UNIVERSIDADE FEDERAL DO PARANÁ

EDUARDO HENRIQUE SILVEIRA DA COSTA

ALGORITMO GENÉTICO APLICADO A PROBLEMAS NA ECONOMIA

CURITIBA
2022

EDUARDO HENRIQUE SILVEIRA DA COSTA

ALGORITMO GENÉTICO APLICADO A PROBLEMAS NA ECONOMIA

Monografia apresentada como requisito à obtenção do grau de Bacharel em Ciências Econômicas pelo Curso de Ciências Econômicas, Setor de Ciências Sociais Aplicada, Universidade Federal do Paraná.

Orientador: Dr. João Basilio Pereima

CURITIBA
2022

TERMO DE APROVAÇÃO

EDUARDO HENRIQUE SILVEIRA DA COSTA

ALGORITMO GENÉTICO APLICADO A PROBLEMAS NA ECONOMIA

Monografia apresentada como requisito à obtenção do grau de Bacharel em Ciências Econômicas pelo Curso de Ciências Econômicas, Setor de Ciências Sociais Aplicada, Universidade Federal do Paraná:

Orientador: Prof. Dr. João Basilio Pereima
Setor de Ciências Sociais Aplicadas
Universidade Federal do Paraná

---

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Setor de Ciências Sociais Aplicadas
Universidade Federal do Paraná

---

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Setor de Ciências Sociais Aplicadas
Universidade Federal do Paraná

Curitiba, xx de Abril de 2022.

AGRADECIMENTOS

XXXXXXX

RESUMO

Em economia, embora haja algumas teorias estabelecidas, não há um consenso de como se analisar certos comportamentos dos agentes econômicos. Algumas linhas de pensamento se destacaram e se tornaram a principal forma de análise e previsão de conjecturas econômicas ao redor do mundo em diferentes momentos do século XX e XXI. A partir do desenvolvimento computacional na metade do século passado, ambas as áreas se tornaram intrinsecamente ligadas, e os modelos econômicos cada vez testados em sistemas computacionais aproveitando o poder de processamento dos computadores para a realização de cálculos. Dessa forma, a presente monografia objetiva apresentar os algoritmos genéticos como alternativa para análise e previsão de cenários econômicos, especialmente, no que diz respeito a problemas de otimização, inovação e aprendizado adaptativo. Para isso, realizou-se uma breve introdução à história do desenvolvimento dos algoritmos genéticos, assim como um aprofundamento em seus componentes para a construção de um algoritmo genético simples e alguns exemplos de aplicações em problemas econômicos.

Palavras-chave: algoritmo genético; otimização; inovação; aprendizado adaptativo.

ABSTRACT

In economics, although there are some established theories, there is no consensus on how to analyze certain behaviors of economic agents. Some lines of thought stood out and became the main way of analyzing and forecasting economic conjectures around the world at different times in the 20th and 21st. From the computational development in the middle of the last century, both areas became intrinsically linked, and the economic models were tested in computational systems taking advantage of the processing power of computers to perform calculations. Thus, the present monograph aims to present genetic algorithms as an alternative for the analysis and prediction of economic scenarios, especially regarding problems of optimization, innovation, and adaptive learning. For this, a brief introduction to the history of the development of genetic algorithms was carried out, as well as a deepening of its components for the construction of a simple genetic algorithm and some examples of applications in economic problems.

Keywords: genetic algorithm; optimization; inovation; adaptative learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AG - Algoritmo Genético
CE - Computação Evolucionária
PE - Programação Evolucionária
EE - Estratégias Evolutivas
PIB - Produto Interno Bruto

SUMÁRIO

Capítulo 1

Instrodução

Capítulo 2

Computação Evolucionária

O contexto da criação e desenvolvimento dos algorítimos genéticos (AG) está intrinsecamente ligado à área de estudos da computação evolucionária que, por sua vez, está sob o guarda-chuva da área de pesquisa da inteligência artificial. Com isso, neste capítulo, procura-se realizar uma contextualização histórica à computação evolucionária, à programação evolucionária, às estratégias evolutivas e aos algoritmos genéticos, objetivando, assim, um maior entendimento de como o desenvolvimento das pesquisas nestas áreas contribuíram para o campo dos algoritmos genéticos.

2.1 Introdução

Como o próprio nome sugere, a computação evolucionária (CE) tem como inspiração os processos de evolução observados nos organismos da natureza, sendo uma metáfora computacional (vide Tabela 2.1) que, de forma geral, visa solucionar problemas computacionais ou entender melhor os processos naturais de evolução. Através de uma simulação destes processos naturais, busca-se pelos indivíduos mais aptos a sobreviverem em um ambiente, assim como analisar como os processos de reprodução e mutação destes indivíduos ocorreram. O próprio ambiente é, em si, um dos elementos mais importantes neste conjunto, tendo grande influência nessa luta pela sobrevivência e a busca de parceiros para reprodução e determinando como a capacidade de se adaptar a esse meio influenciará em suas chances de passar seus genes para as próximas gerações.

TABELA 2.1: METÁFORA DA COMPUTAÇÃO EVOLUCIONÁRIA

Evolution		Problem solving
Environment	$\Longleftrightarrow$	Problem
Individual	$\Longleftrightarrow$	Candidate solution
Fitness	$\Longleftrightarrow$	Quality

FONTE: Eiben and Smith (2015)

Ao fim de 1950, e meados da década de 60, a tecnologia havia avançado até chegar à computação digital, o que possibilitou um avanço na experimentação de novos modelos

de processos evolucionários e um grande número de estudos nas décadas seguintes. Os trabalhos de Friedberg (1958), Friedberg et al. (1959) e Bremermann (1962) são apontados como os primeiros registros de desenvolvimento de processos evolucionários aplicados no contexto de problemas computacionais. Os trabalhos de Friedberg podem ser considerados alguns dos primeiros estudos em *machine learning*[1] e programação automática (Back et al., 1997).

Bremermann (1962), publica sua pesquisa de evolução simulada aplicada à otimização linear e convexa e equações simultâneas não lineares, assim como desenvolve, em 1965[2], um dos primeiros estudos teóricos sobre algoritmos evolucionários, demonstrando que a mutação ótima deve ter um valor $\frac{1}{l}$[3] no caso de l bits codificados como indivíduos quando aplicados a problemas linearmente separáveis (Back et al., 1997).

Com as contribuições dos trabalhados acima apresentados, as pesquisas realizadas na segunda metade dos anos 1960 estabeleceram os três principais campos de estudo em CE, sendo eles a programação evolucionária (PE), as estratégias evolutivas (EE) e os algoritmos genéticos (AG). Na segunda metade da década de 1960, Lawrance Fogel[4] construía as bases da programação evolucionária em San Diego, Califórnia, e John Holland fundava as bases dos algoritmos genéticos na Universidade de Michigan[5]. Por sua vez, as estratégias evolutivas eram desenvolvidas por Inge Rechenberg, Peter Bienert e Hans-Paul Schwefel em Berlim em meados de 1965[6].

Como aponta Back et al. (1997), mesmo com cada uma das áreas seguindo seu próprio caminho de pesquisas ao longo dos quase 30 anos seguintes, a década de 1990 marca o encontro destes campos através dos esforços de seus pesquisadores na organização de diversos congressos com o objetivo de compartilharem os conhecimentos até então absorvidos, culminando, no início dos anos 1990, no consenso do nome **computação evolucionária** (destacado pelo autor) como o nome dessa nova grande área de pesquisa. A partir destas reuniões, o crescimento do número de interessados e novos trabalhos foi naturalmente crescendo ao longo da década. Em 1993, é criado um periódico homônimo pelo Instituto de Tecnologia de Massachusetts[7] e, em 1994, uma das três conferências do Congresso Mundial de Inteligência Computacional organizado pelo Instituto de Engenheiros Eletricistas e Eletrônicos[8] (Back et al., 1997 apud Eiben et al., 1994) .

2.2 Programação Evolucionária

Desenvolvida por Lawrance Fogel na década de 1960, a programação evolucionária (PE) foi construída sobre diversos experimentos visando a previsão, sob algum critério arbitrário, de séries temporais não estacionárias através da evolução simulada dos es-

[1] Do inglês, aprendizado de máquina.
[2] Bremermann H J and S
[3] *length*, do inglês, comprimento
[4] Fogel et al. (1966)
[5] Holland (1962)
[6] Rechenberg et al. (1965)
[7] Evolutionary Computation (1993)
[8] *IEEE World Congress on Computaional Intelligence* (WCCI)

tados das máquinas dentro de um limite de estados predeterminados, ou seja, dado os estados passados, previa-se os estados da máquina resultantes deste processo. Fogel buscou seguir um caminho diferente de pesquisa em relação ao que os trabalhos em inteligência artificial se concentravam à época, uma simulação primitiva de redes neurais. Para Fogel, havia uma grande limitação dos modelos baseados na inteligência humana em relação aos processos de criaturas com desenvolvimento contínuo do intelecto, necessário para sobrevivência em um dado ambiente (evolução).

Segundo Back et al. (1997, pg.A2.3:3), Fogel apresenta as primeiras tentativas de "[...](i) usar a evolução simulada para realizar predições, (ii) incluir codificações de comprimento variável, (iii) usar representações que tomam a forma de uma sequência de instrução, (iv) incorporar uma população de soluções candidatas e (v) coevoluir programas evolutivos" e partindo da premissa que "a inteligência é baseada na adaptação do comportamento para atingir metas em uma variedade de ambientes" (tradução nossa)[9].

Devido ao contexto computacional, esses ambientes eram representados através de uma sequência de símbolos ou caracteres de um alfabeto finito arbitrário e o problema evolutivo definido como uma sequência de instruções, ou algoritmo, aplicadas sobre o conjunto de símbolos já observados. Com isso, ao inserir um conjunto de máquinas (população) em um dado ambiente, onde cada máquina possui um valor definido como o valor de entrada, esperava-se melhorar a performance de previsão do algoritmo, à medida que o valor de saída, ou o resultado, era comparado com o próximo valor de entrada. A qualidade desta previsão era, então, medida por uma função de recompensa que indicava o quanto cada máquina da população se adaptou ao ambiente.

Cada máquina pai pode criar um ou mais descendentes, onde cada descendente é criado pelo processo aleatório de alteração de estado, ou valor, do pai. Esse processo de mutação pode ocorrer sob uma certa distribuição de probabilidade ou ser definido no início da implementação do algoritmo. Ao fim de cada geração, a função de recompensa é aplicada sobre o descendente, assim como foi feito com seu pai, para avaliar o quão apto está em relação ao ambiente. As máquinas que fornecem o maior valor de recompensa permanecem no ambiente e se tornam pais das máquinas da geração seguinte. Este processo acontece sucessivas vezes até o símbolo que o símbolo que se deseja prever seja, efetivamente, previsto. A melhor máquina irá gerar essa previsão e esse novo símbolo é adicionado na população para ser avaliado no ambiente, reiniciando, assim, o processo.

Esse algoritmo foi aplicado com êxito em problemas de previsão, identificação e controle automático, simulação de coevolução de populações, experimentos de previsão de sequência, reconhecimento de padrões e em jogos. Na década de 1980, o algoritmo estendeu-se para novas aplicações, como no ordenamento de itens no problema do caixeiro viajante e em funções de otimização contínua, evoluindo, posteriormente, para implementações em planejamento de rotas, seleção ótima de subconjuntos e treinamento de redes neurais. No início da década de 1990, ocorre a primeira conferência anual de programação evolucionária, com exemplos de diversas aplicações de otimização na área de robótica, planejamento de caminhos e rotas, desenho e treinamento de redes neurais,

[9] *[...] (i) use simulated evolution to perform prediction, (ii) include variable-length encodings, (iii) use representations that take the form of a sequence of instructions, (iv) incorporate a population of candidate solutions, and (v) coevolve evolutionary programs [...] considered intelligence to be based on adapting behavior to meet goals in a range of environments.*

controle automática entre outros (Back et al., 1997 apud Eiben et al., 1994) .

2.3 Estratégias Evolutivas

No meio da década de 1960, Bienert, Rechenberg e Schwefel, três estudantes da Universidade Técnica de Berlim, estudavam modelos de otimização aplicados em problemas da área de aerotecnologia e tecnologia espacial. Uma das principais pesquisas que realizavam à época, era de um robô experimental que, em um túnel de vento, deveria realizar uma série de testes em uma estrutura tridimensional fina e flexível visando minimizar a resistência em relação ao ar. Os primeiros testes não obtiveram sucesso. Foi apenas no ano seguinte ao início dos testes que Rechenberg et al. (1965) decide utilizar um dado para decisões aleatórias elaborando, assim, a primeira versão de uma EE (Back et al., 1997, pg.A2.3:6) chamada, posteriormente, de $(1 + l)EE$.

Essa primeira versão consiste em uma sequência de instruções projetadas para otimização contínua bastante similar à busca aleatória, exceto por uma regra para a força da mutação conhecida como "regra do sucesso $\frac{1}{5}$" para ajuste do desvio padrão dessa mutação. Como a notação sugere, a estratégia de evolução $(1 + l)$ possui apenas um indivíduo pai que irá gerar apenas um indivíduo filho, onde ambos são confrontados e o indivíduo que representa a solução mais fraca, morre. Este indivíduo sobrevivente gera um novo filho e, assim, repetindo essa sequência diversas até se chegar a uma solução ótima. Sendo executado indivíduo a indivíduo, esse processo é computacionalmente custoso, apresentando uma convergência lenta para uma solução ótima, assim como tem a possibilidade de convergir para uma solução local.

Devido aos problemas de desempenho, os autores trabalharam em melhorias na estrutura do algoritmo, desenvolvendo uma nova versão chamada de EE multi-membro[10] com população maior que 1. Novas melhorias foram realizadas nessa nova versão, resultando em dois princípios principais: $(\mu+1)$ e $(\mu, 1)$. No primeiro, μ indivíduos produzem λ descendentes, gerando uma população temporária de $(\mu + \lambda)$ novos indivíduos, havendo a seleção de μ indivíduos para a geração seguinte. No segundo tipo, μ indivíduos produzem λ descendentes, com $\mu < \lambda$, onde a nova população de μ indivíduos possui apenas os indivíduos selecionados do conjunto de λ descendentes, limitando o tempo de vida de um indivíduo apenas a uma geração específica.

A partir da primeira versão, a comunidade de pesquisadores da área de EE realizaram novas aplicações nas décadas seguintes que não se reduziram somente ao objetivo da otimização de valores do mundo real, como a aplicação para otimização binária em estruturas de indivíduos multicelulares usando a ideia de sub populações, estratégias evolutivas para problemas com multi critérios, entre diversas outras aplicações seguindo a ideia principal de melhoria contínua dos indivíduos analisados (Back et al., 1997).

[10] *EE multimembered.*

2.4 Algoritmos Genéticos

Conforme aponta Back et al. (1997, pg.A2.3:4), as primeiras ideias que deram origem aos algoritmos genéticos datam do início da década de 1960 nos trabalhos de Holland (1962), que "estabeleceu uma agenda ampla e ambiciosa para compreender os princípios subjacentes dos sistemas adaptativos – sistemas que são capazes de automodificação em resposta às suas interações com os ambientes em que devem funcionar" (tradução nossa)[11].

Diferentemente dos estudos apresentados anteriormente de algoritmos aplicados em modelos de previsão ou otimização, Holland se debruçou sobre modelos evolutivos para entendimento de sistemas adaptativos robustos naturais e projeção de elementos adaptativos em um dado contexto. Para o autor, em sistemas adaptativos naturais, as características relativas à competição entre os agentes e de inovação ao longo destes processos naturais eram fundamentais para que os indivíduos se adaptassem ao ambiente e às mudanças imprevistas que este ambiente aplicava sobre esses indivíduos (Back et al., 1997).

Para (Goldberg, 1989, p. 1), eram dois os principais objetivos de John Holland e seus colegas no campo de pesquisa dos AGs, sendo eles: uma explicação bem estruturada e fundamentada dos processos de adaptação de sistemas naturais e a construção de programas computacionais de sistemas artificiais com a finalidade de incorporar importantes mecanismos destes sistemas, sendo o foco da pesquisa a robustez dos algoritmos, ou seja, o equilíbrio entre a eficiência e a eficácia necessária para a sobrevivência de possíveis soluções em muitos ambientes diferentes.

O grande deferencial da linha desenvolvida por Holland, foi a incorporação de diversos conceitos da genética que se demonstraram de alta eficiência e performance na resolução de problemas complexos utilizando poucos dados de entrada, assim como os processos de busca para encontrar soluções ótimas apresentavam inovações em relação à resolução de tais problemas e aprendizados dos elementos no ambiente ao longo dos processos evolutivos. Os elementos a serem evoluídos ao longo de um período eram representados como genomas (conjunto de todos os genes de um ser vivo) e os mecanismos de evolução como abstrações de operadores genéticos como a reprodução, cruzamento e mutação.

Em 1967, Holland realiza estudos em uma teoria geral de sistemas adaptativos, desenvolvendo no mesmo período a análise de esquemas de sistemas adaptativos e, em 1969, demonstra aplicações de alocação ótima utilizando o modelo de k-bandidos armados (Holland apud Back et al., 1997) . Cavicchio (1970) absorveu essas ideias como uma forma de busca adaptativa e as testou em problemas de busca complexa envolvendo soluções de subrotinas e reconhecimento de padrões, assim como alguns dos primeiros estudos sobre formas elitistas de seleção e adaptação de taxas de cruzamento e mutação. Hollstien (1971), por sua vez, apresentou ideias aprofundadas de seleção alternada e, através de experimentações em paisagens de adaptação bidimensionais, testou diversos

[11] *Holland set out a broad and ambitious agenda for understanding the underlying principles of adaptive systems—systems that are capable of self-modification in response to their interactions with the environments in which they must function.*

modelos de estratégias de reprodução com origem em técnicas utilizadas por criadores de animais. Em 1975, Holland agrupa as ideias desenvolvidas em seus trabalhos e publica o seu maior trabalho, o livro Adaptação em Sistemas Naturais e Artificiais (Holland, 1975). No mesmo ano, Jong and Alan (1975) publica experimentos demonstrando teórica e praticamente os efeitos no tamanho da população, cruzamento e mutação de AGs aplicados em uma população, seguindo as ideias de Holland.

O interesse pela área foi crescendo progressivamente nas décadas seguintes. Em 1976, pesquisadores da Universidade de Michigan, Universidade de Pittsburgh, entre outras, organizaram a primeira conferência de sistemas adaptativos, que ocorreu nos anos seguintes. Em 1979, Holland, De Jong e Sampson escalam o tamanho da conferência através de um financiamento para realizarem uma conferência interdisciplinar em sistemas adaptativos, que acabou sendo realizado em 1981 na Universidade de Michigan. Em 1985, na Universidade de Pittsburgh, ocorre a Conferência Internacional sobre Algoritmos Genéticos (ICGA) que, devido ao sucesso, passou a ser semestral nos anos seguintes. Em 1989, surge a Sociedade Internacional de Algoritmos Genéticos (ISGA), organização responsável pelo financiamento de conferências e atividades das áreas de pesquisas relacionadas aos AGs, tendo como uma de suas primeiras conquistas a criação de uma das principais conferências da comunidade, sobre os Fundamentos dos Algoritmos Genéticos (FOGA) (Back et al., 1997, pg.A2.3:5).

Capítulo 3

Algoritmos Genéticos

Neste capítulo, procura-se introduzir, aprofundar e discutir os principais componentes, e suas características, para a construção de um algoritmo genético simples.

3.1 Introdução aos Algoritmos Genéticos

Um algoritmo genético é uma meta-heurística[1] com finalidades variadas que, conforme citado por Mitchell (1998, pg.27), pode ser dividido em dois campos principais de aplicação: como uma técnica de busca de possíveis soluções de problemas tecnológicos e como um modelo computacional que objetiva simular sistemas naturais em busca de respostas como, por exemplo, um maior entendimento dos processos evolutivos e de seleção natural. No primeiro, a gama de aplicações é extensa, encontrando-se diversos trabalhos em problemas das ciências exatas e ciências sociais; em relação ao segundo, encontram-se diferentes empregos de algoritmos genéticos nas áreas das ciências biológicas. Neste trabalho, serão abordadas as aplicações relativas a problemas nas ciências sociais, em especial, nas ciências econômicas.

Relativo às ciências econômicas, dentre as diversas aplicações encontradas na literatura, buscar-se-á abordar três principais: a busca por soluções ótimas de problemas de otimização, a procura por padrões ou características que podem ser entendidas como inovações em um dado processo ou contexto e, por último, o aprendizado que pode ser extraído dos processos de um AG através da aplicação de seus operadores e interação entre os elementos analisados. No presente capítulo, com o objetivo de abordar cada operador e processo de um AG simples, utilizar-se-á um exemplo de aplicação que busca encontrar uma solução de um problema de otimização. As demais aplicações apresentadas acima serão aprofundadas no capítulo seguinte.

De forma geral, um AG funciona da seguinte forma: definido um problema, o algoritmo realiza uma busca por uma solução global em um espaço de possíveis soluções, onde, ao realizar essa busca, ele pode encontrar ou não uma solução ótima. Inicialmente, estas possíveis soluções são construídas de forma aleatória e combinam suas

[1] Heurística é uma técnica construída para encontrar, gerar ou selecionar uma solução para um problema específico dentro de um problema maior definido, sendo a meta-heurística, então, uma heurística de alto nível que busca por heurísticas para resolução de um dado problema principal.

características entre si, formando novas possíveis soluções. Tais características determinarão como e quais serão os atributos combinados ou ignorados neste processo, que é realizado de forma iterativa[2], onde cada iteração termina com novas soluções construídas através da troca de informações entre os elementos. Estas sucessivas iterações terminam quando o objetivo predefinido é alcançado ou o algoritmo não encontra nenhuma solução que satisfaz o problema. Um AG básico, ou seja, que contenha pelo menos a utilização dos operadores de reprodução, cruzamento e mutação, é de simples construção e parametrização. Embora simples, é capaz de procurar por soluções em um espaço de busca muito maior e um desempenho acima dos programas convencionais Holland (1992, pg.66) e, conforme cita Goldberg (1989, pg.2), podem ser divididos em três métodos de busca principais: baseado em cálculo, enumerativo e aleatório ou randômico.

O primeiro tipo, é uma heurística de busca local e é subdividido em duas classes: indireto e direto. Através da resolução de, normalmente, um conjunto de equações não lineares, as técnicas de busca indireta procuram por extremos locais resultantes de uma função objetivo igual a zero. Em outras palavras, conforme a direção definida pelo vetor gradiente, analisa-se se o ponto de aclive ou declive, que não possui mais nenhuma variação para qualquer direção, assume o valor de máximo ou mínimo com base nas funções calculadas. Em relação aos métodos diretos, com uma solução arbitrária inicial, são feitos repetidos incrementos nesta solução e, caso essa mudança apresente um resultado melhor, é feito um novo incremento, e assim sucessivamente, até não haver mais nenhuma melhoria, levando-se em consideração as restrições do problema.

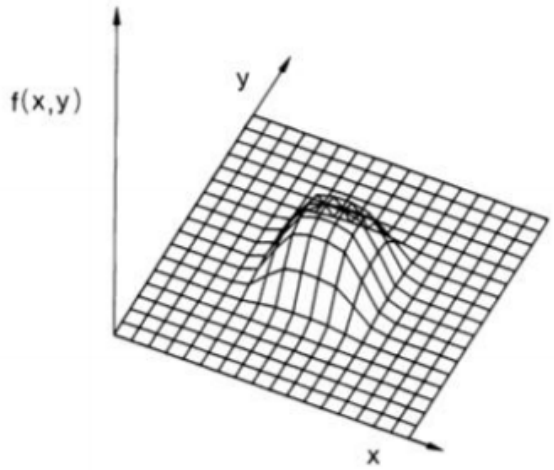
O segundo método, uma heurística de busca global, inicia buscando pelos valores da função objetivo em cada um dos pontos dentro de um espaço de busca delimitado ou um espaço de busca infinito discreto, parando ao encontrar um extremo global que se apresente como uma possível solução para o problema a ser resolvido.

Por último, o método de busca aleatória, também uma heurística de busca global, utiliza algum tipo de aleatoriedade ou probabilidade na busca por ótimos globais dentro de um espaço definido.

Goldberg (1989, pg.5) também cita que os métodos acima apresentados, com um exemplo de extremo fictício ilustrado na Figura 3.1, foram perdendo a relevância ao longo do tempo, pois são métodos de busca úteis em um número muito pequeno de situações, não apresentando uma eficácia e eficiência satisfatórias na resolução de um espectro grande de problemas, de diferentes níveis de complexidade, conforme a realidade demanda. Dessa forma, os algoritmos genéticos foram se destacando por sua robustez na resolução de um número considerável de problemas de otimização através da adaptação para os sistemas artificiais de alguns conceitos da biologia e da genética, sobretudo, o conceito darwiniano de evolução dos indivíduos mais aptos.

[2]Na programação, é uma ação que se repete sucessivamente até atingir um resultado desejado ou alguma ordem de término.

FIGURA 3.1: EXEMPLO DE UM EXTREMO LOCAL



FONTE: Goldberg (1989, p.3)

Dessa forma, nas seções subsequentes, serão explorados os principais conceitos, operadores e processos para a construção de um algoritmo genético simples.

3.2 Algoritmos e Linguagem de Máquina

Para Cormen et al. (2009, pg.2), um algoritmo é um "procedimento computacional bem definido que recebe um valor, ou um conjunto de valores, como entrada e produz algum valor, ou conjunto de valores, como saída"[3] (tradução nossa). Não é diferente com um AG, que, dada uma função de otimização, depende de um conjunto de parâmetros de entrada para encontrar um, ou mais de um, ponto ótimo, ou próximo ao ótimo, como valor, ou valores, de saída.

Sendo o desenho do AG uma simulação de um processo natural, é necessário que haja uma codificação dos valores de entrada para que o sistema computacional possa processá-los. Ou seja, é necessário realizar uma transformação das informações que os humanos interpretam, modificam e constroem, com base nos estudos dos processos naturais, em uma linguagem que o computador entenda, chamada linguagem ou código de máquina.

Como apresentado por Fedeli et al. (2009, pg.42), atualmente, os computadores utilizam apenas dois operadores básicos em sua linguagem, sendo eles os dígitos binários 0 e 1, também conhecidos como bit (do inglês, *binary digit*). O bit é a menor informação armazenada pela memória e processada pela unidade central de processamento (UCP) de um computador, onde, em um determinado espaço da memória, é armazenado um e somente um bit (0 ou 1) por vez. De forma mais ilustrativa, pode-se entender o 0

[3] [...] a well-defined computational procedure that takes some values, or set of values, as input and produces some value, or set of values, as output.

como uma instrução para um corte de energia ou uma informação relativa à negação, impedimento ou inexistência; do contrário, o 1 é uma instrução para passagem de energia ou uma informação relativa à positivação, desimpedimento ou existência

Existem várias formas de agrupamento destes dígitos, havendo interpretações e funções diferentes levando-se em consideração a quantidade e a ordem dos bits nestes agrupamentos, sendo o *American Standard Code for Information Interchange (ASCII)* o método de armazenamento e representação de caracteres mais utilizado pelas plataformas de computadores pessoais (Fedeli et al., 2009, pg.46), onde cada dígito é formado pela junção de 8 bits[4], unidade conhecida como byte (do inglês, *binary term*). Dessa forma, quando o dígito "A", por exemplo, é enviado para o computador, o ASCII o codifica, enviando a sequência 01000001 para armazenamento na memória. O mesmo processo acontece de forma inversa, quando o computador gera algum dígito que precisa ser representado graficamente.

Com isso, por questões relativas à facilidade de interpretação e desempenho computacional, os parâmetros de entrada de um AG são codificados em sequências de 0s e 1s, onde cada sequência tem origem de um determinado alfabeto [5], que permite realizar a codificação e decodificação dos valores de entrada e saída, respectivamente, do algoritmo aplicado.

3.3 Componentes de um Algoritmo Genético

A forma como os parâmetros de um AG são definidos impacta diretamente na robustez de seu funcionamento e das possíveis soluções encontradas. Dessa forma, serão explorados a seguir os principais elementos, e suas características, para a construção de um AG simples. Para isso, será utilizado como exemplo o Problema da Caixa Preta[6] apresentado por Goldberg (1989, p.8).

3.3.1 Alelo e Locus

Conforme abordado anteriormente (ver Seção 3.2), o primeiro passo na construção de um AG é a codificação dos valores de entrada em um conjunto de 0s e 1s. Cada valor, é uma característica binária ou um detector chamado de alelo (equivalente ao gene de um cromossomo na biologia). A posição de cada caractere dentro deste conjunto é chamada de *locus*. Pode-se analisar o locus à parte do gene (ou alelo). Por exemplo, supondo que as características (genes) do cabelo de um ser humano podem ser encontradas no cromossomo 2. Ao analisar os genes relativos ao cabelo dentro deste cromossomo, identifica-se no locus 3 (posição 3) que a cor do cabelo é preta (valor do alelo). Com isso, temos que o valor e a posição de cada elemento nesse conjunto apresentará uma

[4] Originalmente, o conjunto mínimo era de 7 bits Gorn et al. (1963).

[5] Aqui, alfabeto nada mais é que um conjunto finito de algorismos ou caracteres.

[6] Através de uma máquina ou dispositivo que possui um número específico de parâmetros de entrada (interruptores), o Problema da Caixa Preta consiste em obter o maior valor de saída possível com base na configuração destes parâmetros. O objetivo é analisar como um determinado sistema fechado relaciona os valores de entrada e a resposta, com base no estímulo destes valores, de saída.

característica específica, seja olhando para um único valor ou para um subconjunto de valores.

3.3.2 Cadeia de Caracteres

O conjunto codificado dos valores de entrada de um AG é chamado de cadeia de caracteres[7] ou vetor. Comparativamente, na biologia, o cromossomo é uma estrutura constituída por DNA (ácido desoxirribonucleico), sendo cada DNA composto por um número de genes que, por sua vez, são responsáveis por definirem a(s) característica(s) de um indivíduo. Dessa forma, temos que o vetor de um AG é um elemento artificial análogo ao cromossomo nos sistemas naturais. Como exemplo, o vetor com os possíveis valores da Caixa Preta pode ser representado da seguinte forma:

TABELA 3.1: CADEIA DE CARACTERES OU VETOR CONTENDO 5 GENES

Característica Binária (Gene)	Interruptor 1	Interruptor 2	Interruptor 3	Interruptor 4	Interruptor 5
Valor da Característica Binária (Alelo)	0	1	1	0	1
Posição ou Índice do Gene (Locus)	1	2	3	4	5
Referência Caixa Preta	Desligado	Ligado	Ligado	Desligado	Ligado

FONTE: adaptado de Goldberg (1989, p.11)

3.3.3 População Inicial e Gerações

Inicialmente, o AG começa a realizar sua busca sobre um conjunto aleatório de indivíduos (cadeias de caracteres ou vetores), chamado de população inicial. Através dessa busca, o algoritmo analisa cada um dos elementos da população, identificando quais são os mais aptos a sobreviver levando-se em consideração os demais indivíduos e o ambiente em que estão localizados. Esta estrutura composta por vários indivíduos com genes específicos se manterá por toda a vida do organismo (na genética, esta estrutura é chamada de genótipo) e, à medida que as gerações (iterações) passam, os indivíduos interagem entre si e com o ambiente criando, assim, um novo organismo. Na genética, a nova estrutura resultante deste processo é conhecida como fenótipo.

[7] Do inglês, *string*.

TABELA 3.2: POPULAÇÃO DE TAMANHO 4

Indivíduo (População)	Interruptor 1	Interruptor 2	Interruptor 3	Interruptor 4	Interruptor 5
Vetor 1	0	1	1	0	0
Vetor 2	1	1	0	0	0
Vetor 3	0	1	0	0	0
Vetor 4	1	0	0	1	1

FONTE: Elaborado pelo autor.

3.3.4 Paisagem de Aptidão, Sobrevivência do Mais Apto e Função Objetivo

O ambiente em que os indivíduos (vetores) estão localizados é um elemento importante que compõe os processos de um AG, sendo um dos responsáveis por direcionar a escolha de quais indivíduos irão passar para a próxima geração e quais irão morrer. Ou seja, o algoritmo irá analisar cada indivíduo num dado espaço procurando os mais aptas a sobreviverem ao ambiente em que estão localizados com base em suas características. Este espaço de busca é conhecido como paisagem ou horizonte de aptidão[8], definido por Langdon and Poli da seguinte maneira:

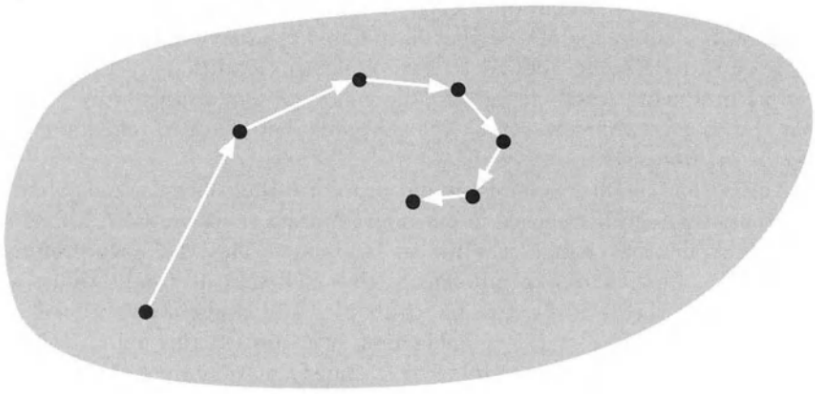
> Na forma mais simples, uma paisagem de aptidão pode ser vista como um gráfico onde cada ponto na direção horizontal representa todos os genes em um indivíduo (genótipo) correspondente àquele ponto. A aptidão daquele indivíduo é plotada como a altura. Se os genótipos podem ser visualizados em duas dimensões, o gráfico pode ser visto como um mapa de três dimensões, que pode conter montes e vales. Grandes regiões de baixa aptidão podem ser consideradas pântanos, enquanto grandes regiões de alta aptidão, que se mantêm num mesmo nível, podem ser consideradas como platôs. (Langdon and Poli, 2002, pg.4) (tradução nossa). [9]

Como ilustrado na Figura 3.2, um AG possui um espaço de busca predeterminado (área cinza) contendo a população inicial a ser analisada, representada na imagem pelo ponto preto mais inferior, à esquerda. Com base nas características binárias (genes) da população ou estrutura (genótipo), o AG determinará quais indivíduos passarão para a próxima geração (seta branca), formando uma nova população. Esse processo acontece sucessivamente até que, com base no estado (fenótipo) de cada nova população, seja encontrada a que possui as maiores chances de sobrevivência, sendo a possível solução para um dado problema de otimização.

[8] do inglês, *fitness landscape*

[9] In its simplest form a fitness landscape can be seen as a plot where each point in the horizontal direction represents all the genes in an individual (known as its genotype) corresponding to that point. The fitness of that individual is plotted as the height. If the genotypes can be visualised in two dimensions, the plot can be seen as a three-dimensional map, which may contain hills and valleys. Large regions of low fitness can be considered as swamps, while large regions of similar high fitness may be thought of as plateaus.

FIGURA 3.2: PAISAGEM DE APTIDÃO REPRESENTADA EM 2 DIMENSÕES

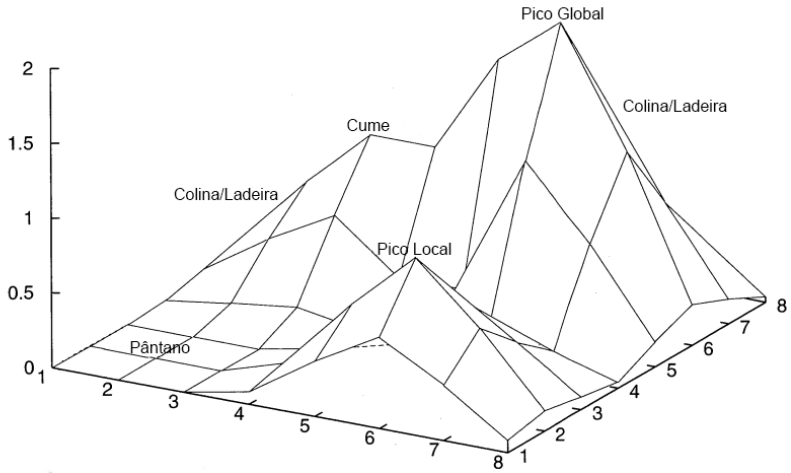


FONTE: Langdon and Poli (2002, p.5)

Desse modo, como foi possível observar, o processo de busca pela população inicial, que é formada de forma randômica, e que determina como será formada a população na geração seguinte, possui grande peso da aleatoriedade, porém de forma diferente do tipo de busca aleatória ou randômica apresentada anteriormente (ver Seção 3.1). Um dos grandes diferenciais do AG é a possibilidade de ter certo controle sobre seus processos aleatórios. Isso acontece, pois o algoritmo observa os dados históricos, que surgem a partir de cada nova geração, assim como segue alguns direcionamentos nas busca por novos pontos com maior chance de sobreviver ao ambiente. Estes direcionamentos são construídos por uma função objetivo (na biologia, chamada de função de aptidão[10]), sendo os seus parâmetros o que definirá quais serão os critérios de sobrevivência ou aptidão para que um ou mais indivíduos de uma população sigam para a geração seguinte, processo representado graficamente pelos pontos mais elevados na paisagem de aptidão na Figura 3.3.

[10]do inglês, *fitness function*.

FIGURA 3.3: PAISAGEM DE APTIDÃO REPRESENTADA EM 3 DIMENSÕES



FONTE: adaptado de Langdon and Poli (2002, p.5)

3.3.5 Reprodução ou Seleção

Dado um primeiro conjunto de indivíduos, chamado de população inicial, o AG precisa determinar quais destes indivíduos têm maior probabilidade de se adaptar ao ambiente e passar seus genes para a próxima geração através de seus descendentes. A esse processo é dado o nome de reprodução ou seleção.

A reprodução é um processo em que os indivíduos são replicados, copiados, conforme seus valores de aptidão, que são calculados pela função objetivo. Os indivíduos que possuem os maiores valores de aptidão têm maior probabilidade de criarem novos descendentes, passando, dessa forma, parte dos seus genes para a população seguinte. Nos sistemas naturais, o que determinará se um indivíduo passará por esse processo é sua capacidade de sobreviver a qualquer elemento que o possa matar antes que consiga se reproduzir, sendo ele um predador, uma doença, etc. (Goldberg (1989, p.11))

Conforme o exemplo da Caixa Preta apresentado na Seção 3.3, foi sugerida uma população inicial, criada aleatoriamente, com os indivíduos (interruptores) 01101, 11000, 01000 e 10011 (ver Tabela 3.1).

O primeiro passo será calcular o valor de aptidão de cada indivíduo, sua participação na população como um todo e ordená-los por valor de aptidão (Tabela 3.3).

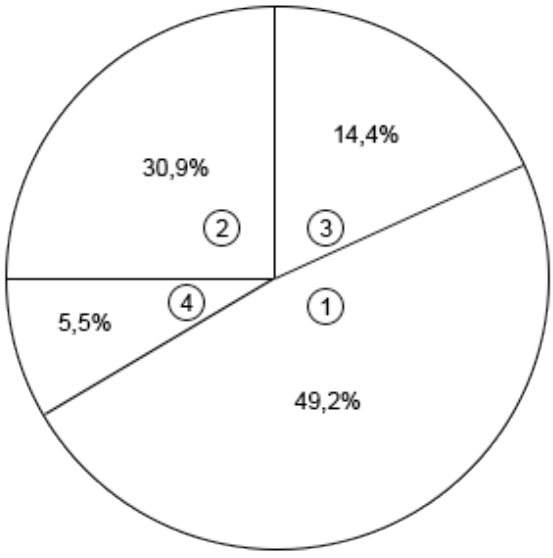
TABELA 3.3: VALORES DE APTIDÃO DOS INDIVÍDUOS DA POPULAÇÃO INICIAL

Índice	Indivíduo	Valor de Aptidão	Valor de Aptidão Acumulado	Participação em Relação à População	Participação Acumulada
1	11000	576	576	49,2%	49,2%
2	10011	361	937	30,9%	80,1%
3	01101	169	1106	14,4%	94,5%
4	01000	64	1170	5,5%	100%
Total		1170		100%	

FONTE: adaptado de Goldberg (1989, p.11)

Com base no peso de cada indivíduo frente à população, o operador da reprodução pode ser aplicado através de uma roleta[11] contendo 4 partes, cada parte relativa a um indivíduo, com proporções equivalentes às suas participações no total da população (Figura 3.4).

FIGURA 3.4: ROLETA ENVIESADA PARA APLICAÇÃO DO OPERADOR DE REPRODUÇÃO



FONTE: adaptado de Goldberg (1989, p.11)

A reprodução ocorre a cada girada na roleta. No caso do nosso exemplo, 4 vezes. Sendo assim, os indivíduos selecionados pelo operador de reprodução são copiados de

[11] O método da roleta é um dos vários possíveis na aplicação do operador de reprodução.

forma ordenada, sem nenhuma alteração, para um reservatório temporário de acasalamento[12] conforme vão sendo selecionados. Ou seja, os que tiverem maior valor de aptidão, logo maior chance de sobrevivência, possuem maiores chances de criarem um número maior de descendentes, passando parte de suas características para a próxima geração. Para o nosso exemplo, será utilizada uma taxa de reprodução de 100%, com todos os indivíduos sendo replicados. Esta taxa pode variar conforme o problema de otimização a ser resolvido.

3.3.6 Cruzamento

No reservatório de acasalamento, os indivíduos irão combinar suas características entre si, processo que é também realizado de forma aleatória. Supondo que, ao girar a roleta, os indivíduos com maior valor de aptidão foram selecionados primeiro que os indivíduos com menor aptidão, formando, assim, os seguintes pares:

TABELA 3.4: PARES FORMADOS NO RESERVATÓRIO DE ACASALAMENTO

Par 1	Indivíduo 1: 11000
	Indivíduo 2: 10011
Par 2	Indivíduo 3: 01101
	Indivíduo 4: 01000

FONTE: Elaborado pelo autor.

Para aplicação do operador de cruzamento, assim como no operador de reprodução, também pode ser utilizada a roleta para o sorteio dos pontos onde os indivíduos serão divididos para cruzamento. Porém, agora, contendo quatro partes iguais, uma para cada ponto entre o locus de cada gene, assim como cada giro equivalerá a um cruzamento por par. Deste modo, ao girar a roleta, vamos supor que tenham sido sorteados os números 2 e 3. Todos os caracteres à direita dos pontos sorteados são trocados entre os indivíduos dos pares formados, o que será representado pelo símbolo separador "|" (Tabela 3.5) e, após o sorteio dos pontos de cruzamento, é realizada a recombinação dos pares (Tabela 3.6).

[12] Do inglês, *mating pool*.

TABELA 3.5: PARES FORMADOS NO RESERVATÓRIO DE ACASALAMENTO COM SEPARADORES DE CRUZAMENTO

Par 1	Indivíduo 1: 1.1|0.0.0
	Indivíduo 2: 1.0|0.1.1
Par 2	Indivíduo 3: 0.1.1.0|1
	Indivíduo 4: 0.1.0.0|0

FONTE: Elaborado pelo autor.

TABELA 3.6: PARES FORMADOS NO RESERVATÓRIO DE ACASALAMENTO APÓS O CRUZAMENTO

Par 1	Indivíduo 1: 1.1|0.1.1
	Indivíduo 2: 1.0|0.0.0
Par 2	Indivíduo 3: 0.1.1.0|0
	Indivíduo 4: 0.1.0.0|1

FONTE: Elaborado pelo autor.

As novas subcadeias de caracteres, ou subvetores, são chamadas de blocos de construção. A Hipótese dos Blocos de Construção de um AG[13] será aprofundada mais à frente. Por agora, fica-se com a analogia apresentada por Goldberg:

> (...) considere uma população de n cadeias de caracteres (...), onde cada uma é uma *ideia* completa ou uma prescrição para realizar uma tarefa particular. As subcadeias de caracteres de cada cadeia de caracteres (ideia) contém várias *noções* do que é importante ou relevante para a tarefa. (...) Então, a ação de cruzamento com reproduções prévias é especulada sobre novas ideias construídas através dos blocos de construção de alto desempenho (noções) de tentativas passadas. (...) A troca de noções para formar novas ideias é intuitiva se nós pensarmos em termos do processo de *inovação*. (Goldberg, 1989, pg.13) (tradução nossa). [14]

[13] Do inglês, *Building block hypothesis*.

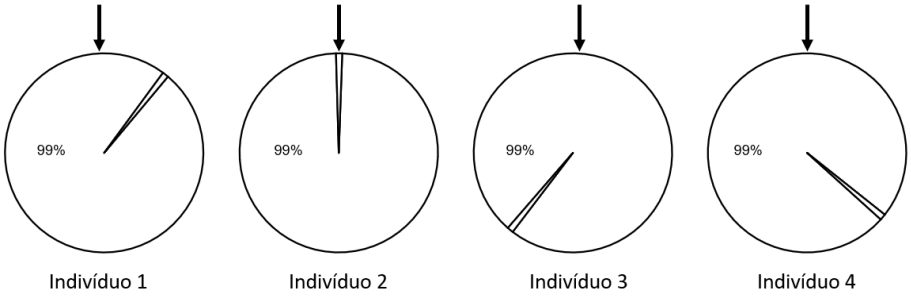
[14] (...), consider a population of n strings (...) over some appopriate alphabet, coded so that each is a complete *idea* or prescription for performing a particular task (...). Substrings within each string (idea) contain various *notions* of what is important or relevant to the task. Viewed in this way, the population contains not just a sample of n ideas; rather, it contains a multitude of notions and rankings of those notions for task performance. (...) Thus, the action of crossover with previus reproduction speculates on new ideas constructed form high-performance building blockis (notions) of past trials. (...) Exchanging of notions to form new ideas is appealing intuitively, if we think in termos of the process of *innovation*

3.3.7 Mutação

Após a aplicação dos operadores apresentados, ainda é necessário aplicar um último operador, o de mutação[15]. Os operadores de reprodução e cruzamento mantêm as características, informação genética, dos indivíduos mais aptos, porém essa aptidão é relacionada exclusivamente à geração corrente. Dessa forma, o AG pode convergir para uma solução mais rápido que o desejado e perder genes importantes ao longo do processo (genes que estão localizados em locus específicos em cada indivíduo). Portanto, o operador de mutação é uma garantia secundária de que haja diversidade nas populações que serão geradas ao longo das gerações, permitindo que o AG procure por soluções mais robustas. Ou seja, através da paisagem de aptidão, o algoritmo irá buscar de forma mais ampla pelo maior pico possível (ótimo global), diminuindo as chances de apresentar como solução picos menores (ótimos locais).

Em relação ao processo em si, a mutação é tão simples quanto a reprodução e o cruzamento. Utilizando novamente a roleta, será escolhido aleatoriamente, levando em consideração uma taxa, qual indivíduo sofrerá ou não uma mutação. Iremos nos aprofundar nesse ponto futuramente, mas, a título de exemplo, vamos utilizar uma taxa de mutação de 1%. Será realizado o giro da roleta para cada um dos indivíduos com o objetivo de selecionar qual ou quais sofrerão uma mutação. Supondo-se que o indivíduo 2 seja sorteado, temos:

FIGURA 3.5: SELEÇÃO ALEATÓRIA DE UM INDIVÍDUO PARA MUTAÇÃO

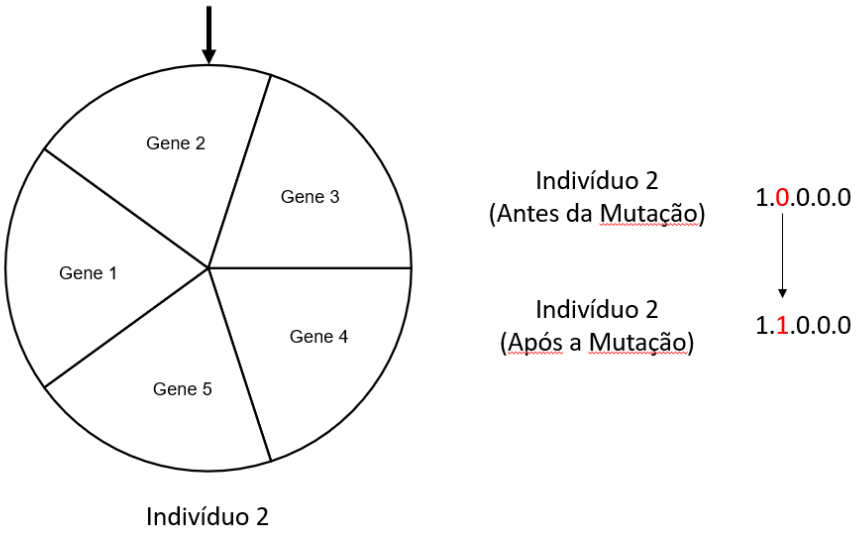


FONTE: Elaborado pelo autor.

Após a seleção dos indivíduos que sofreram a mutação, será definido, também de forma aleatória, em qual gene essa mutação ocorrerá. No caso do indivíduo 2, iremos supor que o gene no locus 2 foi o selecionado:

[15]. O operador de mutação não precisa, necessariamente, ser aplicado na última parte do processo. Isso irá variar conforme as estratégias de construção do AG.

FIGURA 3.6: MUTAÇÃO DO INDIVÍDUO SELECIONADO PARA MUTAÇÃO



FONTE: Elaborado pelo autor.

Como podemos observar no Tabela 3.7, através dos processos de reprodução, cruzamento e mutação é gerada uma nova população, com indivíduos contendo novas características, onde a segunda geração melhorou em relação à primeira, vide o valor total de aptidão de 1530 em comparação aos 1170 da primeira geração, o que a torna uma solução incrementalmente melhor, sendo o indivíduo 1 o que possui as características que mais auxiliarão na resolução do problema da Caixa Preta.

TABELA 3.7: VALORES DE APTIDÃO DOS INDIVÍDUOS DA POPULAÇÃO INICIAL E POPULAÇÃO NA GERAÇÃO SEGUINTE

População 1

Índice	Indivíduo	Valor de Aptidão	Valor de Aptidão Acumulado	Participação em Relação à População	Participação Acumulada
1	11000	576	576	49,2%	49,2%
2	10011	361	937	30,9%	80,1%
3	01101	169	1106	14,4%	94,5%
4	01000	64	1170	5,5%	100%
Total		1170		100%	

População 2

Índice	Indivíduo	Valor de Aptidão	Valor de Aptidão Acumulado	Participação em Relação à População	Participação Acumulada
1	11011	729	729	47,6%	47,6%
2	11000	576	1305	37,6%	85,3%
3	01100	144	1449	9,4%	94,7%
4	01001	81	1530	5,3%	100%
Total		1530		100%	

FONTE: Elaborado pelo autor.

Portanto, após apresentados todos os elementos da composição de um AG, assim como seus processos e operadores, fica mais claro como o algoritmo trabalhada para apresentar uma possível solução de um determinado problema. Claro, no exemplo da Caixa Preta, o indivíduo que apresentará as características que o definem como o mais apto, será o que contiver os genes 11111, porém, não sabemos qual será a geração em que esse indivíduo, ou solução, será criado, assim como não foram consideradas restrições para aplicação do AG, assunto que será tratado no capítulo seguinte.

Capítulo 4

Aplicação de Algoritmos Genéticos na Economia

Neste capítulo, busca-se apresentar algumas aplicações de AGs em problemas na economia e discussões voltadas a contextos econômicos, especialmente, nas áreas de otimização, inovação e aprendizagem adaptativa.

4.1 Introdução

A economia, sendo uma ciência social, não possui um laboratório totalmente controlado para se testar hipóteses e modelos, havendo a necessidade de se utilizar premissas que tornam possível a aplicação matemática de conceitos e modelos teóricos em busca de encontrar respostas para as perguntas a serem respondidas sobre o comportamento dos agentes econômicos e da sociedade como um todo. Contudo, muitas vezes, os modelos se apresentam limitados ou insuficientes quando testados em problemas reais complexos.

A gama de áreas de estudo que compõe a economia no que diz respeito ao escopo de análises de agentes, políticas públicas, conjecturas, entre outras, é vasta. Contudo, pode-se dividi-la em dois ramos principais: a macroeconomia e a microeconomia. Conforme apresenta Pindyck and Rubinfeld (2013, pg.3), a microeconomia estuda os agentes a nível individual, sendo trabalhadores, consumidores, investidores, empresas, etc. Nesse nível, analisa-se como os agentes que participam dos processos econômicos tomam suas decisões, fazem suas escolhas ou interagem entre si, formando agentes econômicos maiores que apenas a unidade analisada. De forma geral, a microeconomia busca observar indicadores não agregados, como a taxa de crescimento de um país, inflação, nível de desemprego, entre outros. De forma geral, cabe à macroeconomia a análise destes agregados, e será nela que iremos nos concentrar para entender as formas de aplicação dos AGs em contextos econômicos.

O escopo de estudos da macroeconomia é apresentado por Snowdon and Vane da seguinte maneira:

> A macroeconomia está preocupada com a estrutura, desempenho e comportamento da economia como um todo. A principal preocupação dos macroeconomistas é analisar e tentar entender os determinantes subjacentes das principais tendências agregadas da economia em relação à produção total de bens e serviços (PIB), desemprego, inflação e transações nacionais. Em particular, [1] a análise macroeconômica procura explicar a causa e o impacto das flutuações de curto prazo no PIB (o ciclo de negócios) e os principais determinantes da trajetória de longo prazo do PIB (crescimento econômico) (Snowdon and Vane, 2005, pg.1) (tradução nossa).

[1] Macroeconomics is concerned with the structure, performance and behaviour of the economy as a whole. The prime concern of macroeconomists is to analyse and attempt to understand the underlying determinants of the main aggregate trends in the economy with respect to the total output of goods and services (GDP), unemployment, inflation and international transactions. In particular, macroeconomic analysis seeks to explain the cause and impact of short-run fluctuations in GDP (the business cycle), and the major determinants of the long-run path of GDP (economic growth).

Capítulo 5

Exemplos de Aplicações de Algoritmos Genéticos

Neste capítulo, busca-se demonstrar a matemática, assim como sua representação programática, por trás de um AG para a resolução de um problema de otimização, levando em consideração os elementos apresentados no Capítulo 3. Para isso, utilizar-se-á uma adaptação do exemplo da Caixa Preta do livro *Genetic Algorithms in Search, Optimization and Machine Learning*, de David E. Goldberg (1989), bem como os processos realizados pelo AG serão construídos em Python, possibilitando maior entendimento prático através de exemplos computacionais em uma das linguagens de programação mais populares da atualidade[1] (código disponibilizado no Apêndice B). Ademais, foi disponibilizado no Apêndice A uma tabela sumarizando os principais termos e conceitos que serão demonstrados no presente capítulo contendo os termos dos sistemas naturais e seus equivalentes nos sistemas artificiais.

5.1 Definição do Problema

Dada uma Caixa Preta [2] com 5 interruptores, onde cada interruptor possui um sinal de entrada 0 ou 1 (desligado ou ligado) e um único sinal de saída como resultado (recompensa) de uma configuração específica, será aplicado um AG para encontrar qual é a configuração que resultará no maior valor de recompensa, considerando o problema de maximização representado pela função

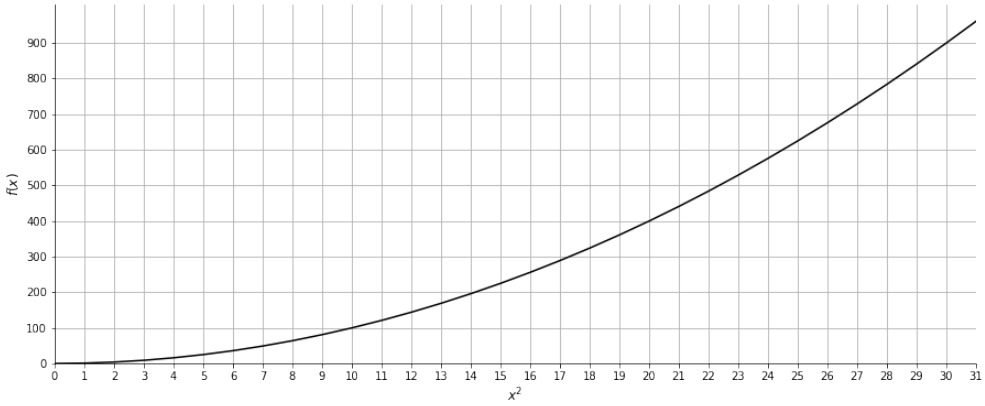
$$f(x) = x^2,$$

onde x pode assumir qualquer inteiro no intervalo $[0, 31]$, função representada na Figura 5.1.

[1] Segundo dados divulgados pelo site TIOBE noa

[2] Ibid, p. 12.

FIGURA 5.1: GRÁFICO DA FUNÇÃO $f(x) = x^2$



FONTE: adaptado de Goldberg (1989, pg.8)

5.2 Construção da População Inicial

O AG é, inicialmente, agnóstico em relação a maioria dos parâmetros que irá utilizar ao longo do processo de um problema de otimização. Como explicado por Goldberg (1989, pg.8), não há a necessidade de trabalhar diretamente com o conjunto de parâmetros de entrada, como outros algoritmos de otimização podem demandar, sendo necessário apenas codificar os valores de entrada em valores binários.

Como visto anteriormente (ver Seção 3.2), o primeiro passo é codificar os parâmetros de entrada do algoritmo, o que será realizado através de um alfabeto binário representado por $V = \{0, 1\}$. Assim, havendo 5 parâmetros de entrada (interruptores com sinal de desligado ou ligado), é possível construir uma representação através de um vetor, ou cadeia de caracteres, de 5 dígitos, que vai de 00000 (inteiro 0) a 11111 (inteiro 31). Esse vetor pode ser expressado como

$$A_i = a_1 a_2 a_3 a_4 a_5 ... a_n,$$

FONTE: adaptado de Goldberg (1989, pg.25)

onde a notação A representa um vetor (indivíduo) e a representa uma característica específica (gene), subscrita por seu locus, contendo o alelo com valor 0 ou 1. As características não precisam, necessariamente, estarem ordenadas. Para ganho de desempenho, por exemplo, podem ser construídos vetores onde as características estejam fora de ordem em relação às suas posições originais, porém continuam com as subscrições de seus locus como, por exemplo, a variação do vetor anterior

$$A'_i = a_3 a_1 a_5 a_2 a_4 ... a_n,$$

FONTE: adaptado de Goldberg (1989, pg.25)

sendo o apóstrofo (') um símbolo de variação ou derivação de algum vetor específico.

Após a decodificação dos valores de entrada, é formada uma população inicial aleatória de tamanho n. Será considerada uma população inicial de tamanho $n = 5$ vetores, denotada por $\mathbf{A}_j, j = 1, 2, ..., n$, onde o caractere maiúsculo e em negrito $\mathbf{A}$ representa uma população num dado momento t (iteração ou geração). A construção randômica dessa população será realizada através do lançamento de uma moeda honesta ($p_{cara} = p_{coroa} = 50\%$) para cada característica dos 5 vetores, ou seja, 25 lançamentos. Em outras palavras, para cada coroa sorteada, é adicionado um 0 na população e, para cada cara, um 1. Assim, têm-se os seguintes resultados baseados em uma simulação:

TABELA 5.1: POPULAÇÃO INICIAL DE TAMANHO $n = 5$

População Inicial				
$\mathbf{A}_1 = 010110011110111111 0001011$				
Vetor A_{11}	Vetor A_{21}	Vetor A_{31}	Vetor A_{41}	Vetor A_{51}
01011	00111	10111	11100	01011

FONTE: Elaborado pelo autor.

5.3 Cálculo dos Valores de Aptidão

Com a população inicial construída, é necessário decodificar os valores binários de cada vetor, o que pode ser feito através da fórmula

$$a_{ij}{}^{l-1-i}, \tag{5.1}$$

onde l é o tamanho do vetor e a é o valor do alelo no locus i da geração j.

Após a decodificação de cada valor binário, é feita a soma de todos os novos valores do vetor para, posteriormente, aplicar a função objetivo e ordenar os vetores conforme seus valores de aptidão (resultados apresentados na Tabela 5.2). Portanto, para o cálculo do valor total do vetor decodificado, aplica-se:

$$x_n = \sum_{ij=1}^{n} a_{ij}{}^{l-1-i} \tag{5.2}$$

TABELA 5.2: VALORES DE APTIDÃO DOS VETORES DA POPULAÇÃO INICIAL

Nome Vetor	Vetor	Vetor com Alelos Decodificados	Valor de x	Valor de Aptidão ($f(x) = {x_{ij}}^2$)
A_4	11100	$[16, 8, 4, 0, 0]$	28	784
A_3	10111	$[16, 0, 4, 2, 1]$	23	529
A_1	01011	$[0, 8, 0, 2, 1]$	11	121
A_5	01011	$[0, 8, 0, 2, 1]$	11	121
A_2	00111	$[0, 0, 4, 2, 1]$	7	49
		Mínimo	7	49
		Média	16	320,8
		Soma	80	1604
		Máximo	28	784

FONTE: Elaborado pelo autor.

5.4 Operadores de Reprodução, Cruzamento e Mutação

Calculados os valores de aptidão, o passo seguinte será criar uma nova geração de indivíduos através da aplicação dos operadores de reprodução, cruzamento e mutação.

5.4.1 Reprodução

Como visto na Subseção 3.3.5, a reprodução é feita de forma aleatória levando em consideração uma probabilidade de seleção. Há diversas formas de "sortear" os vetores que serão selecionadas para o envio ao reservatório de acasalamento. Para este exemplo, a taxa de reprodução da população será de 100%, ou seja, 5 vetores reproduzidos, e será aplicado o método da roleta para sorteio dos indivíduos que serão selecionados para o reservatório. A probabilidade de reprodução de cada indivíduo será igual ao peso do seu valor de aptidão em relação à população, assim como será calculada a probabilidade de seleção esperada como métrica de comparação, sendo a probabilidade calculada por

$$p_s(x_{ij}) = \frac{f(x_{ij})}{\sum_{ij=1}^{n} f(x_{ij})} \tag{5.3}$$

e a probabilidade esperada por

$$E_s(x_{ij}) = \frac{f(x_{ij})}{\overline{f(x_{ij})}} \tag{5.4}$$

Como é possível observar na Tabela 5.3, a simulação através do método da roleta resultou em 3 reproduções do Vetor A_4, 1 reprodução dos Vetores A_3 e A_5 nenhuma reprodução de A_1 e A_2. O resultado foi próximo ao esperado, com um pequeno desvio

no Vetor A_3, que estava mais próxima de 2 reproduções do que uma, e no Vetor A_4, que estava mais próxima de 2 reproduções do que as 3 resultantes. Contudo, pode-se observar que os resultados seguiram as ideias apresentadas até o momento, onde os indivíduos que possuem maior valor de aptidão em relação ao ambiente, têm uma tendência a serem selecionados para reprodução mais vezes, assim como os que possuem valores baixos de aptidão têm chances de serem selecionados poucas vezes ou não serem selecionados, o que significa a morte destes indivíduos.

TABELA 5.3: RESULTADOS APÓS A REPRODUÇÃO DOS VETORES DA POPULAÇÃO INICIAL

Nome Vetor	Vetor	Valor de x	Valor de Aptidão ($f(x) = {x_i}^2$)	Probabilidade de Reprodução	Nº Esperado de Seleções	Nº de Seleções (Roleta)
A_4	11100	28	784	48,9%	2,44	3
A_3	10111	23	529	33%	1,65	1
A_1	01011	11	121	7,5%	0,38	0
A_5	01011	11	121	7,5%	0,38	1
A_2	00111	7	49	3,1%	0,15	0
	Mínimo	7	49	0,7%	0,15	0
	Média	16	320,88	20%	1,00	1,00
	Soma	80	1604	100%	5,00	5,00
	Máximo	28	784	36,9%	2,44	3,00

FONTE: Elaborado pelo autor.

5.4.2 Cruzamento

O passo seguinte, é a aplicação do operador de cruzamento entre os indivíduos que foram copiados para o reservatório de acasalamento. Este processo será feito em 3 partes, sendo elas: a formação aleatória de pares; a escolha aleatória dos pontos de cruzamento; e, por último, o cruzamento entre os pares e a construção das novos vetores.

5.4.2.1 Formação dos pares

Como foram replicados 5 vetores para o reservatório de acasalamento, se for seguida uma premissa de pares exclusivos, um dos vetores não encontrará um par, logo, não será capaz de efetivamente criar um descendente. Há várias estratégias que podem ser seguidas para lidar com esse ponto. Em relação à população construída no exemplo, será considerada a premissa de que todos os indivíduos façam parte de, ao menos, um par. Ou seja, haverá pelo menos um indivíduo que fará par com dois outros indivíduos da população.

Para a formação dos pares, será sorteado um vetor em seguida do outro, sendo que o último vetor selecionado formará par com o selecionado anteriormente. Estes, vão sendo retirados como opções no sorteio à medida que forem sendo escolhidos. No caso

do último vetor, que ficaria sem par nesse processo, será realizado um novo sorteio para seleção de um dos 4 vetores já selecionados, resultando, assim, na formação de 3 pares para cruzamento. A probabilidade de seleção pode ser representada pela fórmula recursiva

$$p_c(x_n) = \begin{cases} 1 & \text{se } n-1=1 \\ \frac{1}{n-1} & \text{se } 1 < n \leq n-1 \end{cases} \tag{5.5}$$

5.4.2.2 Escolha do ponto de cruzamento

Com a formação dos pares para cruzamento, será sorteado em qual ponto do vetor ocorrerá a troca de informações. Como ilustrado por Goldberg (1989, p.12), este processo é bastante simples: escolhida uma posição k de forma aleatória, os vetores pares trocarão todas as informações da posição $k+1$ a l. Cada posição k, é localizada entre os valores binários e representada por um inteiro no intervalo $[1, l-1]$, com probabilidade de escolha aleatória do ponto de cruzamento definida por

$$p_k[x_1, x_2] = \frac{1}{l-1} \tag{5.6}$$

5.4.2.3 Formação dos novos vetores

Por último, definido o ponto de cruzamento, os vetores trocam informações entre si, formando vetores filhos. Em outras palavras, utilizando como exemplo os Vetores $A_4 = 11100$ e $A_3 = 10111$, supondo um valor sorteado de $k = 2$ no intervalo $[1, 4]$, todos os valores binários do locus 3 ao 5 serão trocados entre os dois vetores que formam o par, resultando nos vetores $A'_4 = 11111$ e $A'_3 = 10100$, que farão parte da geração seguinte. Os resultados da aplicação do operador de cruzamento nos indivíduos enviados para o reservatório de cruzamento são apresentados na Tabela 5.4.

Como é possível observar, utilizando a premissa de que todos os indivíduos fizessem parte de ao menos um par, o Vetor A_1 foi sorteado como par do Vetor A_0 e do Vetor A_4, Par 2 e Par 3 respectivamente. Para o Par 1, foi selecionado o ponto de cruzamento $k = 3$ localizado entre o locus 3 e 4 e para os Pares 2 e 3, foi selecionado o ponto $k = 2$, entre os locus 2 e 3. Na Tabela 5.5, é possível observar os vetores após o cruzamento entre os pares.

TABELA 5.4: FORMAÇÃO DOS PARES E SORTEIO DO PONTO DE CRUZAMENTO

Id Par	Id Vetores	Vetores	Ponto de Cruzamento Sorteado	Vetores com Ponto de Cruzamento
Par 1	$[A_2, A_3]$	$[11100, 10111]$	3	$[1.1.1\|0.0, 1.0.1\|1.1]$
Par 2	$[A_1, A_0]$	$[11100, 11100]$	2	$[1.1\|1.0.0, 1.1\|1.0.0]$
Par 3	$[A_4, A_1]$	$[01011, 11100]$	2	$[0.1\|0.1.1, 1.1\|1.0.0]$

FONTE: Elaborado pelo autor.

TABELA 5.5: VETORES APÓS APLICAÇÃO DO OPERADOR DE CRUZAMENTO

Id Par	Id Vetores Pais	Vetores Pais	Id Vetor Filho (Após Cruzamento))	Vetor Filho (Após Cruzamento)
Par 1	$[A_2, A_3]$	$[11100, 10111]$	A'_1	11111
Par 1	$[A_2, A_3]$	$[11100, 10111]$	A'_2	10100
Par 2	$[A_1, A_0]$	$[11100, 11100]$	A'_3	11100
Par 2	$[A_1, A_0]$	$[11100, 11100]$	A'_4	11100
Par 3	$[A_4, A_1]$	$[01011, 11100]$	A'_5	01100
Par 3	$[A_4, A_1]$	$[01011, 11100]$	A'_6	11011

FONTE: Elaborado pelo autor.

5.4.3 Mutação

O operador final a ser aplicado, é o de mutação[3]. Como apresentado na Subseção 3.3.7, a mutação é um recurso secundário para evitar que o algoritmo retorne picos locais, ao invés do pico global, como resultado da busca. A taxa de mutação é aplicada bit a bit, o que, em outras palavras, significa que cada elemento da característica de um indivíduo possui uma probabilidade bem pequena de mudança devido à pressão do ambiente. Para o presente exemplo, será considerada a probabilidade de mutação apresentada por Goldberg (1989, pg.25) de 0,1%, ou:

$$p_m(a_{ij}) = 0,001 \tag{5.7}$$

e o número esperado de bits que sofrerão mutação por

$$E_m(a_{ij}) = n \cdot l \cdot p_m(a_{ij}) \tag{5.8}$$

[3]Ibid, p. 20.

Esperava-se que 0.025 ($5 \cdot 5 \cdot 0,001$) bits sofressem mutação, ou seja, nenhum. Após a aplicação do operador de mutação nos dados simulados, como esperado, nenhum dos 30 bits sofreu mutação.

Referências

index | TIOBE - The Software Quality Company.

(1993). Editorial Introduction. *Evolutionary Computation*, 1(1):iii–v.

Back, T., Fogel, D. B., and Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*. IOP Publishing Ltd., GBR, 1st edition.

Bremermann, H. J. (1962). Optimization through evolution and recombination. page 12.

Bremermann H J, R. M. and S, S. Search by evolution biophysics and cybernetic. page 157–67.

Cavicchio, D. J. (1970). Adaptive search using simulated evolution. Accepted: 2006-02-03T18:12:14Z.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2009). *Introduction to Algorithms, third edition*. Computer science. MIT Press.

Eiben, A., Hinterding, R., and Michalewicz, Z. (1994). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*.

Eiben, A. and Smith, J. (2015). *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, Berlin, Heidelberg.

Fedeli, R., Polloni, E., and Peres, F. (2009). *Introdução à ciência da computação*. Cengage Learning, 2ª edição edition.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). Artificial intelligence through simulated evolution.

Friedberg, R. M. (1958). A learning machine: Part i. *IBM Journal of Research and Development*, 2(1):2–13.

Friedberg, R. M., Dunham, B., and North, J. H. (1959). A learning machine: Part ii. *IBM J. Res. Dev.*, 3:282–287.

Goldberg, V. A. P. o. H. D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, Reading, Mass, 13th ed. edição edition.

Gorn, S., Bemer, R. W., and Green, J. (1963). American standard code for information interchange. *Communications of the ACM*, 6(8):422–426.

Holland, J. Nonlinear environments permitting efficient adaptation.

Holland, J. H. (1962). Outline for a logical theory of adaptive systems. *J. ACM*, 9(3):297–314.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, Oxford, England. Pages: viii, 183.

Holland, J. H. (1992). Genetic Algorithms. *Scientific American*, 267(1):66–73. Publisher: Scientific American, a division of Nature America, Inc.

Hollstien, R. B. (1971). *Artificial genetic adaptation in computer control systems.* OCLC: 726084376.

Jong, D. and Alan, K. (1975). Analysis of the behavior of a class of genetic adaptive systems. Accepted: 2006-02-03T19:05:13Z.

Langdon, W. B. and Poli, R. (2002). *Foundations of Genetic Programming*. Springer, Berlin, Heidelberg.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. The MIT Press.

Pindyck, R. S. and Rubinfeld, D. L. (2013). *Microeconomia*. Pearson Universidades, 8ª edição edition.

Rechenberg, I., Toms, B., and Establishment, R. A. (1965). *Cybernetic Solution Path of an Experimental Problem:*. Library translation / Royal Aircraft Establishment. Ministry of Aviation.

Snowdon, B. and Vane, H. R. (2005). *Modern Macroeconomics: Its Origins, Development and Current Stage*. Edward Elgar Publishing Limited, Cheltenham - UK.

Apêndices

Apêndice A

Sumário de Terminologia Natural e Terminologia Artificial

Terminologia Natural	Terminologia Artificial	Terminologia Artificial (Inglês)	Descrição Curta
Locus	Locus	*Locus*	Posição de um gene (ou característica) dentro de uma cadeia de caracteres.
Alelo	Alelo	*Alelle*	Valor do gene (ou característica), sendo 0 ou 1.
Gene	Característica	*Gene*	Característica, caractere ou algoritmo de uma cadeia de caracteres.
Bloco de construção	Bloco de Construção	*Building Block*	Subvetor de alto desempenho. Responsável por formar novos vetores.
Cromossomo ou Indivíduo	Vetor ou Cadeia de caracteres	*String*	Conjunto de características agrupadas através de um conjunto de caracteres. Cada matriz pode ser uma possível solução para um problema ou fazer parte de uma solução global.
Genótipo	Estrutura	*Structure*	Conjunto de vetores que mantém uma carga ou estrutura genética.
Fenótipo	Estrutura decodificada	*Decoded structure*	Nova carga ou estrutura genética após as alterações das características principais de uma
População	População	*Population*	Conjunto de indivíduos construídos, inicialmente, de forma aleatória, onde o AG irá realizar suas buscas. Cada geração é composta por uma população que pode ser uma possível solução para o problema de otimização.

Geração	Iteração	*Iteration*	Formação de uma nova população após a aplicação dos operadores de reprodução, cruzamento e mutação.
Reprodução ou Seleção	Operador de Reprodução	*Reproduction*	Seleção dos indivíduos com maior aptidão ao ambiente.
Cruzamento	Operador de Cruzamento	*Crossover*	Troca de genes, ou características, entre os indivíduos de uma população.
Mutação	Operador de Mutação	*Mutation*	Processo aleatório de alteração de uma ou mais características de um indivíduo.
Valor de Aptidão	Valor de Aptidão	*Fitness value*	Quantificação da qualidade de um indivíduo ou sua aptidão em relação ao ambiente.
Paisagem ou Horizonte de aptidão	Paisagem de Aptidão	*Fitness landscape*	Espaço pelo qual o AG faz sua busca por soluções, ou seja, diz respeito ao problema a ser resolvido.
Função de aptidão	Função objetivo	*objective function*	Função contendo os parâmetros para resolução do problema.

Apêndice B

Exemplo de Aplicação de um AG simples em Python

```
# -*- coding: utf-8 -*-

'''
Monolito de aplicacao de um AG simples para otimizacao da funcao f(x) = x^2
'''

# ------------------------------------------------------------------
# ------------------------------------------------------------------
# IMPORTACAO DAS BIBLIOTECAS
# ------------------------------------------------------------------
# ------------------------------------------------------------------

from statistics import mean
from typing import Union, List
import numpy as np
from collections import Counter
import pandas as pd
from pathlib import Path

import ipdb

ipdb.set_trace()

# from collections import Counter
# import numpy as np # biblioteca para diferentes calculos numericos
# import pandas as pd

# ------------------------------------------------------------------
# ------------------------------------------------------------------
# DEFINICAO DO PROBLEMA
# ------------------------------------------------------------------
# ------------------------------------------------------------------

# Definicao do Problema:
# # Dada uma Caixa Preta, contendo 5 interruptores com um sinal de entrada
# # de 0 ou 1 (Desligado ou Ligado), defina a configuracao de interruptores
# # com o maior sinal de saida (recompensa), atraves da maximizacao da fun-
# # cao objetivo f(x) = x**2, onde x pode apresentar qualquer inteiro na
# # intervalo [0, 31].
# #
# # Adaptado de Goldberg (1889)

# ------------------------------------------------------------------
```

```
# ----------------------------------------------------------
# VARIAVEIS GLOBAIS
# ----------------------------------------------------------
# ----------------------------------------------------------

# Cada variavel global sera representada por letras maiusculas.

# ----------------------------------------------------------
# INDICADORES PARAMETRIZAVEIS
# ----------------------------------------------------------

# tamanho da populacao
N = 5
# numero de caracteres de cada matriz
NUM_CARACTERISTICAS = 5
# numero de caracteres de cada matriz
PROBABILIDADE_MUTACAO = 0.001
# indicador de uso do gerador de numeroes pseudo aleatorios
# se 1, entao usar gerador; se 0, nao usar;
USAR_RANDOMSTATE = 1
# indicador de uso da populacao gerada como exemplo na monografia ...
# DA COSTA, E. (2022) ou qualquer outra populacao predefinida
# se 1, entao usar populacao predeterminada; se 0, nao usar;
USAR_POPULACAO_INICIAL_PREDETERMINADA = 1
# indicador de uso da matriz nao selecionada para cruzamento no reservatorio
# de cruzamento. Esse caso ocorre quando a populacao tem numero par
# se 1, formar um par para a matriz nao selecionada, senao, ignorar matriz
CONSIDERAR_MATRIZ_SEM_PAR = 1
# numero maximo de geracoes (criterio de parada)
NUM_MAX_GERACOES = 2

# ----------------------------------------------------------
# VARIAVEIS DINAMICAS
# ----------------------------------------------------------

# contador de iteracoes (passagem de geracoes)
CONTADOR_GERACAO = 0
# ---
# populacao inicial gerada monografia DA COSTA, E. (2022)
POPULACAO_INICIAL_PREDEFINIDA = '01011001111011111110001011'
# populacao inicial gerada aleatoriamente
POPULACAO = {}
# ---
# dicionario com os dados dos processos do AG ate a aplicacao dos operadores
BASE_POPULACOES_PRE_OPERADORES = {}
# dicionario com os dados dos processos do AG ate a aplicacao dos operadores
BASE_POPULACOES_POS_OPERADORES = {}

# gerador de numeros pseudo aleatorios.
# necessario para replicacao dos resultados, quando se deseja os mesmos nu-
# meros de saida apos varias execucoes do script.
if (USAR_RANDOMSTATE == 0):
    rnd = np.random.RandomState()
else:
    rnd = np.random.RandomState(15)

# ----------------------------------------------------------
# ----------------------------------------------------------
# FUNCAO OBJETIVO E FUNCAO DE CONVERSAO DE VALORES DE BASE 2 PARA BASE 10
# ----------------------------------------------------------
# ----------------------------------------------------------

def funcao_objetivo(valor_x: Union[int, float]) -> Union[int, float]:
    """Funcao f(x) = x^2. Recebe o valor do alelo decodificado e retorna
```

```
    o valor de aptidao.

    Parameters
    ----------
    valor_x : int | float
        Numero decodificado para aplicacao da funcao objetivo.

    Return
    ----------
    int | float
        valor de aptidao.
    """
    return valor_x ** 2

def decodificacao_base_dois(matriz: List[str]) -> List[int]:
    """Decodificacao dos valores binarios do sistema numerico de base 2 para
    inteiros de base 10 ${a_{ij}}^{n-i}$.

    Parameters
    ----------
    matriz : list
        Lista com os valores binarios para decodificacao.

    Return
    ----------
    list
        uma lista com os valores binarios decodificados para base 10.
    """
    base = 2
    tamanho_matriz = len(matriz)
    lista_binarios = [_ for _ in range(tamanho_matriz)]
    lista_binarios_decodificados = []
    for id_binario in lista_binarios:
        valor_binario = int(matriz[id_binario])
        valor_base_dez = valor_binario * base ** (tamanho_matriz - 1 - id_binario)
        lista_binarios_decodificados.append(valor_base_dez)
    return lista_binarios_decodificados

# ----------------------------------------------------------
# ----------------------------------------------------------
# 1. CONSTRUCAO DA POPULACAO INICIAL
# ----------------------------------------------------------
# ----------------------------------------------------------

# opcoes de criterio de parada
OPCOES_CRITERIO_PARADA = ['num_maximo_geracoes', 'valor_aptidao_maximo_alcancado',
    'valor_aptidao_corrente_menor_que_anterior']
# ---
criterio_parada = OPCOES_CRITERIO_PARADA[1]
# ---
binario_num_max_geracoes = False
binario_valor_mais_aptidao_alcancado = False
binario_valor_aptidao_corrente_menor_que_anterior = False

# iterador de geracoes levando em consideracao os criterios de parada
while not (
    binario_num_max_geracoes if criterio_parada == 'num_maximo_geracoes' else (
        binario_valor_mais_aptidao_alcancado if criterio_parada == 'valor_aptidao_maximo_alcancado' else
            binario_valor_aptidao_corrente_menor_que_anterior
        )
    ):

    print(f'Inicio: Populacao Inicial/Geracao {CONTADOR_GERACAO+1}' if CONTADOR_GERACAO
        == 0 else f'Inicio: Geracao {CONTADOR_GERACAO + 1}')
```

```
breakpoint()

# se base de dados das populacoes estiverem vazias, gera a populacao inicial
if not BASE_POPULACOES_PRE_OPERADORES and not
     BASE_POPULACOES_POS_OPERADORES:
    num_lancamentos = N * NUM_CARACTERISTICAS
    lados_moeda = [0, 1]
    probabilidades = [.5, .5]
    # lancamento da moeda para cada caracteristica da populacao
    lista_lancamentos_moeda = rnd.choice(lados_moeda, size=num_lancamentos, p=probabilidades)
    # ---
    # usar populacao inicial predefinida
    if USAR_POPULACAO_INICIAL_PREDETERMINADA == 1:
        lista_caracteres_predefinidos = [int(caractere) for caractere in
            POPULACAO_INICIAL_PREDEFINIDA]
        for id_caractere, caractere in enumerate(lista_caracteres_predefinidos):
            num_caractere = f'valor_alelo_{id_caractere+1}'
            POPULACAO[num_caractere] = caractere
    # usar populacao inicial aleatoria
    else:
        for id_lancamento, lancamento in enumerate(lista_lancamentos_moeda):
            num_lancamento = f'valor_alelo_{id_lancamento+1}'
            POPULACAO[num_lancamento] = lancamento
else:
    populacao_interior =
        BASE_POPULACOES_POS_OPERADORES.get(f'id_geracao_{CONTADOR_GERACAO-1}')
    alelos_populacao = []
    for matriz in populacao_interior:
        matriz_populacao_anteior = populacao_interior.get(matriz)['matriz_apos_operador_mutacao']
        for alelo in matriz_populacao_anteior:
            alelos_populacao.append(alelo)
    for id_alelo, alelo in enumerate(alelos_populacao):
        num_caractere = f'valor_alelo_{id_alelo+1}'
        POPULACAO[num_caractere] = alelo

# ------------------------------------------------------------
# 1.1 AGRUPAMENTO DOS CARACTERES DA POPULACAO (CRIACAO MATRIZES)
# ------------------------------------------------------------

# dicionario com as informacoes gerada das matrizes da populacao inicial
matrizes = {}
i = 0 # numero de particionamento inicial
for num_caracteristica in range(NUM_CARACTERISTICAS):
    lista_posicao_alelos_populacao = [locus for locus in range(len(POPULACAO))]
    lista_alelos_populacao = list(POPULACAO.values())
    # ---
    # quebra da populacao por individuo e caracteristicas
    alelos_lista = lista_posicao_alelos_populacao[i:i+NUM_CARACTERISTICAS]
    matriz_lista = lista_alelos_populacao[i:i+NUM_CARACTERISTICAS]
    locus_lista = [locus for locus in range(len(alelos_lista))]
    # ---
    # envio dos dados construidos para um dicionario
    key = f'id_{num_caracteristica+1}'
    matriz_str = ''.join([str(alelo) for alelo in matriz_lista])
    matrizes[key] = {
        'matriz': matriz_str,
        'lista_id_alelo_populacao': alelos_lista,
        'lista_locus': locus_lista,
        'lista_valor_alelo': matriz_lista,
    }
    i += NUM_CARACTERISTICAS

# ------------------------------------------------------------
```

```
# 1.2 DECODIFICACAO DOS VALORES BINARIOS DAS MATRIZES
# ————————————————————————————————————————

for id_matriz, matriz in enumerate(matrizes):
    alelos_int = matrizes.get(matriz)
    alelos_str = [str(alelo) for alelo in alelos_int['lista_valor_alelo']]
    # ———
    # conversao de base 2 para base 10
    alelos_decodificados = (decodificacao_base_dois(alelos_str))
    # conversao de base 2 para base 10 levando em considracao que todos os valores dos alelos sao 1
    # necessario para a definicao do criterio de parada
    alelos_decodificados_maximos = decodificacao_base_dois('1' * len(alelos_str))
    # ———
    # alimentacao dos resultados do dicionario das matrizes
    matrizes.get(matriz).update({'lista_alelos_decodificados': alelos_decodificados})
    matrizes.get(matriz).update({'lista_alelos_decodificados_maximos': alelos_decodificados_maximos})

# ————————————————————————————————————————————
# 1.2.1 SOMA DOS VALORES DECODIFICADOS
# ————————————————————————————————————————————

for id_matriz, matriz in enumerate(matrizes):
    lista_alelos_decodificados = matrizes.get(matriz)['lista_alelos_decodificados']
    lista_alelos_decodificados_maximos = matrizes.get(matriz)['lista_alelos_decodificados_maximos']
    # ———
    # alimentacao dos resultados do dicionario das matrizes
    matrizes.get(matriz).update({'valor_de_x': sum(lista_alelos_decodificados)})
    matrizes.get(matriz).update({'max_valor_de_x_possivel': sum(lista_alelos_decodificados_maximos)})

# ————————————————————————————————————————————
# ————————————————————————————————————————————
# 2 CALCULO DOS VALORES DE APTIDAO
# ————————————————————————————————————————————
# ————————————————————————————————————————————

for id_matriz, matriz in enumerate(matrizes):
    valor_x = matrizes.get(matriz)['valor_de_x']
    valor_maximo_x = matrizes.get(matriz)['max_valor_de_x_possivel']
    # ———
    valor_aptidao = funcao_objetivo(valor_x)
    valor_maximo_aptidao = funcao_objetivo(valor_maximo_x)
    # ———
    # alimentacao dos resultados do dicionario das matrizes
    matrizes.get(matriz).update({'valor_aptidao': valor_aptidao})
    matrizes.get(matriz).update({'max_valor_aptidao_possivel': valor_maximo_aptidao})

# ————————————————————————————————————————————
# 2.1 ORDENACAO DAS MATRIZES
# ————————————————————————————————————————————

dicionario_matrizes_temp = {}
for matriz in matrizes:
    dicionario_matrizes_temp[matriz] = matrizes.get(matriz)['valor_aptidao']

matrizes_ordenadas = {}
for matriz in sorted(dicionario_matrizes_temp, key=dicionario_matrizes_temp.get, reverse=True):
    matrizes_ordenadas[matriz] = matrizes.get(matriz)

# ————————————————————————————————————————————
# 2.2 CALCULO DOS MINIMOS, MEDIOS, SOMAS E MAXIMOS DAS MATRIZES
# ————————————————————————————————————————————————

lista_ids_matrizes = [id_matriz for id_matriz in matrizes_ordenadas.keys()]
```

```
for matriz in matrizes_ordenadas:
    min_valor_x = min([matrizes_ordenadas.get(_matriz)['valor_de_x'] for _matriz in lista_ids_matrizes])
    media_valor_x = mean([matrizes_ordenadas.get(_matriz)['valor_de_x'] for _matriz in lista_ids_matrizes])
    soma_valor_x = sum([matrizes_ordenadas.get(_matriz)['valor_de_x'] for _matriz in lista_ids_matrizes])
    max_valor_x = max([matrizes_ordenadas.get(_matriz)['valor_de_x'] for _matriz in lista_ids_matrizes])
    # ---
    # alimentacao dos resultados do dicionario das matrizes
    matrizes_ordenadas.get(matriz).update({'min_valor_x': min_valor_x})
    matrizes_ordenadas.get(matriz).update({'media_valor_x': media_valor_x})
    matrizes_ordenadas.get(matriz).update({'soma_valor_x': soma_valor_x})
    matrizes_ordenadas.get(matriz).update({'max_valor_x': max_valor_x})

for matriz in matrizes_ordenadas:
    min_valor_aptidao = min([matrizes_ordenadas.get(_matriz)['valor_aptidao'] for _matriz in
        lista_ids_matrizes])
    media_valor_aptidao = mean([matrizes_ordenadas.get(_matriz)['valor_aptidao'] for _matriz in
        lista_ids_matrizes])
    soma_valor_aptidao = sum([matrizes_ordenadas.get(_matriz)['valor_aptidao'] for _matriz in
        lista_ids_matrizes])
    max_valor_aptidao = max([matrizes_ordenadas.get(_matriz)['valor_aptidao'] for _matriz in
        lista_ids_matrizes])
    # ---
    # alimentacao dos resultados do dicionario das matrizes
    matrizes_ordenadas.get(matriz).update({'min_valor_aptidao': min_valor_aptidao})
    matrizes_ordenadas.get(matriz).update({'media_valor_aptidao': media_valor_aptidao})
    matrizes_ordenadas.get(matriz).update({'soma_valor_aptidao': soma_valor_aptidao})
    matrizes_ordenadas.get(matriz).update({'max_valor_aptidao': max_valor_aptidao})

# ------------------------------------------------------------
# ------------------------------------------------------------
# 3. APLICACAO DOS OPERADORES
# ------------------------------------------------------------
# ------------------------------------------------------------

# ------------------------------------------------------------
# 3.1 REPRODUCAO
# ------------------------------------------------------------

# ------------------------------------------------------------
# 3.1.1 CALCULO DA PROBABILIDADE DE SELECAO
# ------------------------------------------------------------

for matriz in matrizes_ordenadas:
    valor_aptidao = matrizes_ordenadas.get(matriz)['valor_aptidao']
    soma_valor_aptidao = matrizes_ordenadas.get(matriz)['soma_valor_aptidao']
    probabilidade_selecao = (valor_aptidao / soma_valor_aptidao)
    probabilidade_selecao_norm = round(probabilidade_selecao * 100, 1)
    # ---
    # alimentacao dos resultados do dicionario das matrizes
    matrizes_ordenadas.get(matriz).update({'probabilidade_selecao': probabilidade_selecao})
    matrizes_ordenadas.get(matriz).update({'probabilidade_selecao_norm': probabilidade_selecao_norm})

# ------------------------------------------------------------
# 3.1.2 CALCULO DO NUMERO ESPERADO DE REPRODUCOES
# ------------------------------------------------------------

for matriz in matrizes_ordenadas:
    valor_aptidao = matrizes_ordenadas.get(matriz)['valor_aptidao']
    media_valor_aptidao = matrizes_ordenadas.get(matriz)['media_valor_aptidao']
    num_esperado_reproducao = (valor_aptidao / media_valor_aptidao)
    num_esperado_reproducao_norm = round(num_esperado_reproducao, 2)
    # ---
    # alimentacao dos resultados do dicionario das matrizes
    matrizes_ordenadas.get(matriz).update({'numero_esperado_reproducao_norm':
```

```
        num_esperado_reproducao_norm})

# ————————————————————————————————————————————
# 3.1.3 SORTEIO DAS MATRIZES PARA REPRODUCAO
# ————————————————————————————————————————————

lista_ids_matrizes = [matriz for matriz in matrizes_ordenadas]
probabilidades_reproducao = [matrizes_ordenadas.get(matriz)['probabilidade_selecao'] for matriz in
    matrizes_ordenadas]

# sorteio das matrizes para reproducao
matrizes_selecionadas_reproducao = list(rnd.choice(lista_ids_matrizes, size=len(lista_ids_matrizes),
    p=probabilidades_reproducao))

for matriz in matrizes_ordenadas:
    selecao_roleta = dict(Counter(matrizes_selecionadas_reproducao)).get(matriz)
    # ———
    # alimentacao dos resultados do dicionario das matrizes
    matrizes_ordenadas[matriz].update({'numero_de_reproducoes': 0 if not selecao_roleta else
        selecao_roleta})

# ————————————————————————————————————————————
# 3.1.4 CALCULO DOS MINIMOS, MEDIOS, SOMAS E MAXIMOS DAS MATRIZES
# ————————————————————————————————————————————

lista_ids_matrizes = [id_matriz for id_matriz in matrizes_ordenadas.keys()]

for matriz in matrizes_ordenadas:
    min_probabilidade_selecao = min([matrizes_ordenadas.get(_matriz)['probabilidade_selecao'] for
        _matriz in lista_ids_matrizes])
    media_probabilidade_selecao = mean([matrizes_ordenadas.get(_matriz)['probabilidade_selecao'] for
        _matriz in lista_ids_matrizes])
    soma_probabilidade_selecao = sum([matrizes_ordenadas.get(_matriz)['probabilidade_selecao'] for
        _matriz in lista_ids_matrizes])
    max_probabilidade_selecao = max([matrizes_ordenadas.get(_matriz)['probabilidade_selecao'] for
        _matriz in lista_ids_matrizes])
    # ———
    # alimentacao dos resultados do dicionario das matrizes
    matrizes_ordenadas.get(matriz).update({'min_probabilidade_selecao': min_probabilidade_selecao})
    matrizes_ordenadas.get(matriz).update({'media_probabilidade_selecao': media_probabilidade_selecao})
    matrizes_ordenadas.get(matriz).update({'soma_probabilidade_selecao': soma_probabilidade_selecao})
    matrizes_ordenadas.get(matriz).update({'max_probabilidade_selecao': max_probabilidade_selecao})

for matriz in matrizes_ordenadas:
    min_numero_esperado_reproducao_norm =
        min([matrizes_ordenadas.get(_matriz)['numero_esperado_reproducao_norm'] for _matriz in
        lista_ids_matrizes])
    media_numero_esperado_reproducao_norm =
        mean([matrizes_ordenadas.get(_matriz)['numero_esperado_reproducao_norm'] for _matriz in
        lista_ids_matrizes])
    soma_numero_esperado_reproducao_norm =
        sum([matrizes_ordenadas.get(_matriz)['numero_esperado_reproducao_norm'] for _matriz in
        lista_ids_matrizes])
    max_numero_esperado_reproducao_norm =
        max([matrizes_ordenadas.get(_matriz)['numero_esperado_reproducao_norm'] for _matriz in
        lista_ids_matrizes])
    # ———
    # alimentacao dos resultados do dicionario das matrizes
    matrizes_ordenadas.get(matriz).update({'min_numero_esperado_reproducao_norm':
        min_numero_esperado_reproducao_norm})
    matrizes_ordenadas.get(matriz).update({'media_numero_esperado_reproducao_norm':
        media_numero_esperado_reproducao_norm})
    matrizes_ordenadas.get(matriz).update({'soma_numero_esperado_reproducao_norm':
        soma_numero_esperado_reproducao_norm})
```

```
        matrizes_ordenadas.get(matriz).update({'max_numero_esperado_reproducao_norm':
            max_numero_esperado_reproducao_norm})

    for matriz in matrizes_ordenadas:
        min_numero_de_reproducoes = min([matrizes_ordenadas.get(_matriz)['numero_de_reproducoes'] for
            _matriz in lista_ids_matrizes])
        media_numero_de_reproducoes = mean([matrizes_ordenadas.get(_matriz)['numero_de_reproducoes']
            for _matriz in lista_ids_matrizes])
        soma_numero_de_reproducoes = sum([matrizes_ordenadas.get(_matriz)['numero_de_reproducoes'] for
            _matriz in lista_ids_matrizes])
        max_numero_de_reproducoes = max([matrizes_ordenadas.get(_matriz)['numero_de_reproducoes'] for
            _matriz in lista_ids_matrizes])
        # ---
        # alimentacao dos resultados do dicionario das matrizes
        matrizes_ordenadas.get(matriz).update({'min_numero_de_reproducoes': min_numero_de_reproducoes})
        matrizes_ordenadas.get(matriz).update({'media_numero_de_reproducoes':
            media_numero_de_reproducoes})
        matrizes_ordenadas.get(matriz).update({'soma_numero_de_reproducoes':
            soma_numero_de_reproducoes})
        matrizes_ordenadas.get(matriz).update({'max_numero_de_reproducoes':
            max_numero_de_reproducoes})

    # ------------------------------------------------------------
    # 3.2 CRUZAMENTO
    # ------------------------------------------------------------

    # ------------------------------------------------------------
    # 3.2.1 FORMACAO DOS PARES NO RESERVATORIO DE CRUZAMENTO
    # ------------------------------------------------------------

    # construcao de nova lista com as matrizes selecionadas para reproducao
    lista_matrizes_selecionadas_reproducao = []
    for matriz in matrizes_ordenadas:
        num_reproducao = matrizes_ordenadas.get(matriz)['numero_de_reproducoes']
        if num_reproducao != 0:
            lista_matrizes_reproducao = [matriz] * num_reproducao
            for matriz_reproducao in lista_matrizes_reproducao:
                lista_matrizes_selecionadas_reproducao.append(matriz_reproducao)

    # construcao do reservatorio de cruzamento com as matrizes reproduzidas
    matrizes_reservatorio_cruzamento = {}
    for id_matriz, matriz in enumerate(lista_matrizes_selecionadas_reproducao):
        id_matriz_reservatorio = f'id_matriz_reservatorio_acasalamento_{id_matriz}'
        # envio das informacoes das matrizes reproduzidas para um novo dicionario
        matrizes_reservatorio_cruzamento[id_matriz_reservatorio] = {
            'id_matriz': matriz,
            'matriz': matrizes_ordenadas.get(matriz)['matriz'],
            'lista_valor_alelo': matrizes_ordenadas.get(matriz)['lista_valor_alelo']
        }

    # lista contendo as matrizes sorteadas, de forma excludente, para cruzamento
    matrizes_selecionadas_cruzamento = []
    for matriz in matrizes_reservatorio_cruzamento:
        lista_matrizes = [
            id_matriz for id_matriz
            in matrizes_reservatorio_cruzamento
            if id_matriz != matriz
            and id_matriz not in matrizes_selecionadas_cruzamento
        ]
        # sorteio das matrizes e construcao da lista de pares
        if lista_matrizes:
            probabilidades = [1 / len(lista_matrizes) for _ in lista_matrizes]
            matriz_sorteada = rnd.choice(lista_matrizes, size=1, p=probabilidades)[0]
            matrizes_selecionadas_cruzamento.append(matriz_sorteada)
```

```
# novo dicionario com as matrizes do reservatorio de cruzamento formadas
pares_reservatorio_cruzamento = {}
i = 0
for id_par in range(int(len(matrizes_selecionadas_cruzamento) / 2)):
    par = matrizes_selecionadas_cruzamento[i:i+2]
    pares_reservatorio_cruzamento[f'par_{id_par}'] = {
        'ids_matrizes_pares': par,
        'par_com_matriz_nao_selecionada': False
    }
    i += 2

# no caso do numero de matrizes no reservatorio de cruzamento ser imprar,
# uma matriz ficara de fora do cruzamento. Para este caso, caso seja
# necessario considera—la, esta matriz nao selecionada formara par com
# uma matriz da lista de matrizes que ja possuem par
if CONSIDERAR_MATRIZ_SEM_PAR == 1:
    matrizes_selecionadas = []
    for par in pares_reservatorio_cruzamento:
        par_matrizes = pares_reservatorio_cruzamento.get(par)['ids_matrizes_pares']
        for matriz in par_matrizes:
            matrizes_selecionadas.append(matriz)
    # ———
    matriz_nao_selecionada = list(set(matrizes_reservatorio_cruzamento) - set(matrizes_selecionadas))
    # ———
    if matriz_nao_selecionada:
        probabilidades = [1 / len(matrizes_selecionadas) for _ in matrizes_selecionadas]
        # sorteio de matriz dentre as que ja foram sorteadas anteriormente
        matriz_sorteada = rnd.choice(matrizes_selecionadas, size=1, p=probabilidades)[0]
        num_par = len(pares_reservatorio_cruzamento)
        # alimentacao dos resultados do dicionario das matrizes
        pares_reservatorio_cruzamento.update(
            {
                f'par_{num_par}':
                    {
                        'ids_matrizes_pares': [matriz_nao_selecionada[0], matriz_sorteada],
                        'par_com_matriz_nao_selecionada': True
                        }
                }
                )

# alimentacao do dicionario de matrizes pares
for par in pares_reservatorio_cruzamento:
    par_matrizes = pares_reservatorio_cruzamento.get(par)['ids_matrizes_pares']
    id_matriz_1, id_matriz_2 = par_matrizes[0], par_matrizes[1]
    # ———
    matriz_1 = matrizes_reservatorio_cruzamento.get(id_matriz_1)['matriz']
    matriz_2 = matrizes_reservatorio_cruzamento.get(id_matriz_2)['matriz']
    # ———
    lista_valor_alelo_1 = matrizes_reservatorio_cruzamento.get(id_matriz_1)['lista_valor_alelo']
    lista_valor_alelo_2 = matrizes_reservatorio_cruzamento.get(id_matriz_2)['lista_valor_alelo']
    # alimentacao dos resultados do dicionario das matrizes
    pares_reservatorio_cruzamento.get(par).update({
        'matrizes_pares': [matriz_1, matriz_2],
        'lista_valor_alelo_matrizes_pares': [lista_valor_alelo_1, lista_valor_alelo_2]
    })

# ————————————————————————————————————————————
# 3.2.2 SORTEIO DOS PONTOS DE CRUZAMENTO
# ————————————————————————————————————————————

# escolha aleatoria do ponto de corte para cruzamento entre os pares
for par in pares_reservatorio_cruzamento:
    matrizes_pares = pares_reservatorio_cruzamento.get(par)
```

```
    matriz_1, matriz_2 = matrizes_pares['matrizes_pares'][0], matrizes_pares['matrizes_pares'][1]
    lista_valor_alelo_1, lista_valor_alelo_2 = matrizes_pares['lista_valor_alelo_matrizes_pares'][0],
        matrizes_pares['lista_valor_alelo_matrizes_pares'][1]
    tamanho_matriz = int((len(matriz_1) + len(matriz_2)) / 2) - 1
    # sorteio do ponto de cruzamento para troca de informacoes
    pontos_corte = [ponto+1 for ponto in range(tamanho_matriz)]
    probabilidades = [1 / tamanho_matriz for _ in range(tamanho_matriz)]
    ponto_sorteado = rnd.choice(pontos_corte, size=1, p=probabilidades)[0]
    # ---
    # inclusao dos pontos de cruzamento e ponto de cruzamento sorteado
    lista_valor_alelo_1, lista_valor_alelo_2 = [str(alelo) for alelo in lista_valor_alelo_1], [str(alelo) for alelo
        in lista_valor_alelo_2]
    lista_alelo_ponto_corte_1 = '.'.join(lista_valor_alelo_1[:ponto_sorteado]) + '|' +
        '.'.join(lista_valor_alelo_1[ponto_sorteado:])
    lista_alelo_ponto_corte_2 = '.'.join(lista_valor_alelo_2[:ponto_sorteado]) + '|' +
        '.'.join(lista_valor_alelo_2[ponto_sorteado:])
    # alimentacao dos resultados do dicionario das matrizes
    pares_reservatorio_cruzamento.get(par).update({
        'ponto_corte_cruzamento': ponto_sorteado,
        'ponto_corte_cruzamento_matrizes_pares': [lista_alelo_ponto_corte_1, lista_alelo_ponto_corte_2]
    })

# ----------------------------------------------------------
# 3.2.3 CRIACAO DAS NOVAS MATRIZES
# ----------------------------------------------------------

# troca de informaoess entre as matrizes pares
for par in pares_reservatorio_cruzamento:
    matrizes_pares = pares_reservatorio_cruzamento.get(par)
    ponto_cruzamento = matrizes_pares['ponto_corte_cruzamento']
    matriz_1, matriz_2 = matrizes_pares['matrizes_pares'][0], matrizes_pares['matrizes_pares'][1]
    # cruzamento das matrizes
    matriz_cruzada_1, matriz_cruzada_2 = matriz_1[:ponto_cruzamento] + matriz_2[ponto_cruzamento:],
        matriz_2[:ponto_cruzamento] + matriz_1[ponto_cruzamento:]
    # alimentacao dos resultados do dicionario das matrizes
    pares_reservatorio_cruzamento.get(par).update({
        'matrizes_pares_cruzadas': [matriz_cruzada_1, matriz_cruzada_2]
    })

# ---

# criacao de lista temporaria de dicionarios com as infos das matrizes pares
lista_dict_temp = []
for par in pares_reservatorio_cruzamento:
    matrizes_pares = pares_reservatorio_cruzamento.get(par)['ids_matrizes_pares']
    for matriz in matrizes_pares:
        infos_dict = {**{'id_par': par}, **pares_reservatorio_cruzamento.get(par)}
        lista_dict_temp.append(infos_dict)

# dicionario com as informacoes das matrizes apos o cruzamento
matrizes_cruzadas_reservatorio_cruzamento = {}
for id_matriz_cruzada, matriz_cruzada in enumerate(lista_dict_temp):
    id_matriz_cruzada_reservatorio = f'id_matriz_cruzada_reservatorio_{id_matriz_cruzada}'
    # novo id da matriz apos o cruzamento
    matrizes_cruzadas_reservatorio_cruzamento[id_matriz_cruzada_reservatorio] = {
        'informacao_pares': matriz_cruzada
    }
    # alimentacao do dicionario com as informacoes da matriz respectiva
    # se id_matriz_par for par, entao pegar o primeiro valor das listas
    # se id_matriz_par for impar, entao pegar o segundo valor das listas
    if (id_matriz_cruzada % 2) == 0: # par
        matrizes_cruzadas_reservatorio_cruzamento.get(id_matriz_cruzada_reservatorio).update({
            'id_matriz_pai': matriz_cruzada.get('ids_matrizes_pares')[0],
            'matriz_pai': matriz_cruzada.get('matrizes_pares')[0],
```

```
            'lista_valor_alelo_matriz_pai': matriz_cruzada.get('lista_valor_alelo_matrizes_pares')[0],
            'ponto_cruzamento': matriz_cruzada.get('ponto_corte_cruzamento'),
            'ponto_corte_cruzamento_matriz_pai':
                matriz_cruzada.get('ponto_corte_cruzamento_matrizes_pares')[0],
            'matriz_filha': matriz_cruzada.get('matrizes_pares_cruzadas')[0],
            'lista_valor_alelo_matriz_filha': [alelo for alelo in
                matriz_cruzada.get('matrizes_pares_cruzadas')[0]]
        })
    else: # impar
        matrizes_cruzadas_reservatorio_cruzamento.get(id_matriz_cruzada_reservatorio).update({
            'id_matriz_pai': matriz_cruzada.get('ids_matrizes_pares')[1],
            'matriz_pai': matriz_cruzada.get('matrizes_pares')[1],
            'lista_valor_alelo_matriz_pai': matriz_cruzada.get('lista_valor_alelo_matrizes_pares')[1],
            'ponto_cruzamento': matriz_cruzada.get('ponto_corte_cruzamento'),
            'ponto_corte_cruzamento_matriz_pai':
                matriz_cruzada.get('ponto_corte_cruzamento_matrizes_pares')[1],
            'matriz_filha': matriz_cruzada.get('matrizes_pares_cruzadas')[1],
            'lista_valor_alelo_matriz_filha': [alelo for alelo in
                matriz_cruzada.get('matrizes_pares_cruzadas')[1]]
        })

# ------------------------------------------------------------
# 3.3 MUTACAO
# ------------------------------------------------------------

# aplicacao do operador de mutacao
for matriz in matrizes_cruzadas_reservatorio_cruzamento:
    alelos_matriz_filha =
        matrizes_cruzadas_reservatorio_cruzamento.get(matriz)['lista_valor_alelo_matriz_filha']
    # lista com os alelos apos o operador de mutacao
    alelos_apos_mutacao = []
    for alelo in alelos_matriz_filha:
        if int(alelo) == 0:
            alelos = [0, 1] # possibilidades
            probabilidades = [1-PROBABILIDADE_MUTACAO, PROBABILIDADE_MUTACAO] #
                probabilidade de nao mutacao e mutacao
            alelo_apos_operador = rnd.choice(alelos, size=1, p=probabilidades)[0]
            alelos_apos_mutacao.append(alelo_apos_operador)
        else:
            alelos = [1, 0] # possibilidades
            probabilidades = [1-PROBABILIDADE_MUTACAO, PROBABILIDADE_MUTACAO] #
                probabilidade de nao mutacao e mutacao
            alelo_apos_operador = rnd.choice(alelos, size=1, p=probabilidades)[0]
            alelos_apos_mutacao.append(alelo_apos_operador)
    # ---
    alelos_apos_mutacao_str = [str(alelo) for alelo in alelos_apos_mutacao]
    alelos_matriz_filha =
        matrizes_cruzadas_reservatorio_cruzamento.get(matriz)['lista_valor_alelo_matriz_filha']
    mutacoes_esperadas = N * NUM_CARACTERISTICAS * PROBABILIDADE_MUTACAO
    # alimentacao dos resultados do dicionario das matrizes
    matrizes_cruzadas_reservatorio_cruzamento.get(matriz).update({
        'matriz_apos_operador_mutacao': ''.join(alelos_apos_mutacao_str),
        'lista_valor_alelo_matriz_apos_operador_mutacao': alelos_apos_mutacao_str,
        'houve_mutacao': 0 if alelos_apos_mutacao_str == alelos_matriz_filha else 1, # 0 se nao houve
            mutacao; 1 se houve;
        'num_esperado_mutacoes': mutacoes_esperadas
    })

# ------------------------------------------------------------
# ------------------------------------------------------------
# ALIMENTACAO DAS BASES FINAIS
# ------------------------------------------------------------
# ------------------------------------------------------------
```

```
    id_geracao = f'id_geracao_{CONTADOR_GERACAO}'

    BASE_POPULACOES_PRE_OPERADORES[id_geracao] = matrizes_ordenadas
    BASE_POPULACOES_POS_OPERADORES[id_geracao] = matrizes_cruzadas_reservatorio_cruzamento

    print(f'Fim: Populacao Inicial/Geracao {CONTADOR_GERACAO+1}' if CONTADOR_GERACAO ==
        0 else f'Fim: Geracao {CONTADOR_GERACAO+1}')

    # ----------------------------------------------------------
    # ----------------------------------------------------------
    # ALIMENTACAO DOS BINARIOS DOS CRITERIOS DE PARADA
    # ----------------------------------------------------------
    # ----------------------------------------------------------

    # binario_num_max_geracoes
    if (CONTADOR_GERACAO+1 == NUM_MAX_GERACOES):
        binario_num_max_geracoes = True

    # binario_valor_mais_aptidao_alcancado
    for matriz in matrizes_ordenadas:
        valor_aptidao = matrizes_ordenadas.get(matriz)['valor_aptidao']
        max_valor_aptidao_possivel = matrizes_ordenadas.get(matriz)['max_valor_aptidao_possivel']
        if (valor_aptidao == max_valor_aptidao_possivel):
            binario_valor_mais_aptidao_alcancado = True

    # binario_valor_aptidao_corrente_menor_que_anterior
    if CONTADOR_GERACAO >= 1:
        soma_valor_aptidao_geracao_corrente = []
        soma_valor_aptidao_geracao_anterior = []
        for geracao in BASE_POPULACOES_PRE_OPERADORES:
            geracao_corrente =
                BASE_POPULACOES_PRE_OPERADORES.get(f'id_geracao_{CONTADOR_GERACAO}')
            geracao_anterior =
                BASE_POPULACOES_PRE_OPERADORES.get(f'id_geracao_{CONTADOR_GERACAO-1}')
            lista_soma_valor_aptidao_geracao_corrente = []
            lista_soma_valor_aptidao_geracao_anterior = []
            for matriz in geracao_corrente:
                lista_soma_valor_aptidao_geracao_corrente.append(geracao_corrente.get(matriz)['soma_valor_aptidao'])
            for matriz in geracao_anterior:
                lista_soma_valor_aptidao_geracao_anterior.append(geracao_anterior.get(matriz)['soma_valor_aptidao'])
            soma_valor_aptidao_geracao_corrente.append(mean(lista_soma_valor_aptidao_geracao_corrente))
            soma_valor_aptidao_geracao_anterior.append(mean(lista_soma_valor_aptidao_geracao_anterior))
        # ---
        if (soma_valor_aptidao_geracao_corrente < soma_valor_aptidao_geracao_anterior):
            binario_valor_aptidao_corrente_menor_que_anterior = True

    CONTADOR_GERACAO += 1

# ----------------------------------------------------------
# ----------------------------------------------------------
# SALVAMENTO DAS BASES FINAIS (JSON)
# ----------------------------------------------------------
# ----------------------------------------------------------

nome_arquivos = 'informacoes_ag_simples'

try:
    Path('data').mkdir(exist_ok=False)
    pd.DataFrame(BASE_POPULACOES_PRE_OPERADORES).to_json(f'data/{nome_arquivos}_pre_operadores.json',
        indent=4)
    pd.DataFrame(BASE_POPULACOES_POS_OPERADORES).to_json(f'data/{nome_arquivos}_pos_operadores.json',
        indent=4)
except OSError:
    print('Nao foi possivel salvar os arquivos.')
```