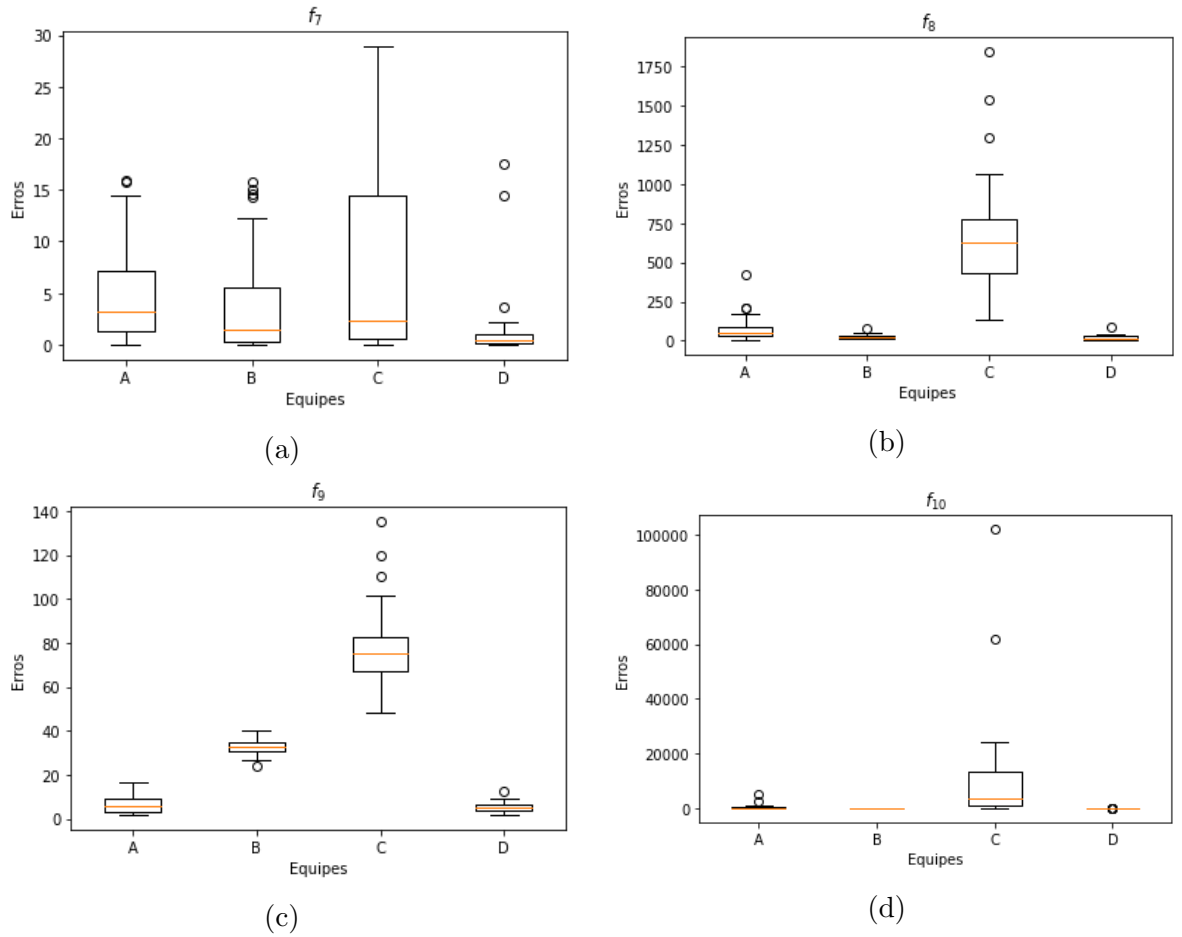
Nota	A	B	C	D
N_1 (5.14)	3,5	34,9	0,2	**50**
N_2 (5.16)	35,0	39,7	29,7	**50**
$\boldsymbol{N}$ (5.17)	38,5	74,6	29,9	**100**
Classificação	3°	2°	4°	**1°**

Figura 5.8: Boxplots das 30 execuções de cada um dos problemas f_1 ao f_6 considerando os erros dados nas matrizes $\boldsymbol{E}_A$, $\boldsymbol{E}_B$, $\boldsymbol{E}_C$ e $\boldsymbol{E}_D$.



Fonte: Elaborada pelo autor.

Figura 5.9: Boxplots das 30 execuções de cada um dos problemas f_7 ao f_{10} considerando os erros dados nas matrizes $\boldsymbol{E}_A$, $\boldsymbol{E}_B$, $\boldsymbol{E}_C$ e $\boldsymbol{E}_D$.



Fonte: Elaborada pelo autor.

C apresentou resultados consideravelmente piores que os outros algoritmos. Por outro lado, o algoritmo da Equipe D venceu a competição, apresentando os melhores resultados em 5 dos 10 problemas (ver Tabela 5.2) e sendo bastante competitivo nos outros 5 problemas. Esse bom desempenho do algoritmo da Equipe D também pode ser visualizado através dos Boxplots (Figuras 5.8a e 5.8b). Portanto, verifica-se que é possível combinar elementos de diversas áreas da computação evolucionária, mas a escolha cuidadosa de quais elementos serão combinados e como eles serão combinados irão impactar de forma significativa os resultados principalmente para problemas de maior dimensão. Nesse sentido, as escolhas feitas para a Equipe C e Equipe D mostraram comportamentos bastante distintos na elaboração do algoritmo. A Equipe C comportou-se apenas conectando elementos distintos de AG e ED, sem utilizar critérios para esta mistura e sem considerar ajuste fino dos parâmetros, dado que todos eles foram colocados com valor de 50%. Por outro lado, a Equipe D realizou a mistura dos elementos com maior critério, inclusive utilizando uma

forma de mutação vinda de AG e outra de ED. Além disso, um ajuste fino dos parâmetros combinado às formas de mutação utilizadas pela Equipe D foram realizados de modo a favorecer exploração ou diversificação das soluções no espaço de busca na fase inicial e o refinamento das soluções na fase final do algoritmo, em que espera-se que as regiões promissoras já tenham sido encontradas. Ademais, os parâmetros são ajustados de modo a existir uma transição entre essas duas fases.

É importante ressaltar que diversos aspectos desta atividade são propícios para se trabalhar as habilidades da BNCC. Além daquelas mencionadas especificamente sobre pensamento computacional, podemos mencionar as habilidades intrínsecas aos conhecimentos de probabilidade e estatística tais como EM13MAT406, EM13MAT407, EM13MAT311 e EM13MAT316. Elas aparecem desde a concepção dos algoritmos a partir da utilização de probabilidades para definir a ocorrência de eventos tais como mutações, cruzamentos e seleções, até a utilização de dados estatísticos oriundos dos experimentos computacionais por meio de medidas de tendências central, medidas de dispersão, gráficos, tabelas e boxplots. Todas as habilidades aqui mencionadas estão descritas em detalhes no Capítulo 1.

Portanto, o presente capítulo apresentou o produto educacional foco desta dissertação. O capítulo seguinte apresenta algumas discussões, comentários e considerações finais sobre o presente trabalho.

6 Conclusões

Este trabalho se dedicou a tentar responder a questão de pesquisa proposta no Capítulo 1, objetivo principal desta dissertação. Assim, foram traçados objetivos secundários também apresentados no referido capítulo. Um desses objetivos secundários consistia em construir um material teórico de boa qualidade que aliasse tanto o rigor necessário quanto uma linguagem mais acessível, de modo a facilitar o entendimento e a aplicação em sala de aula pelo professor interessado em fazê-lo. Nesse sentido, tais fundamentos teóricos foram apresentados nos Capítulos 2, 3 e 4.

O Capítulo 2 discorreu sobre os principais conceitos e definições tipicamente utilizadas na área de Otimização que tem importância para o entendimento sobre problemas de otimização e suas características. Conforme mencionado, o intuito desse capítulo foi o de fornecer um embasamento sólido para o desenvolvimento do produto educacional e para o professor que irá utilizar o produto com finalidade pedagógica.

Uma visão geral sobre a evolução biológica, inspiração dos AEs, características e etapas básicas para o funcionamento desses algoritmos estão no Capítulo 3. Aspectos históricos e os diversos ramos da área de computação evolucionária também são apresentados, além de importantes fundamentos para a implementação de dois algoritmos essenciais para a área - AGS e ED - são discutidos neste ponto do texto. Variações e formas de se aplicar as operações básicas dos algoritmos - cruzamentos, mutações e seleções - também estão presentes no capítulo. Assim, nesse capítulo, outro objetivo secundário relacionado a se estudar aspectos fundamentais envolvendo AEs foi cumprido.

Para situar o trabalho no contexto do PROFMAT e do estado da arte da pesquisa atual com foco em produtos educacionais que envolvam temas como otimização, algoritmos, computação evolucionária e gamificação, tem-se o Capítulo 4. Mostramos nesse capítulo que a utilização de AEs no contexto de pesquisa relacionada à educação básica ainda é incipiente, seja no PROFMAT, seja em um contexto global. Portanto, tal fato serviu como motivação para explorar o potencial dos algoritmos evolutivos como ferramenta

pedagógica de forma inovadora para a educação básica. Nesse sentido, essa revisão de literatura permitiu identificar lacunas no conhecimento e fundamentar a proposta do produto educacional.

A principal contribuição deste trabalho consiste da proposta de um produto educacional que envolve uma competição de algoritmos evolutivos pensada para a educação básica, especialmente no Ensino Médio. Tal contribuição foi obtida ao se buscar responder a questão de pesquisa proposta no Capítulo 1. A ideia de gamificação envolvendo a competição de algoritmos visa estimular o interesse e o aprendizado dos alunos, conforme se discutiu os benefícios deste tipo de estratégia no Capítulo 4. Assim, engajamento, estímulo a criatividade, trabalho em equipe e o pensamento analítico para a resolução de problemas complexos são aspectos interessantes que estão atrelados à aplicação deste trabalho.

Devido a variedade de assuntos complexos envolvidos, a sugestão é que o trabalho seja aplicado para estudantes que já tenham alguma familiaridade com a linguagem de programação que será utilizada. De forma geral, o público que apresenta o potencial de melhor aproveitar o trabalho são estudantes de itinerários formativos do Novo Ensino Médio que estejam relacionados à programação e modalidades de Ensino Médio atreladas a cursos técnicos em que linguagens de programação são previamente trabalhadas. Entretanto, o produto mencionado e apresentado no Capítulo 5 pode ser adaptado, a critério do professor, para as necessidades e/ou características do conjunto de estudantes que estiver trabalhando com o mesmo. Por exemplo, pode ser que não seja adequado utilizar funções de 10 dimensões ou utilizar muitas variantes dos algoritmos. Enfim, esta etapa fica a cargo do professor planejar o que será mais adequado para a turma. Entendemos também que os aspectos interdisciplinares, principalmente aqueles relacionados à Química e à Biologia devem ser estimulados, e sugerimos que possam existir parcerias com os professores dessas disciplinas ao se desenvolver o trabalho no contexto de sala de aula. Por fim, consolidar os conhecimentos e habilidades relativas a aspectos matemáticos que foram mencionadas no Capítulo 1 é um ponto importante de atenção do professor.

Percebendo que ao se iniciar esta pesquisa, novas perguntas e perspectivas de investigação se abrem, a seguir, propõem-se os seguintes estudos como propostas de continuidade e trabalhos futuros:

- Aprimorar o produto educacional por meio da criação de um *framework* com as

principais classes e métodos implementadas previamente, tais como *pymoo*[86] e *PlatEMO*[87] que são dois *frameworks* bem conhecidos da área de otimização multi-objetivo. O objetivo é melhorar a qualidade de um produto educacional específico, utilizando um novo *framework* desenvolvido para essa finalidade. Esse *framework* será projetado para incluir diversas classes e métodos essenciais, já implementados previamente, a fim de facilitar o processo de desenvolvimento.

- Aplicar a atividade proposta no Capítulo 5 em sala de aula e verificar os resultados de aprendizagem no Ensino Básico, como, por exemplo, em cursos técnicos de Informática, que é uma modalidade de Educação Básica, e em Itinerários Formativos do Novo Ensino Médio que versam sobre Matemática e suas Tecnologias.

- Verificar a aplicabilidade da atividade para o Ensino Superior, uma vez que Algoritmos Evolutivos também é tema de estudo em diversos cursos de graduação, tais como Engenharias e Ciências da Computação.

- Neste trabalho, são propostas a utilização apenas de algoritmos inspirados em aspectos da evolução biológica. O uso de algoritmos que envolvem aspectos culturais (ver Reynolds[51] para mais detalhes) e outras formas bioinspiradas baseadas em populações também será explorado em trabalhos futuros. Alguns exemplos de algoritmos bioinspirados baseados em populações são:

    1. Otimização por Enxame de Partículas (PSO, do inglês *Particle Swarm Optimization*), inspirado no comportamento de enxames de pássaros ou cardumes de peixes. Cada partícula atualiza sua posição com base em sua experiência pessoal (melhor posição encontrada) e na experiência coletiva do enxame [88].
    2. Algoritmo de Colônia de Abelhas (ABC, do inglês, *Artificial Bee Colony*), baseado no comportamento de colônias de abelhas, utiliza-se abelhas trabalhadoras que exploram soluções em potencial. As abelhas compartilham informações sobre as soluções encontradas, permitindo uma busca eficiente [89].
    3. Algoritmo de Busca de Alimento de Vaga-lumes (FA, do inglês, *Firefly Algorithm*). Este algoritmo é inspirado no comportamento de vaga-lumes, o FA utiliza a comunicação visual e a atração mútua dos vaga-lumes para otimizar problemas, sendo que os vaga-lumes se movem no espaço de busca, buscando se aproximar dos vaga-lumes mais brilhantes (soluções melhores) [90].

4. Algoritmo de Nuvem de Partículas (PCO, do inglês *Particle Cloud Optimization*): inspirado na formação de nuvens, o PCO utiliza partículas que se movem e interagem entre si no espaço de busca, de modo que as partículas são atraídas por regiões de alta densidade, permitindo uma busca distribuída e eficiente [91].

- Incorporar aspectos que tornam as funções mais complexas, tais como rotações e translações, composições e hibridização entre as funções, tal como é feito nas competições do CEC.

- Expandir os problemas de otimização para que os próprios estudantes realizem a modelagem dos problemas, obtendo as funções objetivo a partir de problemas baseados em problemas do cotidiano.

- Expandir a competição de modo que os próprios estudantes proponham os problemas de otimização que serão resolvidos.

Em suma, espera-se que o produto educacional proposto possa contribuir para o ensino de temas relacionados à otimização e computação evolucionária, despertando o interesse e promovendo uma aprendizagem mais significativa e prazerosa na educação básica.

Referências

1 PAIVA, S. *Introdução à Programação e ao Pensamento Computacional Usando a Linguagem Python e Portugol Studio Univali*. Rio de Janeiro: Editora Ciência Moderna, 2021.

2 BRASIL. MINISTÉRIO DA EDUCAÇÃO. *Base Nacional Curricular Comum*. Brasília, DF, 2018.

3 BRASIL. MINISTÉRIO DA EDUCAÇÃO. *Lei de Diretrizes e Bases da Educação Nacional n. 9394, de 20 de Dezembro de 1996*. Brasília, DF, 1996.

4 CORRÊA, S. D. *O Uso de Métodos Numéricos em Problemas de Otimização*. Dissertação (Mestrado) — Universidade Estadual de Campinas (PROFMAT), 2016.

5 NUNES, C. de A. P. *Otimização como Recurso de Aprendizagem Aplicado ao Ensino Médio e Superior*. Dissertação (Mestrado) — Universidade do Estado do Rio de Janeiro (PROFMAT), 2022.

6 BRANDÃO, M.; SARAMAGO, S. Métodos estocásticos de otimização: algoritmos genéticos e evolução diferencial. *Notas em Matemática Aplicada: SBMAC*, v. 55, 2011.

7 BRASIL, R. M. L. R. F.; SILVA, M. A. da. *Otimização de projetos de engenharia*. São Paulo: Editora Blucher, 2019.

8 EIBEN, A. E.; SMITH, J. E. *Introduction to evolutionary computing*. Berlin: Springer, 2003. v. 53.

9 DOMÍNGUEZ, A. et al. Gamifying learning experiences: Practical implications and outcomes. *Computers & education*, Elsevier, v. 63, p. 380–392, 2013.

10 KIRYAKOVA, G.; ANGELOVA, N.; YORDANOVA, L. Gamification in education. In: TRAKYA UNIVERSITY. *Proceedings of 9th international Balkan education and science conference*. Edirne, Turkey, 2014. v. 1, p. 679–684.

11 BUSARELLO, R. I. *Gamification: princípios e estratégias*. São Paulo: Pimenta Cultural, 2016.

12 ORTIZ-COLÓN, A.-M.; JORDÁN, J.; AGREDAL, M. Gamification in education: An overview on the state of the art. *Educação e Pesquisa*, SciELO Brasil, v. 44, 2018.

13 HUANG, W. H.-Y.; SOMAN, D. Gamification of education. *Report Series: Behavioural Economics in Action*, Rotman School of Management, University of Toronto Toronto, Canada, v. 29, n. 4, p. 37, 2013.

14 DICHEVA, D. et al. Gamification in education: A systematic mapping study. *Journal of educational technology & society*, JSTOR, v. 18, n. 3, p. 75–88, 2015.

15 ZAINUDDIN, Z. et al. The impact of gamification on learning and instruction: A systematic review of empirical evidence. *Educational Research Review*, Elsevier, v. 30, p. 100326, 2020.

16 STATNIKOV, R. B.; MATUSOV, J. B. *Multicriteria optimization and engineering*. New York: Springer Science & Business Media, 2012.

17 TAKAHASHI, R. H. C. Otimização escalar e vetorial vol.1 conceitos preliminares. *Notas de Aula*, 2007. Disponível em: <http://150.164.25.15/~taka/Download/OTEV-Vol1.pdf>.

18 GROSSMANN, I. E. *Global optimization in engineering design*. New York: Springer Science & Business Media, 2013. v. 9.

19 TAVARES, L. A.; ABREU, P.; AGUIRRE, L. A. Estimaçao de Parâmetros de modelos Bouc-Wen via algoritmos evolutivos para compensaçao de histerese. *Anais do SBAI*, v. 14, 2019.

20 TAVARES, L. A.; ABREU, P. E.; AGUIRRE, L. A. Nonlinearity compensation based on identified narx polynomials models. *Nonlinear Dynamics*, Springer, v. 107, n. 1, p. 709–725, 2022.

21 PFLUG, G. C. Stochastic optimization and statistical inference. *Handbooks in operations research and management science*, Elsevier, v. 10, p. 427–482, 2003.

22 ROCKAFELLAR, R. T.; URYASEV, S. The fundamental risk quadrangle in risk management, optimization and statistical estimation. *Surveys in Operations Research and Management Science*, Elsevier, v. 18, n. 1-2, p. 33–53, 2013.

23 CAUNHYE, A. M.; NIE, X.; POKHAREL, S. Optimization models in emergency logistics: A literature review. *Socio-economic planning sciences*, Elsevier, v. 46, n. 1, p. 4–13, 2012.

24 CASTANEDA, J. et al. Optimizing transport logistics under uncertainty with simheuristics: Concepts, review and trends. *Logistics*, MDPI, v. 6, n. 3, p. 42, 2022.

25 CHERRUAULT, Y. Global optimization in biology and medicine. *Mathematical and computer modelling*, Elsevier, v. 20, n. 6, p. 119–132, 1994.

26 PENAS, D. R. et al. Enhanced parallel differential evolution algorithm for problems in computational systems biology. *Applied Soft Computing*, Elsevier, v. 33, p. 86–99, 2015.

27 HANDL, J.; KELL, D. B.; KNOWLES, J. Multiobjective optimization in bioinformatics and computational biology. *IEEE/ACM Transactions on computational biology and bioinformatics*, IEEE, v. 4, n. 2, p. 279–292, 2007.

28 PURSIHEIMO, A. et al. Optimization of statistical methods impact on quantitative proteomics data. *Journal of proteome research*, ACS Publications, v. 14, n. 10, p. 4118–4126, 2015.

29 LEONARD, D.; LONG, N. V.; NGO, V. L. *Optimal control theory and static optimization in economics*. Cambridge, UK: Cambridge University Press, 1992.

30 TAPIA, M. G. C.; COELLO, C. A. C. Applications of multi-objective evolutionary algorithms in economics and finance: A survey. In: IEEE. *2007 IEEE congress on evolutionary computation*. Singapore, 2007. p. 532–539.

31 PONSICH, A.; JAIMES, A. L.; COELLO, C. A. C. A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications. *IEEE Transactions on evolutionary computation*, IEEE, v. 17, n. 3, p. 321–344, 2012.

32 ZAMAN, M. et al. Evolutionary algorithms for dynamic economic dispatch problems. *IEEE Transactions on Power Systems*, IEEE, v. 31, n. 2, p. 1486–1495, 2015.

33 RAMIREZ, A.; ROMERO, J. R.; VENTURA, S. A survey of many-objective optimisation in search-based software engineering. *Journal of Systems and Software*, Elsevier, v. 149, p. 382–395, 2019.

34 ISMAILOV, A.; SOLODOV, M. *Otimização Vol. 1 Condições de Otimalidade, Elementos de Análise Convexa e de Dualidade*. Rio de Janeiro: IMPA, 2005.

35 ISMAILOV, A.; SOLODOV, M. *Otimização Vol. 2 Métodos Computacionais*. Rio de Janeiro: IMPA, 2005.

36 LIMA, E. L. *Análise Real - Volume 2 - Funções de n Variáveis*. Rio de Janeiro: IMPA, 1999. v. 5.

37 STEWART, J. *Cálculo*. 5. ed. São Paulo: Cengage Learning, 2006. v. 1.

38 FOGEL, L. J. Artificial intelligence through a simulation of evolution. In: UNIVERSITY OF SOUTHERN CALIFORNIA. *Proc. of the 2nd Cybernetics Science Symp., 1965*. Los Angeles, 1965.

39 HOLLAND, J. H. Genetic algorithms and the optimal allocation of trials. *SIAM journal on computing*, SIAM, v. 2, n. 2, p. 88–105, 1973.

40 HOLLAND, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge, MA: MIT press, 1992.

41 RECHENBERG, I. Evolutionsstrategie. *Optimierung technischer Systeme nach Prinzipien derbiologischen Evolution*, Frommann-Holzboog Verlag, 1973.

42 SCHWEFEL, H.-P. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: mit einer vergleichenden Einführung in die Hill-Climbing-und Zufallsstrategie*. Basel: Springer, 1977. v. 1.

43 HOWARD, J. *Darwin*. São Paulo: Edições Loyola, 1982.

44 RIDLEY, M. *Evolução*. Porto Alegre: Artmed Editora, 2009.

45 LOPES, S. G. B. C. *Bio: introdução a biologia, citologia, embriologia animal, histologia animal, os seres vivos, genética, evolução e ecologia*. São Paulo: Saraiva, 1994.

46 REECE, J. et al. *Biologia de Campbell*. Porto Alegre: Artmed, 2022. v. 12. ISBN 9788582712306.

47 SCHWEFEL, H. Evolution and optimum seeking wiley interscience. *New York*, 1995.

48 BANZHAF, W. et al. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. San Francisco: Morgan Kaufmann Publishers Inc., 1998.

49 BACK, T. Evolutionary programming and evolution strategies: Similarities and differences. In: MORGAN KAUFMANN PUBLISHERS INC. *Proc. of the Second Ann. Conf. on Evolutionary Programming*. San Francisco, CA, 1993. p. 11–22.

50 PRICE, K. V. Differential evolution. *Handbook of Optimization: From Classical to Modern Approach*, Springer, p. 187–214, 2013.

51 REYNOLDS, R. G. An introduction to cultural algorithms. In: WORLD SCIENTIFIC. *Proceedings of the 3rd annual conference on evolutionary programming, World Scientific Publishing*. San Diego, CA, 1994. p. 131–139.

52 KOZA, J. R. Genetic evolution and co-evolution of computer programs. *Artificial life II*, Citeseer, v. 10, p. 603–629, 1991.

53 FOGEL, D. B. The advantages of evolutionary computation. *Bcec*, p. 1–11, 1997.

54 GOLBERG, D. E. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, v. 1989, n. 102, p. 36, 1989.

55 MATHIAS, K.; WHITLEY, L. Transforming the search space with gray coding. In: IEEE. *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. Orlando, USA, 1994. v. 1, p. 513–518.

56 HAUPT, R. L.; HAUPT, S. E. *Practical genetic algorithms*. New York: John Wiley & Sons, 2004.

57 WU, G. et al. Ensemble of differential evolution variants. *Information Sciences*, Elsevier, v. 423, p. 172–186, 2018.

58 ZHANG, J.; SANDERSON, A. C. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, IEEE, v. 13, n. 5, p. 945–958, 2009.

59 NERI, F.; TIRRONEN, V. Recent advances in differential evolution: a survey and experimental analysis. *Artificial intelligence review*, Springer, v. 33, p. 61–106, 2010.

60 SLOWIK, A.; KWASNICKA, H. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, Springer, v. 32, p. 12363–12379, 2020.

61 CASTRO, R. L. de. *Problemas de Otimização e Aplicações ao Estudo de Funções no Ensino Médio*. Dissertação (Mestrado) — Universidade Federal da Paraíba (PROFMAT), 2015.

62 FERNANDES, D. C. *Problemas de Otimização e Aplicações ao Estudo de Funções no Ensino Médio*. Dissertação (Mestrado) — Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP - PROFMAT), 2020.

63 SALDANHA, I. V. *Estratégias de Resolução de Problemas de Otimização*. Dissertação (Mestrado) — Universidade Federal do Ceará (PROFMAT), 2020.

64 SENA, A. S. *Método Geométrico para Resolução de Problemas de Otimização em 3D e o Uso do Geogebra*. Dissertação (Mestrado) — Universidade Federal do Recôncavo Baiano (PROFMAT), 2021.

65 NASCIMENTO, A. P. D. do. *Conhecendo o Mundo da Otimização: Propostas de Introdução da Otimização Geométrica no Ensino Básico*. Dissertação (Mestrado) — Universidade Federal Rural de Pernambuco (PROFMAT), 2021.

66 TAVEIRA, R. C. L. *Ensino de Matemática por Meio de Problemas Clássicos de Otimização Combinatória*. Dissertação (Mestrado) — Universidade Federal do Triângulo Mineiro (PROFMAT), 2021.

67 LIMA, J. A. F. S. *Introdução ao Cálculo Variacional e Problemas de Otimização Aplicados no Ensino Básico*. Dissertação (Mestrado) — Universidade Federal de Goiás (PROFMAT), 2019.

68 PEREIRA, M. D. de C. *Grafo e o Problema do Caminho Mínimo: Algoritmo e Programação em Pascal*. Dissertação (Mestrado) — Universidade Federal Rural do Rio de Janeiro (PROFMAT), 2022.

69 SILVA, E. A. da. *Métodos Alternativos de Cálculos na Multiplicação e Divisão: para além dos Algoritmos Usuais da Aritmética*. Dissertação (Mestrado) — Universidade Federal do Pará (PROFMAT), 2021.

70 SANTOS, N. S. *Alguns Algoritmos em Java para Matemática Básica*. Dissertação (Mestrado) — Universidade Estadual de Feira de Santana (PROFMAT), 2020.

71 CASTRO, S. A. D. *Algoritmos e Pensamento Computacional como Ferramenta no Processo de Ensino-Aprendizagem*. Dissertação (Mestrado) — Universidade Federal do Piauí (PROFMAT), 2020.

72 SANTOS, I. F. dos. *Algoritmos de Ordenação: uma Abordagem Didática para o Ensino Médio*. Dissertação (Mestrado) — Universidade Federal Rural de Pernambuco (PROFMAT), 2020.

73 PIRES, D. E. *Uma Proposta de Interdisciplinaridade Utilizando Análise Combinatória e o Algoritmo de Colônia de Formigas no Ensino Médio*. Dissertação (Mestrado) — Universidade Federal de Goiás (PROFMAT), 2020.

74 XINOGALOS, S. Designing and deploying programming courses: Strategies, tools, difficulties and pedagogy. *Education and Information Technologies*, Springer, v. 21, p. 559–588, 2016.

75 SANTOS, S. C. et al. Innovative approaches in teaching programming: A systematic literature review. In: SPRINGER. *Proceedings of the 12th International Conference on Computer Supported Education*. Virtual Event, 2020. v. 1, p. 205–214.

76 SILVA, T. R. da et al. Ensino-aprendizagem de programação: uma revisão sistemática da literatura. *Revista Brasileira de Informática na Educação*, v. 23, n. 01, p. 182, 2015.

77 ROSA, A. C. *Uma Introdução à Problemas de Otimização Utilizando o Método da Seção Áurea e Algoritmos Genéticos*. Dissertação (Mestrado) — Universidade Federal de Goiás (PROFMAT), 2016.

78 JÚNIOR, E. L. S. *Uso de Algoritmo Genético no Ajuste Linear Através de Dados Experimentais*. Dissertação (Mestrado) — Universidade Federal de Paraíba (PROFMAT), 2015.

79 ALT, D. Assessing the benefits of gamification in mathematics for student gameful experience and gaming motivation. *Computers & Education*, Elsevier, v. 200, p. 104806, 2023.

80 ZAHARIN, F. Z. et al. Gamification in mathematics: Students' perceptions in learning perimeter and area. *Jurnal Pendidikan Sains Dan Matematik Malaysia*, v. 11, p. 72–80, 2021.

81 GERONIMO, R. R. *Uma proposta para o ensino do teorema de Tales com gamificação*. Tese (Doutorado) — Pontifícia Universidade Católica de São Paulo, 2021.

82 BOAS, J. L. V. *Contribuição da GamAPI na Efetivação dos Conceitos de Gamificação e Gerência em Ambientes de Treinamento*. Dissertação (Mestrado) — Universidade Estadual de Londrina, Londrina, 2016. Mestrado em Ciência da Computação.

83 SERRA, C. H. R. *Gamificação e ensino de matemática: proposta de um jogo para a aprendizagem de equações polinomiais de primeiro grau*. Dissertação (Mestrado) — Centro Federal de Educação Tecnológica de Minas Gerais (PROFMAT), 2022.

84 ESCOBAR, D. M. *Xadrez de Sociedade: do Game à Gamificação*. Dissertação (Mestrado) — Universidade de Brasília (PROFMAT), 2021.

85 ŠKVORC, U.; EFTIMOV, T.; KOROŠEC, P. CEC real-parameter optimization competitions: Progress from 2013 to 2018. In: IEEE. *2019 IEEE congress on evolutionary computation (CEC)*. Wellington, New Zealand, 2019. p. 3126–3133.

86 Blank, J.; Deb, K. pymoo: Multi-objective optimization in python. *IEEE Access*, v. 8, p. 89497–89509, 2020.

87 TIAN, Y. et al. PlatEMO: A MATLAB platform for evolutionary multi-objective optimization. *IEEE Computational Intelligence Magazine*, v. 12, n. 4, p. 73–87, 2017.

88 WANG, D.; TAN, D.; LIU, L. Particle swarm optimization algorithm: an overview. *Soft computing*, Springer, v. 22, p. 387–408, 2018.

89 KARABOGA, D.; AKAY, B. A comparative study of artificial bee colony algorithm. *Applied mathematics and computation*, Elsevier, v. 214, n. 1, p. 108–132, 2009.

90 FISTER, I. et al. A comprehensive review of firefly algorithms. *Swarm and Evolutionary Computation*, Elsevier, v. 13, p. 34–46, 2013.

91 ZHANG, Y.; WANG, S.; JI, G. A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical problems in engineering*, Hindawi, v. 2015, 2015.

Apêndice A
Principais códigos de AEs em Python

Abaixo estão listados os principais códigos de AEs em *Python* para competição de algoritmos. Uma lista mais completa e detalhada está disponível no link abaixo:

https://colab.research.google.com/drive/1n3DvD3aeA6rba4q7PMhAmmT1E6tCJN2W.

Nos comentários, as equações são citadas conforme numeração da dissertação sendo apresentadas entre parêntesis.

A.1 Funções a Serem Minimizadas

```
import numpy as np

# Conjunto de funcoes a serem minimizadas
# Cada funcao possui duas entradas:
# x: argumento a ser avaliado pela funcao objetivo
# cont: quantidade de vezes que a funcao objetivo ja foi avaliada

#f1 - Funcao Soma de Senoides - (5.2)
def f1(x,cont):
    fronteiras = [[-2.7, 7.5]]
    f1.fronteiras =fronteiras
    f1.y = []
    f1.cont = []
    if x != [] and x[0] >=fronteiras[0][0] and x[0] <=fronteiras[0][1]:
        f1.y = np.sin(x[0])+np.sin(10/3*x[0])
        f1.cont = cont+1
    return f1

#f2 - Funcao de Styblinski-Tang de uma dimensao - (5.3)
```

```
def f2(x,cont):
    fronteiras = [[-5, 5]]
    f2.fronteiras =fronteiras
    f2.y = []
    f2.cont = []
    if x != [] and x[0] >=fronteiras[0][0] and x[0] <=fronteiras[0][1]:
        f2.y = (x[0]**4 - 16*x[0]**2 + 5*x[0]) / 2
        f2.cont = cont+1
    return f2

#f2 - Funcao Descontinua - (5.4)
def f3(x,cont):
    fronteiras = [[-2, 2]]
    f3.fronteiras =fronteiras
    f3.y = []
    f3.cont = []
    if x != []:
        f3.cont = cont+1
        if -2 <= x[0] and x[0] < 0.99:
            f3.y = 2 - x[0]
        elif x[0] >= 0.99 and x[0] <= 1.01:
            f3.y = 0
        elif x[0] > 1.01 and x[0] <= 2:
            f3.y = x[0]**2

    return f3

#f4 - Funcao Rastringin de uma dimensao - (5.5)
def f4(x,cont):
    fronteiras = [[-5.12, 5.12],[-5.12, 5.12]]
    f4.fronteiras =fronteiras
    f4.y = []
```

```
    f4.cont = []
    if x != [] and x[0] >=fronteiras[0][0] and x[0] <=fronteiras[0][1]
   and x[1] >=fronteiras[1][0] and x[1] <=fronteiras[1][1]:
        f4.cont = cont+1
        f4.y = 20+x[0]**2-10*np.cos(2*np.pi*x[0])+x[1]**2-10*np.cos(2*
   np.pi*x[1])
    return f4

#f5 - Funcao Rosenbrock de uma dimensao - (5.6)
def f5(x,cont):
    fronteiras = [[-100, 100],[-100, 100]]
    f5.fronteiras =fronteiras
    f5.y = []
    f5.cont = []
    if x != [] and x[0] >=fronteiras[0][0] and x[0] <=fronteiras[0][1]
   and x[1] >=fronteiras[1][0] and x[1] <=fronteiras[1][1]:
        f5.cont = cont+1
        f5.y = 100*(x[1]-x[0]**2)**2+(1-x[0])**2
    return f5

#f6 - Funcao McCormick - (5.7)
def f6(x,cont):
    fronteiras = [[-1.5, 4],[-3, 4]]
    f6.fronteiras =fronteiras
    f6.y = []
    f6.cont = []
    if x != [] and x[0] >=fronteiras[0][0] and x[0] <=fronteiras[0][1]
   and x[1] >=fronteiras[1][0] and x[1] <=fronteiras[1][1]:
        f6.cont = cont+1
        f6.y = np.sin(x[0]+x[1])+(x[1]-x[0])**2-1.5*x[0]+2.5*x[1]+1
    return f6
#f7 - Funcao de Styblinski-Tang de duas dimensoes - (5.8)
```

```
def f7(x,cont):
    fronteiras = [[-5, 5],[-5, 5]]
    f7.fronteiras =fronteiras
    f7.y = []
    f7.cont = []
    if x != [] and x[0] >=fronteiras[0][0] and x[0] <=fronteiras[0][1]
   and x[1] >=fronteiras[1][0] and x[1] <=fronteiras[1][1]:
        f7.cont = cont+1
        f7.y = (x[0]**4 - 16*x[0]**2 + 5*x[0]+x[1]**4 - 16*x[1]**2 + 5*x
   [1]) / 2
    return f7

#f8 - Funcao Esfera de duas dimensoes - (5.8)
def f8(x,cont):
    fronteiras = [[-100, 100],[-100, 100],[-100, 100],[-100,
   100],[-100, 100],[-100, 100],[-100, 100],[-100, 100],[-100,
   100],[-100, 100]]
    f8.fronteiras =fronteiras
    f8.y = []
    f8.cont = []
    if x != []:
        f8.cont = cont+1
        f8.y = x[0]**2+x[1]**2+x[2]**2+x[3]**2+x[4]**2+x[5]**2+x[6]**2+x
   [7]**2+x[8]**2+x[9]**2
    return f8

#f9 - Funcao Rastringin de duas dimensoes - (5.9)
def f9(x,cont):
    fronteiras = [[-100, 100],[-100, 100],[-100, 100],[-100,
   100],[-100, 100],[-100, 100],[-100, 100],[-100, 100],[-100,
   100],[-100, 100]]
    f9.fronteiras =fronteiras
```

```
    f9.y = []
    f9.cont = []
    if x != []:
        f9.cont = cont+1
        result = 100
        for i in range(10):
            if  x[i] >=fronteiras[i][0] and x[0] <=fronteiras[i][1]:
                result += x[i]**2-10*np.cos(2*np.pi*x[i])
            else:
                f9.y = []
                return f9.y
        f9.y = result
    return f9

#f10 - Funcao Rosenbrock de duas dimensoes - (5.10)
def f10(x,cont):
    fronteiras = [[-100, 100],[-100, 100],[-100, 100],[-100,
   100],[-100, 100],[-100, 100],[-100, 100],[-100, 100],[-100,
   100],[-100, 100]]
    f10.fronteiras =fronteiras
    f10.y = []
    f10.cont = []
    if x != []:
        f10.cont = cont+1
        if  x[0] >=fronteiras[0][0] and x[0] <=fronteiras[0][1]:
            result = 0.0
            for i in range(9):
                if  x[i+1] >=fronteiras[i+1][0] and x[0] <=fronteiras[i
   +1][1]:
                    result += 100 * (x[i+1] - x[i]**2)**2 + (1 - x[i])
   **2
                else:
```

```
                    f10.y = []
                    return f10.y
            f10.y = result
    return f10
```

A.2 AG Simples – Algoritmo Equipe A

```
# Funcoes relativas a EQUIPE A — AG Simples
# Implementacao baseada no codigo disponivel em
# https://machinelearningmastery.com/simple-genetic-algorithm-from-
    scratch-in-python/

# Decodifica palavra binaria para base 10
def decode(fronteiras, n_bits, bitstring):
    #fronteiras: limites inferiores e superiores de cada variavel
    #n_bits: numero de bits
    #bitstring: palavra binaria que sera decodificada

    decoded = list()
    largest = 2**n_bits
    for i in range(len(fronteiras)):
        # Extrai a substring
        start, end = i * n_bits, (i * n_bits)+n_bits
        substring = bitstring[start:end]
        # Converte bitstring para string de chars
        chars = ''.join([str(s) for s in substring])
        # Converte string para inteiro
        integer = int(chars, 2)
        # Ajusta inteiro para range desejado
        value =fronteiras[i][0] + (integer/largest) * (fronteiras[i][1]
    -fronteiras[i][0])
        # Salva e retorna o valor decodificado
```

```
        decoded.append(value)
    return decoded

# Selecao por torneio
def torneio(pop, scores, k=3):
    # pop: populacao de individuos
    # scores: valores da funcao objetivo de pop
    # k: tamanho do torneio

    # Primeira selecao aleatoria
    selection_ix = randint(len(pop))
    for ix in randint(0, len(pop), k-1):
        # Checa melhora (aplica torneio)
        if scores[ix] < scores[selection_ix]:
            selection_ix = ix
    # Retorna o individuo vencedor do torneio
    return pop[selection_ix]

# Cruzamento - crossover binario
def crossover_1ponto(p1, p2, r_cross):
    # p1: codigo genetico do primeiro pai
    # p2: codigo genetico do segundo pai
    # r_cross: probabilidade de cruzamento

    # filhos sao copias dos pais por padrao
    c1, c2 = p1.copy(), p2.copy()
    # verifica se ocorrera cruzamento
    if rand() < r_cross:
        # seleciona o ponto de corte
        pt = randint(1, len(p1)-2)
        # aplica o cruzamento
        c1 = p1[:pt] + p2[pt:]
```

```
            c2 = p2[:pt] + p1[pt:]
        # retorna os filhos
        return [c1, c2]

# Operador de Mutacao
def mutacao_bit_flip(bitstring, r_mut):
    # bitstring: indidivuo codificado em binario
    # r_mut: probabilidade de mutacao

    for i in range(len(bitstring)):
        # verifica se ocorrera cruzamento
        if rand() < r_mut:
            # troca o bit
            bitstring[i] = 1 - bitstring[i]
    # Retorna o individuo codificado mutado
    return bitstring

# Algoritmo Genetico Simples (ver Secao 3.4 para mais detalhes)
def algoritmo_genetico_simples(objective,fronteiras, n_bits, n_iter,
   n_pop, r_cross, r_mut):
    #objective: funcao objetivo a ser otimizada
    #fronteiras: limites inferiores e superiores de cada variavel
    #n_bits: numero de bits da codificacao (deve ser inteiro positivo)
    #n_pop: tamanho da populacao (deve ser inteiro positivo)
    #r_cross: probabilidade de cruzamento (deve ser real entre 0 e 1)
    #r_mut: probabilidade de cruzamento (deve ser real entre 0 e 1)

    # Populacao inicial de bitstrings geradas aleatoriamente
    pop = [randint(0, 2, n_bits*len(fronteiras)).tolist() for _ in range
   (n_pop)]
    best_eval = math.inf
    best = []
```

```
    scores = np.zeros(n_pop)
    # Inicializa contador de avaliacoes da funcao objetivo
    numero_avaliacoes_fo = 0
    # enumerate generations
    for gen in range(n_iter):
        # Decodifica populacao
        decoded = [decode(fronteiras, n_bits, p) for p in pop]
        # Avalia todos os individuos da populacao e procura nova solucao
    otima
        for i in range(n_pop):
            avaliacao = objective(decoded[i],numero_avaliacoes_fo)
            numero_avaliacoes_fo = avaliacao.cont
            scores[i] = avaliacao.y
            if scores[i] < best_eval:
                best, best_eval = pop[i], scores[i]
        # Seleciona os pais
        selected = [torneio(pop, scores) for _ in range(n_pop)]
        # Produz a proxima geracao
        children = list()
        for i in range(0, n_pop, 2):
            # Seleciona os pais em pares
            p1, p2 = selected[i], selected[i+1]
            # Aplica cruzamento e mutacao
            for c in crossover_1ponto(p1, p2, r_cross):
                # Aplica mutacao
                mutacao_bit_flip(c, r_mut)
                # Salva os filhos para a proxima geracao
                children.append(c)
        # Substitui toda a populacao
        pop = children
    return [best, best_eval,numero_avaliacoes_fo]
```

```
def equipeA(fo,n_iter,n_pop):
    #fo: funcao objetivo
    #n_iter: numero de geracoes (deve ser inteiro positivo)
    #n_pop: tamanho da populacao (deve ser inteiro positivo)

    fronteiras = fo([],0).fronteiras
    # Bits por variavel
    n_bits = 16
    # Taxa de crossover
    r_cross = 0.9
    # Taxa de mutacao
    r_mut = 1.0 / (float(n_bits) * len(fronteiras))
    # Aplica o AGS
    best, score, numero_avaliacoes_fo = algoritmo_genetico_simples(fo,
   fronteiras, n_bits, n_iter, n_pop, r_cross, r_mut)
    print('%f * %d' % (score, numero_avaliacoes_fo))
    return score
```

A.3 ED Simples – Algoritmo Equipe B

```
from numpy.random import rand, choice
from numpy import clip

# Funcoes relativas a EQUIPE B — ED Simples

# Define operador de mutacao
def mutationED(x, F):
    #x: vetores que serao usados para se aplicar a mutacao
    #F: fator de escala (deve ser positivo)

    #Aplica mutacao (3.17) e retorna o individuo mutado
    mutacao=np.array(x[0]) + F * (np.array(x[1]) - np.array(x[2]))
```

```
    return mutacao.tolist()

# Checa se o individuo esta dentro das fronteiras do problema
def checar_fronteiras1(mutated,fronteiras):
    #mutated: individuo mutado
    #fronteiras: limites inferior e superior de cada variavel

    mutated_bound = [clip(mutated[i],fronteiras[i][0],fronteiras[i][1])
   for i in range(len(fronteiras))]
    return mutated_bound

# Define operador de cruzamento binomial
def crossoverED(mutated, target, dims, cr):
    #mutated: vetor vindo da mutacao
    #target: vetor alvo
    #dims: numero de dimensoes do problema
    #cr: probabilidade de cruzamento (deve estar entre 0 e 1)

    # Gera um valor aleatorio de uma distribuicao uniforme para cada
   dimensao
    rs = rand(dims)

    # Gera e retorna o vetor tentativa via cruzamento binomial
    trial = [mutated[i] if rs[i] < cr else target[i] for i in range(dims
   )]
    return trial

# Implementacao do Algoritmo de ED Simples (ver Secao 3.5)
def evolucao_diferencial_simples(objective,fronteiras, n_iter, n_pop, cr
   , F):
    #objective: funcao objetivo a ser otimizada
    #fronteiras: limites inferior e superior de cada variavel
```

```
#n_iter: numero de geracoes (deve ser inteiro positivo)
#n_pop: tamanho da populacao (deve ser inteiro positivo)
#cr: probabilidade de cruzamento (deve estar entre 0 e 1
#F: fator de escala (deve ser positivo)

#Gera populacao inicial e a avalia
pop = []
scores = np.zeros(n_pop)
numero_avaliacoes_fo = 0
for c_pop in range(n_pop):
    individuo=[]
    for c_dim in range(len(fronteiras)):
        individuo.append(fronteiras[c_dim][0] + (rand() * (
fronteiras[c_dim][1] -fronteiras[c_dim][0])))
    pop.append(individuo)
    # Avalia vetor produzido, monta vetor de scores e contabiliza
numero de avaliacoes
    avaliacao = objective(individuo,numero_avaliacoes_fo)
    scores[c_pop] = avaliacao.y
    numero_avaliacoes_fo = avaliacao.cont
# Define melhor individuo como vazio e melhor valor de funcao
objetivo como infinito
best = []
best_eval = math.inf

#Aplica ED ao longo das geracoes (iteracoes)
for i in range(int(n_iter-1)):
    for j in range(n_pop):
        candidates = [candidate for candidate in range(n_pop) if
candidate != j]
        escolhidos = choice(candidates, 3, replace=False).tolist()
        v1=pop[escolhidos[0]]
```

```
            v2=pop[escolhidos[1]]
            v3=pop[escolhidos[2]]
            # Aplica mutacao
            mutated = mutationED([v1, v2, v3], F)
            # Checa se o individuo gerado por mutacao esta dentro das
    fronteiras
            mutated = checar_fronteiras1(mutated,fronteiras)
            # Aplica cruzamento
            trial = crossoverED(mutated, pop[j], len(fronteiras), cr)
            # Busca funcao objetivo para vetor alvo
            obj_target = scores[j]
            # Calcula funcao objetivo para vetor de teste
            avaliacao_trial = objective(trial,numero_avaliacoes_fo)
            obj_trial = avaliacao_trial.y
            numero_avaliacoes_fo = avaliacao_trial.cont
            # Aplica a Selecao
            if obj_trial < obj_target:
                # Troca o vetor alvo pelo vetor de teste
                pop[j] = trial
                scores[j] = obj_trial
                if obj_trial < best_eval:
                    best, best_eval = pop[j], obj_trial
    return [best, best_eval, numero_avaliacoes_fo]

def equipeB(fo,n_iter,n_pop):
    #fo: funcao objetivo
    #n_iter: numero de geracoes (deve ser inteiro positivo)
    #n_pop: tamanho da populacao (deve ser inteiro positivo)

    # Definicao de parametros
    fronteiras = fo([],0).fronteiras
    cr=0.9
```

```
    F=0.5

    #Aplicacao de ED
    best, score, numero_avaliacoes_fo = evolucao_diferencial_simples(fo,
   fronteiras, n_iter, n_pop, cr, F)
    print('%f * %d' % (score, numero_avaliacoes_fo))
    return score
```

A.4 AE híbrido com elementos inspirados em AG e ED – Algoritmo Equipe C

```
from numpy.random import rand, choice
from numpy import clip
import copy

# Funcoes relativas a EQUIPE C — AE com elementos de AG e ED

# Checa se individuo esta dentro das fronteiras que limitam a regiao de
   busca
def checar_fronteiras(individuo, fronteiras):
    #individuo: individuo que sera verificado se esta dentro da regiao
   factivel
    #fronteiras: valores superior e inferior para cada variavel

    #Verifica se o individuo esta dentro da regiao de busca
    individuo_dentro = individuo
    for i in range(len(fronteiras)):
        if individuo[i] < fronteiras[i][0]:
           individuo_dentro[i] = individuo[i] % fronteiras[i][0]
        elif individuo[i] > fronteiras[i][1]:
            individuo_dentro[i] = individuo[i] % fronteiras[i][1]
    return individuo_dentro.tolist()
```

```
# Define operador de mutacao baseado em ED/rand/1
def mutacao_rand_1(candidatos,avaliacoes,F,fronteiras):
    # Mutacao ED/best/1
    # candidatos: populacao de candidatos dos quais serao selecionados
   os 3 vetores em que se aplica a mutacao
    # avaliacoes: avaliacoes das funcoes objetivo organizada de acordo
   com a posicao dos candidatos
    # F: fator de escala (deve ser inteiro positivo)
    # fronteiras: limites inferior e superior de cada variavel

    #Escolhe 3 individuos aleatoriamente dentre os candidatos
    indices_escolhidos = choice(len(candidatos), 3, replace=False).
   tolist()

    #Aplica mutacao ED/best/1 (3.17)
    if avaliacoes[indices_escolhidos[0]] <= avaliacoes[
   indices_escolhidos[1]] and avaliacoes[indices_escolhidos[0]] <=
   avaliacoes[indices_escolhidos[2]]:
        v1 = candidatos[indices_escolhidos[0]]
        v2 = candidatos[indices_escolhidos[1]]
        v3 = candidatos[indices_escolhidos[2]]
    elif avaliacoes[indices_escolhidos[1]] <= avaliacoes[
   indices_escolhidos[0]] and avaliacoes[indices_escolhidos[1]] <=
   avaliacoes[indices_escolhidos[2]]:
        v1 = candidatos[indices_escolhidos[1]]
        v2 = candidatos[indices_escolhidos[0]]
        v3 = candidatos[indices_escolhidos[2]]
    else:
        v1 = candidatos[indices_escolhidos[2]]
        v2 = candidatos[indices_escolhidos[0]]
        v3 = candidatos[indices_escolhidos[1]]
```

```
    # aplica a mutacao
    mutacao=np.array(v1) + F * (np.array(v2) - np.array(v3))
    mutacao = checar_fronteiras(mutacao, fronteiras)

    return mutacao

# Selecao dos Pais por Torneio Probabilistico
def torneio_probabilidade(pop, avaliacoes, n_cruz, prob_torneio):
    #pop: populacao
    #avaliacoes: avaliacoes da populacao
    #n_cruz: numero de cruzamentos que serao executados (deve ser
   inteiro positivo)
    #prob_torneio: probabilidade do torneio ocorrer de forma
   deterministica (deve ser um valor entre 0 e 1)

    pais1 = []
    pais2 = []
    # Define os pais selecionados para cada cruzamento
    for i in range(n_cruz):
        # Seleciona 4 candidatos dentro da populacao
        indices_pais= np.random.choice(range(len(pop)), 4, replace=False
   )
        indices_pais1 = indices_pais[0:2]
        indices_pais2 = indices_pais[2:5]

        # Sorteia um valor aleatoriamente entre 0 e 1. Caso este valor
   seja menor que a probabilidade de torneio, vence o torneio individuo
   mais bem adaptado. Caso contrario, vence o individuo pior avaliado.

        numero_torneio = np.random.uniform(0, 1)
        if numero_torneio < prob_torneio:
```

```
            if avaliacoes[indices_pais1[0]] < avaliacoes[indices_pais1
    [1]]:
                pais1.append(pop[indices_pais1[0]])
            else:
                pais1.append(pop[indices_pais1[1]])
            if avaliacoes[indices_pais2[0]] < avaliacoes[indices_pais2
    [1]]:
                pais2.append(pop[indices_pais2[0]])
            else:
                pais2.append(pop[indices_pais2[1]])
        else:
            if avaliacoes[indices_pais1[0]] < avaliacoes[indices_pais1
    [1]]:
                pais1.append(pop[indices_pais1[1]])
            else:
                pais1.append(pop[indices_pais1[0]])
            if avaliacoes[indices_pais2[0]] < avaliacoes[indices_pais2
    [1]]:
                pais2.append(pop[indices_pais2[1]])
            else:
                pais2.append(pop[indices_pais2[0]])

    # Retorna os pais selecionados
    return [pais1, pais2]

# Aplica o Cruzamento Aritmetico
def cruzamento_aritmetico(pais1, pais2, n_cruz):
    #pais 1, pais2: conjuntos de pais que serao submetidos ao cruzamento
    aritmetico (3.2)
    #n_cruz: numero de cruzamentos que serao realizados (deve ser
   inteiro positivo)
```

```
    filhos = []
    for i in range(n_cruz):
        pai1 = np.array(pais1[i])  # Converte para numpy array
        pai2 = np.array(pais2[i])  # Converte para numpy array
        alpha = np.random.uniform(0, 1)
        filho1 = alpha * pai1 + (1 - alpha) * pai2
        #filho2 = (1 - alpha) * pai1 + alpha * pai2
        filhos.append(filho1.tolist())  # Converte de volta para lista
        #filhos.append(filho2.tolist())  # Converte de volta para lista
    return filhos

# Define a forma de elitismo
def elitismoC(n_pop, individuos, avaliacoes, filhos, avaliacoes_filhos,
   mutantes, avaliacoes_mutantes, candidatos_otimos, avaliacoes_otimos):

    # Seleciona candidatos otimos para serem preservados na proxima
   geracao

    indice_otimo = np.argmin(avaliacoes_otimos)
    otimo = candidatos_otimos[indice_otimo]
    avaliacao_otimo = avaliacoes_otimos[indice_otimo]

    nova_pop, nova_avaliacoes_pop = [], []
    nova_pop.extend(candidatos_otimos)
    nova_avaliacoes_pop.extend(avaliacoes_otimos)

    individuos.extend(filhos)
    avaliacoes.extend(avaliacoes_filhos)
    individuos.extend(mutantes)
    avaliacoes.extend(avaliacoes_mutantes)
```

```
    indices = np.random.choice(range(len(individuos)), n_pop-3, replace
   =False)
    nova_pop.extend([individuos[i] for i in indices])
    nova_avaliacoes_pop.extend([avaliacoes[i] for i in indices])

    return [nova_pop, nova_avaliacoes_pop, otimo, avaliacao_otimo]

# Algoritmo Evolutivo C
def evolutivoC(f_objetivo, fronteiras, n_iter, n_pop, prob_cross,
   prob_torneio, prob_mut, F):
    n_cruz = round(n_pop*prob_cross)
    n_mut = round(n_pop*prob_mut)
    pop, avaliacoes, avaliacao_ind, avaliacao_filho, avaliacao_mut,
   avaliacoes_filhos, avaliacoes_mutantes  = [],[],[],[],[],[],[]
    numero_avaliacoes_fo = 0

    otimo_individuo = []
    avaliacao_otimo_individuo = math.inf

    for c_pop in range(n_pop):
        individuo=[]
        for c_dim in range(len(fronteiras)):
            individuo.append(fronteiras[c_dim][0] + (rand() * (
   fronteiras[c_dim][1] - fronteiras[c_dim][0])))
        pop.append(individuo)
        avaliacao_ind = f_objetivo(individuo,numero_avaliacoes_fo)
        avaliacoes.append(avaliacao_ind.y)
        numero_avaliacoes_fo = avaliacao_ind.cont
        if avaliacao_ind.y <= avaliacao_otimo_individuo:
            otimo_individuo = individuo
            avaliacao_otimo_individuo = avaliacao_ind.y
```

```
 for i in range(n_iter-1):

     # Seleciona otimo pai para ser preservado para proxima geracao
     candidatos_otimos,avaliacoes_otimos = [],[]
     candidatos_otimos.append(otimo_individuo)
     avaliacoes_otimos.append(avaliacao_otimo_individuo)

     # Seleciona os pais para cruzamento via torneio
     [pais1, pais2] = torneio_probabilidade(pop,avaliacoes,n_cruz,
prob_torneio)
     # Obtem os filhos via cruzamento aritmetico
     filhos = cruzamento_aritmetico(pais1,pais2,n_cruz)
     # Avalia os filhos
     avaliacao_otimo_filho = math.inf
     otimo_filho = []
     for d in range(n_cruz):
         avaliacao_filho = f_objetivo(filhos[d],numero_avaliacoes_fo)
         avaliacoes_filhos.append(avaliacao_filho.y)
         numero_avaliacoes_fo = avaliacao_filho.cont
         if avaliacao_filho.y <= avaliacao_otimo_filho:
             otimo_filho = filhos[d]
             avaliacao_otimo_filho = avaliacao_filho.y

     # Seleciona otimo filho para ser preservado para proxima geracao
     if otimo_filho != []:
         candidatos_otimos.append(otimo_filho)
         avaliacoes_otimos.append(avaliacao_otimo_filho)

     candidatos_mut = copy.deepcopy(pop)
     candidatos_mut.extend(filhos)
     av_candidatos_mut = copy.deepcopy(avaliacoes)
     av_candidatos_mut.extend(avaliacoes_filhos)
```

```
        mutantes, otimo_mut, avaliacao_otimo_mut = [], [], []
        avaliacao_otimo_mut = math.inf
        # Aplica a mutacao baseada em ED aos filhos
        for j in range(n_mut):
            mutante = mutacao_rand_1(candidatos_mut,av_candidatos_mut,F,
    fronteiras)
            mutantes.append(mutante)
            avaliacao_mut = f_objetivo(mutante,numero_avaliacoes_fo)
            avaliacoes_mutantes.append(avaliacao_mut.y)
            numero_avaliacoes_fo = avaliacao_mut.cont
            if avaliacao_mut.y <= avaliacao_otimo_mut:
                otimo_mut = mutante
                avaliacao_otimo_mut = avaliacao_mut.y

        # Seleciona otimo mutante para ser preservado para proxima
    geracao
        candidatos_otimos.append(otimo_mut)
        avaliacoes_otimos.append(avaliacao_otimo_mut)

        # Aplica criterio de elitismo
        pop,avaliacoes,otimo_individuo,avaliacao_otimo_individuo =
    elitismoC(n_pop,pop,avaliacoes,filhos,avaliacoes_filhos,mutantes,
    avaliacoes_mutantes,candidatos_otimos,avaliacoes_otimos)

     return [otimo_individuo, avaliacao_otimo_individuo,
    numero_avaliacoes_fo]

def equipeC(fo,n_iter,n_pop):
     fronteiras = fo([],0).fronteiras
     F=0.5
     prob_cross=0.5
     prob_mut=0.5
```

```
    prob_torneio=0.5
    otimo_individuo, avaliacao_otimo_individuo, numero_avaliacoes_fo =
   evolutivoC(fo, fronteiras, n_iter, n_pop, prob_cross, prob_torneio,
   prob_mut, F)
    print('%f * %d' % (avaliacao_otimo_individuo,numero_avaliacoes_fo))
    return avaliacao_otimo_individuo
```

A.5 AE híbrido com elementos inspirados em AG e ED – Algoritmo Equipe D

```
from numpy.random import rand, choice
from numpy import clip
import copy

# Funcoes relativas a EQUIPE D — AE com elementos de AG e ED

# Ajusta parametros
def ajusta_parametrosD(iter_atual,n_iter,n_pop,fronteiras):
    if iter_atual/n_iter <= 0.5:
        probM = 0.5
        taxa_sel = 0.4
        c = 0.3
    elif iter_atual/n_iter >= 0.85:
        probM = 0.95
        taxa_sel = 0.95
        c = 0.1
    else:
        probM = 0.8
        taxa_sel = 0.6
        c = 0.2

    n_sel = int(taxa_sel*n_pop)
```

```
    n_mut1 = int(n_pop*probM)
    n_mut2 = n_pop - n_mut1
    var = (c*abs(np.diff(fronteiras))/2)**2
    cov = np.diagflat(var)
    return n_mut1,n_mut2,cov,n_sel,c

# Define operador de mutacao baseado em ED
def mutacao1D(atual,otimo_individuo,candidatos_cp,avaliacoes,F,
   fronteiras):

    # aplica a mutacao
    indices_escolhidos = choice(len(candidatos_cp), 4, replace=False).
   tolist()
    v1 = atual
    v2 = otimo_individuo
    v3 = candidatos_cp[indices_escolhidos[0]]
    v4 = candidatos_cp[indices_escolhidos[1]]
    v5 = candidatos_cp[indices_escolhidos[2]]
    v6 = candidatos_cp[indices_escolhidos[3]]
    mutacao = np.array(v1) + F * (np.array(v2) - np.array(v1) + np.
   array(v3) - np.array(v4) + np.array(v5) - np.array(v6))

    mutacao = checar_fronteiras(mutacao, fronteiras)

    return mutacao

# Define operador de mutacao
def mutacao2D(candidato,cov,fronteiras):
    mean = np.zeros(len(fronteiras))
    s = np.random.multivariate_normal(mean, cov)
    mutacao1 = candidato + s
    mutacao1 = checar_fronteiras(mutacao1, fronteiras)
```

```
    return mutacao1

# Define a forma de elitismo
def elitismoD(n_sel,n_pop,individuos,avaliacoes,mutantesX,
   avaliacoes_mutantesX,mutantesY,avaliacoes_mutantesY,candidatos_otimos
   ,avaliacoes_otimos):

    indice_otimo = np.argmin(avaliacoes_otimos)
    otimo = candidatos_otimos[indice_otimo]
    avaliacao_otimo = avaliacoes_otimos[indice_otimo]

    individuos.remove(candidatos_otimos[0])
    avaliacoes.remove(avaliacoes_otimos[0])

    mutantesX.remove(candidatos_otimos[1])
    avaliacoes_mutantesX.remove(avaliacoes_otimos[1])



    individuos.extend(mutantesX)
    avaliacoes.extend(avaliacoes_mutantesX)

    individuos_copia = copy.deepcopy(individuos)
    avaliacoes_copia = copy.deepcopy(avaliacoes)

    nova_pop, nova_avaliacoes_pop, indices_nova_pop = [], [],[]
    avaliacoes_ord = sorted(avaliacoes_copia)

    for j in range(n_sel):
        index_elemento = avaliacoes_copia.index(avaliacoes_ord[j])
        nova_pop.append(individuos_copia[index_elemento])
        nova_avaliacoes_pop.append(avaliacoes_copia[index_elemento])
        if individuos_copia[index_elemento] in individuos:
```

```
            individuos.remove(individuos_copia[index_elemento])
            avaliacoes.remove(avaliacoes_copia[index_elemento])

    if n_pop-n_sel-3 < 1:
        n_rand = 1
    else:
        n_rand = n_pop-n_sel-3

    indices_nova_pop = np.random.choice(range(len(individuos)), n_rand,
   replace=False)
    nova_pop.extend([individuos[i] for i in indices_nova_pop])
    nova_avaliacoes_pop.extend([avaliacoes[i] for i in indices_nova_pop
   ])

    nova_pop.extend(candidatos_otimos)
    nova_avaliacoes_pop.extend(avaliacoes_otimos)

    return [nova_pop, nova_avaliacoes_pop, otimo, avaliacao_otimo]

# Algoritmo Evolutivo D
def evolutivoD(f_objetivo,fronteiras,n_iter,n_pop):
    # Gera a populacao inicial
    pop, avaliacoes = [],[]
    numero_avaliacoes_fo = 0
    avaliacao_otimo_individuo = math.inf
    for c_pop in range(n_pop):
        individuo=[]
        for c_dim in range(len(fronteiras)):
            individuo.append(fronteiras[c_dim][0] + (rand() * (
   fronteiras[c_dim][1] - fronteiras[c_dim][0])))
        pop.append(individuo)
        avaliacao_ind = f_objetivo(individuo,numero_avaliacoes_fo)
```

```
        avaliacoes.append(avaliacao_ind.y)
        numero_avaliacoes_fo = avaliacao_ind.cont
        # Encontra o otimo da populacao inicial
        if avaliacao_ind.y <= avaliacao_otimo_individuo:
            otimo_individuo = individuo
            avaliacao_otimo_individuo = avaliacao_ind.y

    for i in range(n_iter-1):
        n_mut1,n_mut2,cov,n_sel,F = ajusta_parametrosD(i,n_iter,n_pop,
fronteiras)
        pop_antiga = []
        pop_antiga = copy.deepcopy(pop)
        mutantesX, mutantesY, avaliacoes_mutantesX, avaliacoes_mutantesY
 = [], [], [], []
        avaliacao_otimo_mutanteX = math.inf

        # Produz a primeira mutacao - ED/current-to-best/2
        indices = choice(len(pop), n_mut1, replace=False).tolist()
        for j in range(n_mut1):
            mutante1 = mutacao1D(pop[indices[j]],otimo_individuo,pop,
avaliacoes,F,fronteiras)
            # Avalia o individuo mutante 1
            av_mutante1 = f_objetivo(mutante1,numero_avaliacoes_fo)
            numero_avaliacoes_fo = av_mutante1.cont
            mutantesX.append(mutante1)
            avaliacoes_mutantesX.append(av_mutante1.y)
            # Procura pelo individuo mutante 1 otimo
            if av_mutante1.y <= avaliacao_otimo_mutanteX:
                otimo_mutanteX = mutante1
                avaliacao_otimo_mutanteX = av_mutante1.y
```

```
        indices = choice(len(pop_antiga), n_mut2, replace=False).tolist
    ()
        avaliacao_otimo_mutanteY = math.inf

        # Produz a segunda mutacao - perturbacao considerando uma
    distribuicao normal multivariada
        for j in range(n_mut2):
            mutante2 = mutacao2D(pop_antiga[indices[j]],cov,fronteiras)
            # Avalia o individuo mutante 2
            av_mutante2 = f_objetivo(mutante2,numero_avaliacoes_fo)
            numero_avaliacoes_fo = av_mutante2.cont
            mutantesY.append(mutante2)
            avaliacoes_mutantesY.append(av_mutante2.y)
            # Procura pelo individuo mutante 2 otimo
            if av_mutante2.y <= avaliacao_otimo_mutanteY:
                otimo_mutanteY = mutante2
                avaliacao_otimo_mutanteY = av_mutante2.y

        # Separa os otimos: geracao anterior, mutante 1 e mutante 2
        candidatos_otimos = []
        candidatos_otimos.append(otimo_individuo)
        candidatos_otimos.append(otimo_mutanteX)
        candidatos_otimos.append(otimo_mutanteY)
        avaliacoes_otimos = []
        avaliacoes_otimos.append(avaliacao_otimo_individuo)
        avaliacoes_otimos.append(avaliacao_otimo_mutanteX)
        avaliacoes_otimos.append(avaliacao_otimo_mutanteY)

        #Aplica o elitismo
        pop,avaliacoes,otimo_individuo,avaliacao_otimo_individuo =
    elitismoD(n_sel,n_pop,pop,avaliacoes,mutantesX,avaliacoes_mutantesX,
    mutantesY,avaliacoes_mutantesY,candidatos_otimos,avaliacoes_otimos)
```

```
    return [otimo_individuo, avaliacao_otimo_individuo,
   numero_avaliacoes_fo]

def equipeD(fo,n_iter,n_pop):
    fronteiras = fo([],0).fronteiras
    otimo_individuo, avaliacao_otimo_individuo, numero_avaliacoes_fo =
   evolutivoD(fo,fronteiras,n_iter,n_pop)
    print('%f * %d' % (avaliacao_otimo_individuo, numero_avaliacoes_fo))
    return avaliacao_otimo_individuo
```

A.6 Código para Batelada de Testes

```
from numpy.random import randint
from numpy.random import rand

# Gerar Resultados das Simulacoes
NExec = 30
EA = np.zeros((NExec,10))
EB = np.zeros((NExec,10))
EC = np.zeros((NExec,10))
ED = np.zeros((NExec,10))
EE = np.zeros((NExec,10))

ra, rb, rc, rd = [], [], [], []
fmin = [-1.8996, -39.16617, 0, 0, 0, -1.9133, -78.33233, 0, 0, 0]

n_iter = [4, 4, 4, 30, 30, 10, 10, 50, 200, 200]
n_pop = [8, 8, 8, 30, 30, 10, 10, 30, 200, 200]

for cfo in range(1, 11):
    print('Funcao Objetivo', cfo)
```

```
    print('\n')

    fo = globals()['f{}'.format(cfo)]

    for equipe in ['A','B','C','D']:
        print('Equipe', equipe, '\n')
        equipe_results = globals()['E' + equipe]  # Seleciona a matriz
   de resultados correta

        for ce in range(NExec):
            np.random.seed(ce)
            r = globals()['equipe' + equipe](fo, n_iter[cfo-1], n_pop[
   cfo-1])
            equipe_results[ce, cfo - 1] = abs(r - fmin[cfo - 1])
            r = []

        print('\n')

# Gerar Notas
def calculaS(E, NExec):
    S = 0.2*(sum(sum(E[:,0:3])))/(3*NExec)+0.3*(sum(sum(E[:,4:8])))/(4*
   NExec)+0.5*(sum(sum(E[:,8:11])))/(3*NExec)
    return S

def calculaN(vetorS):
    n_equipes = len(vetorS)
    N = np.zeros(n_equipes)
    Smin = min(vetorS)
    for j in range(n_equipes):
        N[j] = (1 - (vetorS[j] - Smin) / vetorS[j]) * 50
    return N
```

```
def calculaP(EA, EB, EC, ED):
    n_lin = np.size(EA, 0)
    n_col = np.size(EA, 1)

    PA = np.zeros((n_lin, n_col))
    PB = np.zeros((n_lin, n_col))
    PC = np.zeros((n_lin, n_col))
    PD = np.zeros((n_lin, n_col))

    for i in range(n_lin):
        for j in range(n_col):
            lista = sorted([EA[i, j], EB[i, j], EC[i, j], ED[i, j]])
            PA[i, j] = lista.index(EA[i, j]) + 1
            PB[i, j] = lista.index(EB[i, j]) + 1
            PC[i, j] = lista.index(EC[i, j]) + 1
            PD[i, j] = lista.index(ED[i, j]) + 1

    return PA, PB, PC, PD

SEA = calculaS(EA,NExec)
SEB = calculaS(EB,NExec)
SEC = calculaS(EC,NExec)
SED = calculaS(ED,NExec)

N1 = calculaN([SEA, SEB, SEC, SED])
N1A = N1[0]
N1B = N1[1]
N1C = N1[2]
N1D = N1[3]

PA, PB, PC, PD = calculaP(EA, EB, EC, ED)
SPA = calculaS(PA,NExec)
```

```
SPB = calculaS(PB,NExec)
SPC = calculaS(PC,NExec)
SPD = calculaS(PD,NExec)

N2 = calculaN([SPA, SPB, SPC, SPD])
N2A = N2[0]
N2B = N2[1]
N2C = N2[2]
N2D = N2[3]

NA = N1A + N2A
NB = N1B + N2B
NC = N1C + N2C
ND = N1D + N2D
```