MARÍLIA GOUVEIA RODRIGUES COSTA

ALGORITMO EVOLUTIVO APLICADO À OTIMIZAÇÃO DA RESOLUÇÃO DE SUDOKU

RECIFE

2025

MARÍLIA GOUVEIA RODRIGUES COSTA

ALGORITMO EVOLUTIVO APLICADO À OTIMIZAÇÃO DA RESOLUÇÃO DE SUDOKU

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Cícero Garrozi

RECIFE

2025

Dados Internacionais de Catalogação na Publicação
Sistema Integrado de Bibliotecas da UFRPE
Bibliotecário(a): Lorena Teles – CRB-4 1774

C838a Costa, Marília Gouveia Rodrigues.
Algoritmo evolutivo aplicado à otimização da resolução de Sudoku / Marília Gouveia Rodrigues Costa. – Recife, 2025.
59 f.; il.

Orientador(a): Cícero Garrozi.

Trabalho de Conclusão de Curso (Graduação) – Universidade Federal Rural de Pernambuco, Bacharelado em Sistemas da Informação, Recife, BR-PE, 2025.

Inclui referências.

1. Sudoku. 2. Algorítmos genéticos. 3. Abordagem híbrida. 4. Métodos determinísticos 5. Programação heurística. I. Garrozi, Cícero, orient. II. Título

CDD 004

MARÍLIA GOUVEIA RODRIGUES COSTA

ALGORITMO EVOLUTIVO APLICADO À OTIMIZAÇÃO DA RESOLUÇÃO DE SUDOKU

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado em: 07/08/2025

BANCA EXAMINADORA

Cícero Garrozi (Orientador)
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Aluizio Fausto Ribeiro Araújo
Centro de Informática
Universidade Federal de Pernambuco

AGRADECIMENTOS

Em primeiro lugar, agradeço a minha família, que me apoiou durante toda a minha trajetória acadêmica, em especial, a meus pais que sempre se fizeram presentes e nunca pouparam esforços para me ajudar. Agradeço a todos os meus colegas e amigos com os quais compartilhei bons momentos e sempre me motivaram e incentivaram. Por último, obrigada a todo o corpo docente do Bacharelado em Sistemas de Informação da UFRPE por todo ensinamento e conhecimento compartilhados, principalmente, a meu orientador, Prof. Cícero Garrozi, que me direcionou ao longo da realização deste trabalho de conclusão. Todos foram primordiais para o meu desenvolvimento e formação.

RESUMO

O presente trabalho propõe um algoritmo híbrido para a resolução de quebra-cabeças Sudoku que combina estratégias lógicas, como: *Naked Singles*, *Hidden Singles*, *Naked Pairs* e *X-Wings* com um Algoritmo Genético (AG) para otimizar a busca por soluções. A motivação para esse estudo encontra-se na elevada complexidade computacional do Sudoku, que o torna um objetivo de pesquisa relevante para validação de novos algoritmos que, futuramente, podem ser aplicados para resolução de problemas da vida real. A abordagem híbrida sugerida demonstrou ser significativamente mais eficaz do que apenas o AG, alcançando uma taxa de sucesso de 94,56% em um conjunto de 46 Sudokus de diferentes níveis e características nas 12 rodadas de teste realizadas, o que representa um aumento de cerca de 28,8% em comparação com a taxa de 65,76% alcançada apenas pelo AG. A etapa de pré-processamento, em que as técnicas de resolução são aplicadas, resolveu de forma autônoma 73,91% dos *puzzles* testados e, nos casos mais complexos, reduziu consideravelmente o espaço de busca e acelerou a convergência do AG. Nesse contexto, o trabalho concluiu que a integração de métodos determinísticos com meta-heurísticas representa uma estratégia robusta para problemas de otimização combinatória. Desse modo, as principais contribuições deste projeto incluem o modelo híbrido proposto, a análise empírica de seu desempenho e a disponibilização do código-fonte.

Palavras-chave: sudoku**;** algoritmo genético; abordagem híbrida; métodos determinísticos; metaheurística.

ABSTRACT

This study proposes a hybrid algorithm for solving Sudoku puzzles that combines logical strategies such as Naked Singles, Hidden Singles, Naked Pairs, and X-Wings with a Genetic Algorithm (GA) to optimize the search for solutions. The motivation for this research lies in the high computational complexity of Sudoku, which makes it a relevant research target for validating new algorithms that may later be applied to real-world problem-solving. The proposed hybrid approach proved to be significantly more effective than the GA alone, achieving a success rate of 94.56% in a set of 46 Sudokus of varying levels and characteristics over 12 test rounds, representing an increase of approximately 28.8% compared to the 65.76% rate achieved by the GA alone. The preprocessing stage, in which the solving techniques are applied, autonomously solved 73.91% of the tested puzzles and, in more complex cases, significantly reduced the search space and accelerated the GA's convergence. In this context, the study concluded that integrating deterministic methods with metaheuristics represents a robust strategy for combinatorial optimization problems. Thus, the main contributions of this project include the proposed hybrid model, the empirical analysis of its performance, and the availability of its source code.

Keywords: sudoku; genetic algorithm; hybrid approach; deterministic methods; metaheuristic.

SUMÁRIO

1. INTRODUÇÃO

Esta seção tem como objetivo apresentar o tema abordado, introduzindo o Sudoku sob uma perspectiva computacional e destacando as razões que justificam sua escolha como objeto de pesquisa. Para isso, são explorados o contexto em que o problema está inserido, os fatores que motivaram o desenvolvimento deste trabalho, os objetivos estabelecidos e as contribuições esperadas. Por fim, também é apresentada a organização adotada ao longo do texto, com intuito de orientar a leitura das seções seguintes.

1.1. CONTEXTO

Apesar de ser amplamente reconhecido como um passatempo lógico, o Sudoku, desde a sua popularização, foi além da sua origem recreativa e despertou um interesse considerável na comunidade científica. Essa atenção se deve especialmente ao contraste entre a simplicidade de suas regras e a complexidade computacional inerente à sua resolução, que o estabeleceu como um problema desafiador e um objeto de estudo relevante em várias áreas.

No campo da matemática, por exemplo, a investigação de seus fundamentos levou a descobertas consideráveis, como a quantificação de seu universo em $6,671\ x\ 10^{21}$ quebra-cabeças distintos (1). Outro resultado notável foi o estabelecimento do número mínimo de dicas iniciais para que um *puzzle* possua uma solução única (2).

Por sua vez, na ciência da computação, a complexidade do Sudoku serviu de estímulo para o desenvolvimento de uma vasta gama de algoritmos projetados para resolver, classificar e gerar os quebra-cabeças (3). Esse esforço resultou na exploração de diferentes abordagens que vão desde métodos de busca convencionais, como o *depth-first search* (DFS) (4), *breadth-first search* (BFS), *backtracking* (4, 5) e o *Algorithm X* utilizando a técnica *Dancing Links* (6), até o uso de metaheurísticas como Algoritmos Culturais (7), Otimização por Colônia de Formigas (8) e Algoritmos Genéticos (3, 8), incluindo propostas que unem essas duas últimas (9). Ampliando a discussão sobre abordagens combinadas, o trabalho de Amil e Mantere (13), mostra-se particularmente relevante, pois introduz e define o termo pré-processamento, que também será adotado no presente trabalho, para descrever uma etapa inicial na qual técnicas de resolução lógica são usadas para simplificar o quebra-cabeça antes da aplicação de um Algoritmo Evolutivo.

Desse modo, a justificativa para persistir no estudo do Sudoku, mesmo diante dos diversos trabalhos desenvolvidos sobre o tema, baseia-se em três pilares fundamentais. O primeiro é seu desafio teórico como um problema NP-completo (10); o segundo, seu papel como *benchmark* para o projeto e a validação de novos algoritmos (10); e o terceiro, seu potencial de aplicações práticas em problemas reais (4), que vão além da esfera acadêmica. Estes pilares, que serão aprofundados na subseção seguinte, demonstram a importância de continuar desenvolvendo e aprimorando as pesquisas relacionadas a esse problema.

1.2. MOTIVAÇÃO

A motivação para esse trabalho fundamenta-se na dupla natureza do Sudoku: um problema de alta complexidade computacional, e, ao mesmo tempo, uma ferramenta com grande potencial para o avanço algorítmico e tecnológico. Na sequência, são aprofundados os três pilares que justificam esta pesquisa: sua profundidade teórica, utilidade como *benchmark* e aplicações no mundo real.

1.2.1. Relevância Teórica e Desafio Computacional

A base da importância do Sudoku no meio acadêmico reside em sua classificação como um problema NP-completo (10), pertencente a uma das classes de problemas mais difíceis da ciência da computação (3). Essa propriedade foi demonstrada pela primeira vez em (11) através de uma redução do problema de complementação do Quadrado Latino (Latin Square Completion) (8). De modo semelhante, o quebra-cabeça também pode ser reduzido a um problema SAT (12), comprovando igualmente sua NP-completude.

Essa característica implica que não existem algoritmos conhecidos capazes de resolver qualquer instância do problema de forma rápida e garantidamente eficiente, o que o torna um objeto de estudo relevante. De fato, já em 2006, no auge da popularização do Sudoku, Aaronson (10) sugeriu que a fama do jogo poderia, inclusive, levar a avanços na ciência da computação.

1.2.2. Sudoku como *Benchmark*

O fato de o Sudoku ser um problema complexo com regras claras e bem definidas, o transforma em um bom laboratório para o projeto de novos algoritmos (10) e um problema *benchmark* ideal (8), considerando que em contraste com os problemas do mundo real, cujas

variáveis e restrições podem ser ambíguas ou incertas, o Sudoku oferece um ambiente determinístico e controlado.

Essa característica permite que o desempenho de diferentes algoritmos e suas versões seja avaliado de forma minuciosa, tornando a comparação entre eles mais precisa e confiável, do que em cenários de maior incerteza, como os dos problemas do mundo real. Isso significa que algoritmos validados em um ambiente controlado como o do Sudoku podem ser aplicados a outros desafios de otimização combinatória, a exemplo dos problemas de carregamento uniforme de veículos (balanceamento técnico) (13). Sendo assim, a criação de algoritmos eficientes para este quebra-cabeça não é apenas um exercício teórico, mas um desafio representativo para a área de otimização combinatória.

1.2.3. Aplicações no Mundo Real

Além do seu valor teórico, as pesquisas relacionadas ao Sudoku e os algoritmos desenvolvidos para sua solução possuem aplicações práticas e diretas em diferentes áreas, visto que sua estrutura lógica e características servem como ferramenta nas seguintes áreas:

- Educação: O Sudoku é amplamente utilizado como ferramenta pedagógica para o desenvolvimento do raciocínio lógico e de habilidades de resolução de problemas (14);

- Segurança da Informação: A estrutura do Sudoku é utilizada como base em modelos de segurança. Em uma técnica de esteganografia de vídeo proposta em 2019 (15), por exemplo, a grade do quebra-cabeça funciona como uma chave para ocultar dados, previamente criptografados, nos quadros de um vídeo.

- Engenharia: A lógica do Sudoku pode ser aplicada para otimização do arranjo de painéis fotovoltaicos, visando reduzir os efeitos do sombreamento mútuo (16), bem como no design de formas de onda de radar (17).

1.3. OBJETIVOS

Este projeto tem como objetivo principal: desenvolver e avaliar uma implementação com abordagem híbrida para solução de Sudoku, que combina estratégias de resolução com Algoritmo Genético, de modo a construir um solucionador capaz de lidar com quebra-cabeças de níveis médio e difícil.

Para alcançar o objetivo geral definido, foram estabelecidos os seguintes objetivos específicos:

a) Identificar e selecionar estratégias clássicas de resolução de Sudoku, a fim de implementá-las na etapa de pré-processamento;

b) Projetar e desenvolver um Algoritmo Genético adaptado para a resolução de Sudoku;

c) Unir o módulo de pré-processamento ao AG, de maneira que as deduções lógicas aplicadas nessa etapa restrinjam o problema a ser resolvido pela metaheurística;

d) Criar uma interface visual que apresenta os resultados obtidos nas etapas de pré-processamento e de resolução com AG;

e) Avaliar os impactos da abordagem híbrida na otimização da resolução do Sudoku, destacando a contribuição individual das estratégias de resolução e do Algoritmo Genético;

f) Comparar os resultados obtidos com as soluções propostas em trabalhos semelhantes.

1.4. CONTRIBUIÇÃO

Mesmo diante da elevada complexidade computacional do problema abordado e da vasta literatura dedicada à sua resolução, este trabalho apresenta contribuições que podem agregar ao campo do Sudoku e das técnicas de otimização combinatória, conforme detalhado a seguir:

- Desenvolvimento de um Modelo Algorítmico Híbrido: A principal contribuição deste projeto é a concepção e implementação de uma abordagem híbrida que integra técnicas determinísticas com meta-heurística. Especificamente, este modelo combina a precisão das estratégias de resolução lógica do Sudoku com a adaptabilidade dos Algoritmos Genéticos para buscar a solução dos quebra-cabeças;

- Refinamento do Espaço de Busca: Este trabalho apresenta uma metodologia para reduzir a complexidade do problema por meio da diminuição do espaço de busca, realizada antes da aplicação do algoritmo evolutivo. Isso ocorre através do preenchimento de células “óbvias” e da eliminação de candidatos impossíveis mediante a aplicação de estratégias lógicas de resolução. Além disso, a inclusão desse conhecimento específico do problema potencializa a solução (7), acelerando e, em

alguns casos, possibilitando a obtenção do resultado, conforme será evidenciado nas seções futuras;

- Análise Comparativa de Desempenho: A pesquisa expõe uma análise empírica que quantifica os ganhos decorrentes da combinação das abordagens apresentadas. Essa avaliação considera a solução proposta aplicada à quebra-cabeças de diferentes níveis de dificuldade e realiza uma comparação direta com uma implementação baseada apenas em Algoritmo Genético. Isso permite isolar e mensurar o impacto da etapa de pré-processamento, contribuindo para o avanço do conhecimento na área;

- Disponibilização do Algoritmo em Código Aberto: O algoritmo híbrido proposto encontra-se disponível em código-fonte aberto[1], podendo ser utilizado como base para a reprodutibilidade dos experimentos descritos, assim como para a extensão e adaptação por parte de outros pesquisadores em investigações futuras.

- Visualização do Processo através de Interface Gráfica: O projeto também contribui com o desenvolvimento de uma ferramenta com interface gráfica, que possibilita a visualização do processo de resolução e torna o funcionamento complexo do algoritmo mais compreensível;

1.5. ORGANIZAÇÃO DO TRABALHO

Com o propósito de apresentar a pesquisa de forma clara e objetiva, este trabalho foi estruturado em seis seções. A Seção 1, a presente Introdução, contextualiza o problema da resolução de quebra-cabeças Sudoku, detalha a motivação, os objetivos e as principais contribuições do projeto. Posteriormente, a Seção 2, Referencial Teórico, descreve os fundamentos necessários para a compreensão do trabalho, abordando as estratégias de resolução de Sudoku, os conceitos dos Algoritmos Genéticos e as técnicas de busca local *Hill Climbing e Min-Conflicts.* A Seção 3, Trabalhos Relacionados, realiza uma revisão da literatura, apresentando as abordagens propostas em estudos semelhantes. Em seguida, a Seção 4, Desenvolvimento, detalha a metodologia e a implementação da solução híbrida proposta. A Seção 5, Resultados, evidencia e analisa os dados experimentais obtidos, discutindo a performance e a eficiência da solução desenvolvida. Por fim, a Seção 6, Considerações Finais, apresenta as conclusões alcançadas, verifica o atendimento dos objetivos e sugere direções para pesquisas futuras.

[1] https://github.com/mariliagrcosta/evolutionary-sudoku-solver

2. REFERENCIAL TEÓRICO

A seção atual reúne os fundamentos teóricos que embasam a abordagem desenvolvida neste trabalho. Inicialmente, são abordados aspectos essenciais relacionados ao Sudoku, incluindo sua origem, regras e principais características. Em seguida, são discutidas algumas das principais estratégias determinísticas utilizadas na resolução do problema, que constituem a base da etapa de pré-processamento do sistema proposto. Por fim, são detalhados os princípios dos Algoritmos Genéticos e das técnicas de busca local *Hill Climbing* e *Min-Conflicts*, que fazem parte do núcleo da solução híbrida desenvolvida neste estudo.

2.1. SUDOKU

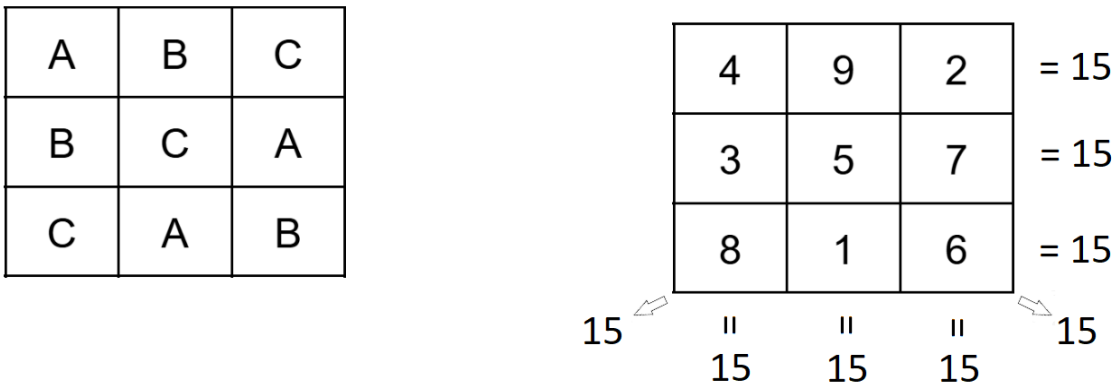
O Sudoku é um jogo lógico, amplamente conhecido em todo mundo, baseado no preenchimento de uma grade $n\ x\ n$, comumente $9\ x\ 9$, com números que devem obedecer a restrições específicas: cada valor deve aparecer apenas uma vez em cada linha, coluna e subgrade. Nesse contexto, a seguir são apresentados os aspectos fundamentais do *puzzle,* que embasam este estudo.

2.1.1. Origem e Fundamentos Matemáticos

Apesar da origem japonesa de seu nome, *SuDoku*, uma abreviação para "Suuji wa dokushin ni kagiru", que significa "os dígitos devem permanecer únicos", as bases desse quebra-cabeça são mais antigas e originárias da Europa. Seu conceito fundamental deriva dos Quadrados Latinos, ilustrados na Figura 1, estruturas que foram propostas no século XVIII pelo matemático suíço Leonhard Euler, que consistem em grades $n\ x\ n$ nas quais cada número de 1 até n aparece apenas uma vez em cada linha e coluna (13).

No final do século XIX, um precursor mais direto do Sudoku foi publicado por um jornal parisiense da época, o *Le Siècle*. Tratava-se de uma versão $9\ x\ 9$ parcialmente preenchida de um Quadrado Mágico que possuía sub-blocos $3\ x\ 3$. Um Quadrado Mágico, apresentado na Figura 1, corresponde a uma tabela quadrada preenchida com números, não necessariamente diferentes, em que a soma de cada linha e coluna resulta no mesmo valor (13). Sendo assim, o objetivo principal do jogo publicado pelo jornal era completar o quadrado de forma a satisfazer a essa característica principal.

Figura 1 - Exemplo de quadrado (esquerda) latino e quadrado mágico (direita)



Fonte: Autor (2025)

Foi somente em 1986 que o quebra-cabeça passou a ser chamado da maneira que é conhecido atualmente, quando a empresa japonesa *Nikoli, Inc.* começou a publicar a sua versão do *Sudoku* por sugestão do seu presidente, Maki Kaji, a quem se atribui o nome do jogo. A sua fama internacional, no entanto, apenas foi alcançada em 2005, após o jornal londrino *The Times* passar a publicá-lo como seu quebra-cabeça diário no ano anterior. Fato que se sucedeu devido ao esforço de Wayne Gould, que passou anos desenvolvendo um programa de computador capaz de gerar Sudokus (13).

2.1.2. Definição e Características

O Sudoku pode ser visto tanto como um problema de satisfação de restrições (*Constraint Satisfaction Problem* - CSP) (3), quanto um problema de otimização combinatória, que se baseia em lógica de posicionamento de números (18). Seu objetivo consiste em preencher uma grade $n \, x \, n$, tradicionalmente 9 x 9, de modo que os números de 1 a n apareçam uma única vez em cada linha, coluna e subgrade.

Considerando o caso clássico ($n = 9$), essas restrições implicam propriedades matemáticas claras: a soma dos números em qualquer linha ou coluna completa deve ser igual a 45, enquanto o produto dos números deve ser igual a 9!, isto é, 362.880.

Formalmente, o problema pode ser representado por uma matriz $X = [x_{ij}]$, em que x_{ij} representa o valor presente na célula da linha i e coluna j. De modo que a formulação pode ser expressa como a minimização do número total de violações da restrições:

$$Minimizar\colon f(X) = número\ de\ violações\ das\ restrições \quad (1)$$

Sujeito às seguintes condições:

1. Domínio dos valores: $x_{ij} \in \{1, 2, ..., n\}, \forall i, j$;
2. Unicidade em cada linha: $x_{ij} \neq x_{ik} \forall i, j \neq k$;
3. Unicidade em cada coluna: $x_{ij} \neq x_{kj} \forall j, i \neq k$;
4. Unicidade em cada subgrade $m\,x\,m$ (para $n = m^2$): $x_{ij} \neq x_{pq} \forall (i,j) \neq (p,q)$ no mesmo bloco;
5. Respeito às dicas (pistas iniciais): $x_{ij} = c_{ij}, se\ c_{ij} \neq 0$.

Assim, em um contexto de otimização, a função objetivo $f(x)$ representa o número total de conflitos (repetições) encontrados e atingir $f(x) = 0$ indica o encontro de uma solução válida. Desse modo, ainda que haja múltiplas ordens possíveis de preenchimento, a resolução consiste em encontrar uma configuração de x que satisfaça todas as restrições impostas pelas pistas fornecidas pelo quebra-cabeça.

2.1.3. Unicidade da Solução e Dificuldade

A classificação da dificuldade de um Sudoku é uma tarefa de alta complexidade, pois o número de dicas, por si só, não define o quão desafiador um quebra-cabeça é, visto que fatores como a simetria e o posicionamento desses números também são relevantes. De fato, um artigo (18) cita a existência de cerca de 15 a 20 fatores distintos que seriam capazes de influenciar o nível de complexidade de um *puzzle* para um solucionador humano. Sendo assim, quando uma revista, livro ou site indica a dificuldade de um Sudoku como "fácil", "médio" ou "difícil", não é possível ter certeza de quais critérios foram considerados para fazer essa classificação. Isso significa que dois quebra-cabeças definidos como de mesmo nível não possuem, necessariamente, o mesmo grau de complexidade.

Outra questão central no estudo do Sudoku é a unicidade das soluções. Sabe-se que a remoção de uma dica de um quebra-cabeça pode alterar sua natureza, gerando um novo problema para o qual múltiplas soluções podem existir (13). Essa característica levou à dúvida sobre o limite mínimo de pistas, o que culminou no estabelecimento de 17 dicas para se ter uma solução única (2). Essa distinção técnica é a raiz de um debate sobre a "validade" ou razoabilidade de um *puzzle*: já que embora seja comum encontrar artigos (4, 19) que considerarem válidos apenas os Sudokus com uma solução, do ponto de vista matemático, aqueles com múltiplas são igualmente consistentes.

2.2. ESTRATÉGIAS DE RESOLUÇÃO DE SUDOKU

As estratégias de resolução de Sudoku correspondem a métodos lógicos e sistemáticos que são utilizados pelos jogadores para deduzir valores para uma determinada célula vazia, bem como identificar e descartar candidatos, sem a necessidade de recorrer à adivinhação ou tentativa e erro. Elas variam em complexidade, compreendendo técnicas básicas até estratégias avançadas, que são essenciais para os Sudokus mais difíceis.

Tom Davis (19) apresenta algumas dessas táticas e as define como matematicamente óbvias, embora reconheça que, para um ser humano, seja difícil aplicá-las em um tabuleiro, devido à grande quantidade de informações a serem analisadas. Por esse motivo, a apresentação dos métodos se iniciará pelos mais evidentes, seguindo em ordem crescente de dificuldade para um ser humano. Agora, considerando uma máquina, diferentes abordagens são possíveis e, muitas vezes, mais simples.

Os três primeiros métodos que serão detalhados nas subseções adiante, possibilitam o preenchimento de células vazias, enquanto, os dois últimos, por outro lado, apenas eliminam certos candidatos de determinadas células, o que pode contribuir para o complemento futuro do tabuleiro.

2.2.1. Full House/Last Digit

O *Full House (19)*, também conhecido como *Last Digit*, acontece quando oito dos nove números em qualquer linha, coluna ou sub-bloco já estiverem definidos, de modo que o elemento final seja aquele que está faltando (19).

A Figura 2 apresenta um exemplo em que uma célula vazia, localizada na linha sete e na coluna dois, pode ser preenchida por meio do *Full House*, pois o número 6 é a única opção para essa posição, considerando que todos os outros elementos já encontram-se dispostos na coluna.

Figura 2 - Exemplo da Técnica Full House

Fonte: Autor (2025)

2.2.2. Naked Single/Obvious Single

O *Naked Single (19)*, ou *Obvious Single*, ocorre quando, ao listar todos os candidatos para uma determinada célula vazia, apenas um permanece viável após a eliminação daqueles que já estão presentes na mesma linha, coluna e sub-bloco, de modo que a célula possa apenas assumir esse único elemento.

A Figura 3 mostra um exemplo em que uma célula vazia, localizada na linha um e na coluna um, pode ser preenchida através do *Naked Single,* considerando que o único candidato disponível para ela é o número 8, uma vez que os números 1, 4, 7 e 9 já estão presentes na linha, os números 3, 5 e 9 na coluna e os números 2, 4, 6 no sub-bloco 3x3 correspondentes.

Figura 3 - Exemplo da Técnica Naked Single

Fonte: Autor (2025)

2.2.3. Hidden Single

O *Hidden Single* (19) dá-se quando um determinado número apenas pode ser posto em uma única célula dentro de uma unidade (linha, coluna ou sub-bloco), mesmo que ela possua vários outros candidatos.

A Figura 4 ilustra um exemplo em que uma célula vazia, localizada na linha cinco e na coluna seis, pode ser preenchida recorrendo ao *Hidden Single*, uma vez que os três números 1, posicionados nas coordenadas: linha seis, coluna dois; linha quatro, coluna oito; linha oito, coluna quatro, restringem a possibilidade de inserção do número 1 no sub-bloco central, fazendo com que ele só possa ser colocado nas colunas cinco e seis e, após considerar as restrições das demais unidades, define-se que a única célula possível é a da linha cinco, coluna seis.

Figura 4 - Exemplo da Técnica Hidden Single

Fonte: Autor (2025)

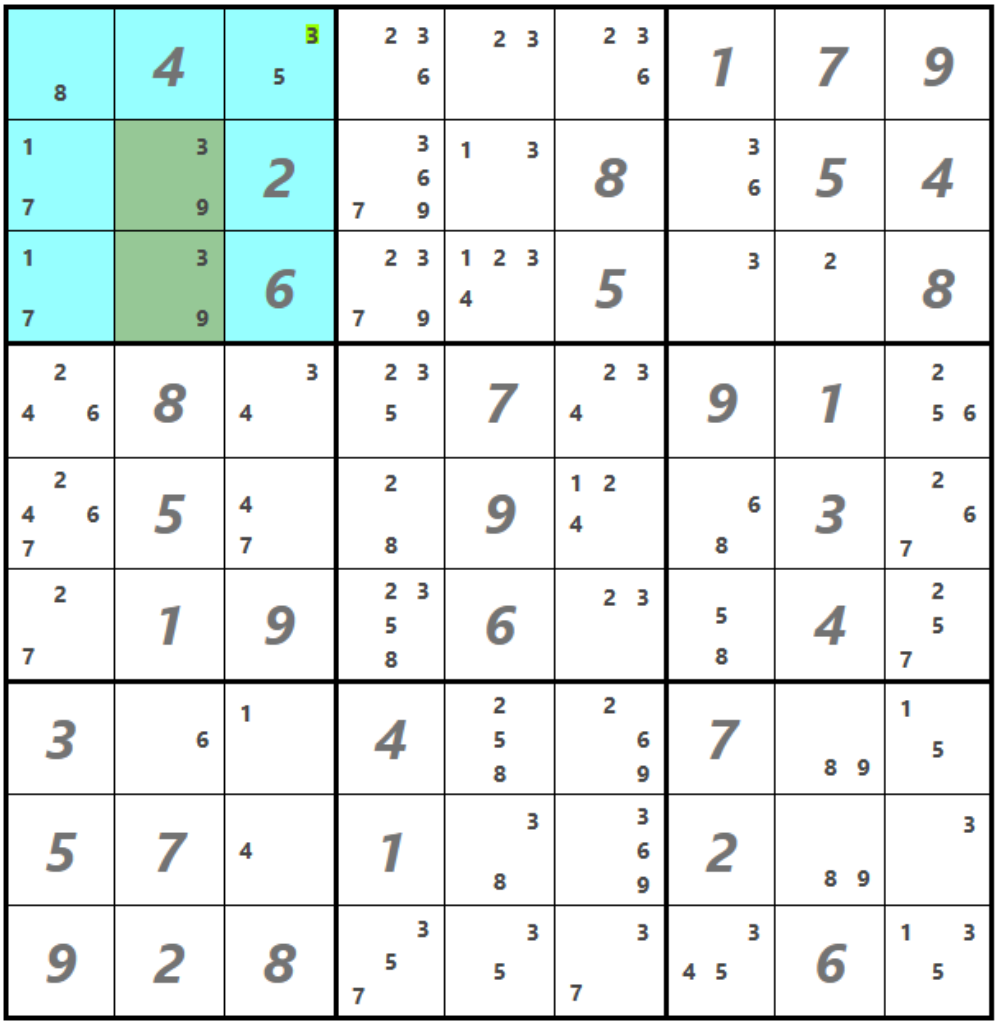
2.2.4. Naked Pairs/Obvius Pairs

O *Naked Pairs* (19), ou Obvious *Pairs*, acontece quando, em uma mesma unidade (linha, coluna ou sub-bloco), duas células apresentam exatamente os mesmo dois candidatos, e nenhum outro. Quando essa configuração ocorre, significa que esses dois números apenas podem ocupar essas duas células específicas, ainda que não se saiba ao certo em qual delas. Consequentemente, esses elementos podem ser eliminados como candidatos das demais células da unidade.

A Figura 5 apresenta um exemplo em que candidatos podem ser eliminados através do *Naked Pairs*, pois ao analisar o sub-bloco superior esquerdo, nota-se que duas células, destacadas em verde, contém exatamente os mesmo dois candidatos: os números 3 e 9, de modo que esses dois valores não podem estar presentes em nenhuma outra célula do

sub-bloco, desse modo, para a célula vazia na linha um e coluna três, o único candidato restante é o número 5, que pode, então, ser preenchido.

Figura 5 - Exemplo da Técnica Naked Pairs



Fonte: Autor (2025)

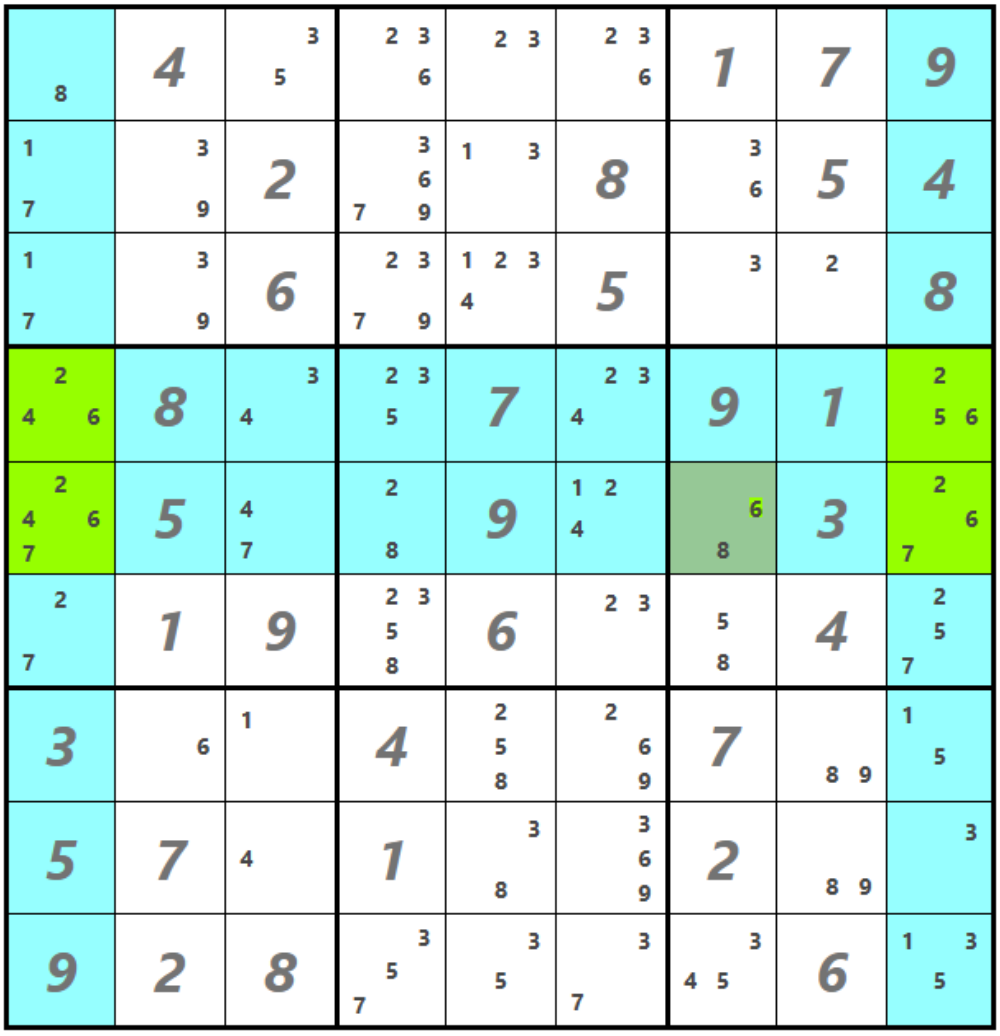
2.2.5. X-Wings

O *X-Wings* (19) dá-se quando um mesmo candidato aparece exatamente duas vezes em duas linhas, e essas ocorrências se alinham perfeitamente nas mesmas duas colunas. Nessa situação, o posicionamento desse número nas duas linhas acontece obrigatoriamente nas colunas que formam o padrão e baseado nisso o candidato em questão pode ser eliminado de todas as outras células daquelas duas colunas em questão. Similarmente, esse fato continua válido ao trocar as palavras “linha” e “coluna”.

A Figura 6 expõe um exemplo em que candidatos podem ser eliminados por meio da técnica *X-Wing*, visto que ao analisar as colunas um e nove, percebe-se que o candidato 6

aparece unicamente nas células que cruzam com as linhas quatro e cinco, o que assegura que o posicionamento deste número, nessas duas linhas, ocorrerá exclusivamente dentro das colunas citadas, de modo que ele pode ser removido de qualquer outra célula presente nelas, como a destacada em verde.

Figura 6 - Exemplo da Técnica X-Wings



Fonte: Autor (2025)

2.3. ALGORITMO GENÉTICO

Baseados diretamente na teoria da evolução de Charles Darwin, os Algoritmos Genéticos (AGs) representam uma classe de metaheurísticas de busca e otimização computacional (20), cuja inspiração surgiu dos estudos de John Holland na Universidade de Michigan, voltados a replicar a robustez e a eficiência da evolução natural como processo de otimização. Foi a partir dessas pesquisas que Holland formalizou suas ideias na década de

1970, publicando a obra “Adaptation in Natural and Artificial Systems” (21), que tornou-se uma das referências mais importantes sobre o tema.

A lógica dos AGs simula diretamente a seleção natural e o princípio da “sobrevivência do mais apto”. Sob essa premissa, os melhores indivíduos, isto é, as soluções de maior qualidade, são selecionadas preferencialmente para a reprodução, transmitindo suas características e permitindo que a população evolua a cada geração (20). Esse ciclo iterativo, viabilizado por operadores genéticos, refina continuamente a população de soluções, guiando-a em direção a um estado de maior adaptação e a resultados progressivamente superiores (21).

Embora esses algoritmos não garantam a obtenção de uma solução ótima global, sua principal vantagem encontra-se na capacidade de explorar espaços de buscas vastos e complexos para encontrar soluções de boa qualidade em tempo computacionalmente viável (20). Nas subseções seguintes, serão apresentados em detalhe os componentes fundamentais dos AGs, seus operadores genéticos, os mecanismos de seleção de sobrevivência e o ciclo evolutivo que o integra.

2.3.1. Componentes Fundamentais

O funcionamento de um Algoritmo Genético é definido pela configuração de seus componentes básicos, responsáveis por traduzir os conceitos biológicos para uma estrutura computacional. O modo como cada um dos elementos é projetado e parametrizado é determinante para o desempenho do algoritmo, pois balanceia a exploração de novas áreas do espaço de busca e o refinamento das melhores possíveis soluções encontradas. Dentre esses elementos, destacam-se a forma de representar as soluções (indivíduos e cromossomos), o método de avaliar sua qualidade (função de aptidão) e os mecanismos que geram (operadores genéticos) e selecionam as novas soluções. As subseções seguintes apresentam cada um desses componentes.

2.3.1.1. Indivíduos e População

A estrutura de um Algoritmo Genético opera sobre um conjunto de soluções candidatas, conhecido como população, que corresponde a uma amostra de diferentes pontos no espaço de busca. Cada elemento que a constitui é chamado de indivíduo, que representa uma única solução para o problema. Por sua vez, essa solução é codificada em uma estrutura de dados, como um vetor ou uma lista, denominada cromossomo (20).

Além de sua função estrutural, a população desempenha um papel duplo fundamental no algoritmo. Primeiramente, ela reflete o conhecimento acumulado sobre as regiões mais promissoras do espaço de busca, visto que os indivíduos mais aptos possuem genes que representam soluções mais eficazes. Em segundo lugar, serve como fonte de variabilidade, fornecendo a base sobre a qual os operadores genéticos atuam para gerar novas soluções e ampliar a exploração do espaço de busca (20).

2.3.1.2. Função de Aptidão

A função de aptidão é o elemento central na avaliação das soluções em um Algoritmo Genético, pois define como cada indivíduo será julgado em relação ao problema. Ela consiste em uma função matemática que analisa um indivíduo e atribui um valor numérico que expressa sua qualidade como solução. Definir uma função de aptidão eficaz é um dos passos mais desafiadores e críticos na construção de um AG, já que seus critérios precisam refletir com precisão o que caracteriza uma boa solução. Do contrário, há o risco de recompensar apenas parte do objetivo ou valorizar características indesejadas, direcionando a busca para caminhos improdutivos (20).

O valor de aptidão orienta diretamente o processo de seleção, aplicando uma pressão seletiva que guia a busca pelas regiões mais promissoras do espaço de soluções. A intensidade desta pressão depende da forma como os valores de aptidão estão distribuídos. Funções que geram grandes variações de aptidão tendem a impor uma pressão elevada, acelerando a convergência da população. No entanto, essa estratégia envolve um risco: a perda de diversidade populacional, que pode levar o algoritmo a ficar preso em ótimos locais e deixar de explorar regiões de elevado potencial (20).

2.3.1.3. Operadores Genéticos

Os operadores genéticos são os mecanismos que conduzem o processo de evolução do AG, simulando o processo de reprodução e a variação genética. Eles são responsáveis por manipular o cromossomo dos indivíduos para gerar novos descendentes que podem herdar características promissoras de seus pais (20). A escolha e a adaptação de cada operador dependem diretamente da natureza do problema abordado, de modo que para casos baseados em permutação, como o Sudoku, são necessárias técnicas específicas de cruzamento e mutação que mantenham a validade dessa estrutura. Nas subseções seguintes encontram-se detalhados os principais operadores utilizados neste trabalho: seleção de pais, cruzamento e mutação.

2.3.1.3.1. Mecanismo de Seleção de Pais

A primeira etapa para a criação de uma nova geração é a escolha dos membros da população atual que irão se reproduzir. Este mecanismo, denominado seleção dos pais, utiliza os valores de aptidão para identificar e selecionar os indivíduos com base em sua qualidade, possibilitando que os melhores se tornem genitores da próxima geração (20). Dentre as estratégias disponíveis para esse fim, destacam-se a Seleção por Roleta e a Seleção por Torneio.

- Seleção por Roleta: Nesta estratégia, o algoritmo calcula a distribuição de probabilidade cumulativa para a população, em seguida, para cada seleção, um número aleatório é gerado e o indivíduo correspondente a essa faixa de distribuição é escolhido para o grupo de acasalamento (20).

- Seleção por Torneio: Neste método, um grupo de k indivíduos é escolhido aleatoriamente para competir em um "torneio" e aquele com maior aptidão entre os competidores é selecionado como genitor. Neste contexto, os principais fatores que influenciam essa estratégia são: o tamanho do torneio, uma vez que aumentar k intensifica a chance de escolher um indivíduo mais apto; a probabilidade (p) do membro mais apto do torneio ser selecionado, já que a seleção não determinística $(p < 1)$ introduz um elemento de aleatoriedade; e o tipo de seleção, com ou sem reposição, considerando que a seleção sem reposição tende a favorecer os mais aptos (20).

2.3.1.3.2. Cruzamento

O cruzamento, também conhecido como recombinação, é um operador genético tradicionalmente binário (mas, em casos mais raros, n-ário) que mescla informações genéticas de dois ou mais indivíduos para gerar um ou mais descendentes. O princípio fundamental deste operador é a combinação de membros da população com características distintas, porém desejáveis, com objetivo de produzir um descendente que herde o melhor de ambos e, potencialmente, represente uma solução de maior qualidade. Essa capacidade de explorar combinações para gerar soluções possivelmente superiores faz do cruzamento o principal operador de busca dos AGs (20).

Como indicado anteriormente, os operadores genéticos são dependentes da representação do indivíduo, sendo assim para problemas de permutação, como o Sudoku, os

operadores de cruzamento convencionais não são adequados, considerando que, na maioria dos casos, resultam em descendentes inválidos, com valores duplicados e/ou ausentes, que violariam as restrições fundamentais do problema. Dessa forma, a resolução de problemáticas desse tipo exige técnicas de cruzamento especializadas, projetadas para garantir que a propriedade de permutação seja mantida nos descendentes (20). A seguir, encontram-se detalhados três dos operadores mais conhecidos para essa classe de problema: o Partially Mapped Crossover (PMX), o Order Crossover (OX) e o Cycle Crossover (CX).

- O Partially Mapped Crossover (PMX):

O *Partially Mapped Crossover* (PMX), proposto originalmente por Goldberg e Lingle (22), é um dos operadores mais utilizados para problemas em que a adjacência entre os valores no cromossomo é um fator importante (20). Seu funcionamento consiste em um processo de troca e mapeamento, que ocorre conforme as seguintes etapas:

1. Seleção do segmento: Dois pontos de corte são escolhidos de forma aleatória, definindo um segmento correspondente em ambos os pais;
2. Cópia do segmento: O segmento do primeiro pai é copiado para o descendente;
3. Realização do mapeamento: Os valores no segmento trocado do primeiro pai estabelecem um mapeamento com os valores no segmento correspondente do segundo pai;
4. Preenchimento das posições vazias: Fora do segmento, os valores do segundo pai são copiados para as posições vazias do descendente. Caso um valor do segundo pai já exista no segmento que foi trocado, ele é substituído pelo valor correspondente no mapeamento, estabelecido na etapa anterior, antes de ser inserido no descendente.

Esse processo garante que as informações de adjacência e de posição absoluta sejam parcialmente herdadas dos genitores (20).

- Order Crossover (OX):

O *Order Crossover* (OX), desenvolvido por Davis (23), é um operador projetado para problemas em que a ordem relativa dos valores nos cromossomos é mais importante do que suas posições absolutas ou adjacentes (20). Seu mecanismo se baseia na cópia de um segmento de um pai e no preenchimento do restante com base na ordem em que os valores aparecem no outro genitor, conforme descrito nas etapas a seguir:

1. Seleção do segmento: Semelhante ao PMX, dois pontos de corte aleatórios definem um segmento em um dos pais;

2. Cópia do segmento: O segmento é copiado diretamente para o descendente nas mesmas posições;

3. Preenchimento das posições vazias: As posições vazias são preenchidas na ordem em que aparecem no segundo pai, a partir do segundo ponto de corte, ignorando os valores já copiados do primeiro pai. O preenchimento é feito de forma circular, isto é, após o final do cromossomo, o processo continua a partir do início;

- Cycle Crossover (CX):

O *Cycle Crossover* (CX) é um operador cuja principal característica é a preservação da posição absoluta dos valores presentes no cromossomo de cada pai no descendente (20). Seu mecanismo se baseia na identificação de "ciclos", um conjunto de valores que ocupam as posições uns dos outros nos cromossomos dos pais, e encontra-se detalhado nos passos adiante:

1. Identificação dos ciclos: O processo inicia na primeira posição do primeiro pai. O valor nessa posição é mapeado para o valor que se encontra no mesmo local do segundo pai. Em seguida, a posição deste segundo valor é localizada no primeiro pai e a busca continua de forma iterativa até que se retorne ao valor inicial, fechando o ciclo. Essa operação é repetida para as posições ainda não visitadas até que todos os valores sejam mapeados em ciclos;

2. Geração dos descendentes: A formação do descendente começa com a cópia dos valores do primeiro ciclo, provenientes do primeiro pai. Em seguida, os valores do segundo ciclo são copiados do segundo pai, e assim por diante, alternando a ordem para cada ciclo subsequente.

2.3.1.3.3. Mutação

A mutação é um operador genético secundário responsável por introduzir variações aleatórias nos indivíduos. Sua função é evitar a convergência prematura do algoritmo para um ótimo local, possibilitando a exploração de novas regiões do espaço de busca. Para representações de permutação, não é viável considerar cada gene de forma independente, sendo assim encontrar mutações válidas se torna uma questão de trocar valores no

cromossomo. Como consequência, nesses casos, o parâmetro de mutação é interpretado como a chance do cromossomo sofrer mutação, e não de uma única posição dele ser alterada (20). Isso pode ser constatado nos seguintes exemplos:

- Mutação por Troca (Swap Mutation): Duas posições (genes) no cromossomo são selecionadas aleatoriamente e seus valores são trocados;
- Mutação por Inserção: Dois valores que fazem parte do cromossomo são selecionados aleatoriamente e o segundo é movido para junto do primeiro, deslocando os outros para abrir espaço;
- Mutação por Embaralhamento (Scramble Mutation): O cromossomo inteiro ou um subconjunto que o constitui tem suas posições embaralhadas;
- Mutação por Inversão (Inversion Mutation): Duas posições no cromossomo são selecionadas aleatoriamente e a ordem em que os valores aparecem entre elas é revertida.

2.3.1.4. Mecanismo de Seleção de Sobreviventes e Elitismo

Após a geração de novos indivíduos por meio dos operadores genéticos, o AG lida com um um excedente populacional temporário, uma vez que o conjunto agora é composto pelos pais (μ) e seus descendentes (λ). Neste contexto, para que o ciclo evolutivo prossiga, é aplicada uma etapa crucial conhecida como seleção de sobreviventes. Ela corresponde ao processo responsável por reduzir o conjunto μ de pais e λ de filhos de volta a uma população de μ que avançará para o ciclo evolutivo seguinte. As estratégias para realizar essa seleção podem ser categorizadas de duas formas: aquelas baseadas na idade dos indivíduos e aquelas baseadas em sua aptidão (20).

No caso do modelo geracional ($\mu = \lambda$), a seleção é inteiramente baseada na idade, pois nele o número de descendentes gerados é igual ao tamanho da população de pais, e toda a população de genitores é descartada para ser integralmente substituída pelo conjunto de descendentes, que então se torna a nova geração. Nesse tipo de abordagem, a aptidão dos indivíduos não é um critério para a sobrevivência, uma vez que os pais são substituídos independentemente de quão adaptados são. Essa estratégia pode ser benéfica para evitar a convergência prematura em ótimos locais, contudo ela apresenta uma desvantagem

significativa: a possibilidade de perder um indivíduo de alta aptidão caso nenhum de seus descendentes atinja um nível de qualidade similar ou superior (20).

Por outro lado, no modelo de estado estacionário, pais e filhos coexistem e competem por um lugar na próxima geração e a sobrevivência é determinada pela aptidão. Nesse cenário, um conjunto unificado de pais e filhos ($\mu + \lambda$) é formado, e a partir dele, os μ indivíduos mais aptos são escolhidos para compor a próxima geração. De modo geral, qualquer mecanismo de seleção de pais, como: roleta, torneio e ranking, pode ser aplicado a esse conjunto unificado para determinar os sobreviventes (20).

Existem também abordagens híbridas, como o modelo geracional ($\lambda > \mu$), que combinam critérios de idade e aptidão. Nesse caso, os μ pais são sempre descartados (critério de idade) e os μ sobreviventes são selecionados a partir do conjunto de λ descendentes, com base em sua aptidão (20).

Um fator importante, especialmente em estratégias que descartam os pais, é a possibilidade do melhor indivíduo da população ser perdido, caso nenhum dos descendentes gerados tenha aptidão igual ou superior a ele. Para contornar essa questão e garantir a preservação de soluções de alta qualidade, pode-se aplicar o mecanismo de elitismo. Essa técnica consiste em identificar o melhor ou melhores indivíduos da população atual e garantir que ele seja transferido para a próxima geração, a menos que descendentes melhores tenham sido criados. Desse modo, pode-se dizer que o elitismo assegura que a qualidade da melhor solução encontrada pelo algoritmo nunca reduza de uma geração para outra (20).

2.3.2. Ciclo Evolutivo

O ciclo evolutivo de um Algoritmo Genético é estruturado através da interação entre os componentes previamente descritos e se inicia após a criação de uma população inicial, frequentemente construída de forma aleatória. Uma vez estabelecida essa população, o algoritmo prossegue com um laço iterativo, que se repete até que um critério de parada, como o número máximo de gerações ou a estagnação da qualidade da solução, seja atingido. Dentro de cada iteração, ou geração, são executadas as seguintes etapas:

1. Avaliação da População: A função de aptidão é utilizada para avaliar a qualidade de cada indivíduo da população atual;

2. Seleção de Pais: Com base nos valores de aptidão, os indivíduos são selecionados para participar do processo reprodutivo;

3. Geração de Descendentes: A aplicação dos operadores genéticos depende das estratégias estabelecidas para o algoritmo, contudo, na maioria dos casos, o cruzamento é aplicado primeiro para combinar os indivíduos selecionados e, em seguida, a mutação é aplicada aos descendentes;

4. Seleção de Sobreviventes: Uma estratégia de sobrevivência, comumente associada ao elitismo, é aplicada para determinar quais indivíduos (da população original e/ou da de descendentes) formarão a população da próxima geração;

5. Substituição da População: A população antiga é substituída pela nova.

Este processo se repete de modo contínuo, promovendo a evolução da população em direção a regiões de maior aptidão no espaço de busca (20).

2.4. BUSCA LOCAL

A busca local representa uma classe de algoritmos de otimização heurística, que explora o espaço de soluções de um problema movendo-se de um estado inicial para uma "estado vizinho" (24). O conceito de "vizinhança" é fundamental para essa abordagem, pois define o conjunto de soluções que podem ser alcançadas a partir do estado atual, aplicando-se uma operação ou um conjunto de operações de modificação (25). Diferente dos algoritmos que constroem uma solução do início, os métodos de busca local partem de uma solução inicial, muitas vezes, gerada de forma aleatória (24) ou por uma heurística construtiva, e buscam aprimorá-la iterativamente, por meio de alterações sucessivas para vizinhos de maior qualidade. A efetividade desses algoritmos depende dos passos de busca local (*local search steps*), que formam o gráfico de vizinhança (*neighbourhood graph*), e dos mecanismos e parâmetros que auxiliam a exploração do espaço de busca (26).

A principal vantagem dessa abordagem é sua eficiência no uso de recursos, considerando que ela utiliza pouca memória e, frequentemente, consegue encontrar soluções razoáveis em conjuntos de estados grandes ou infinitos, para os quais algoritmos sistemáticos são inadequados (24). Contudo, uma limitação comum a muitos desses métodos é a tendência de conversão para ótimos locais, que são soluções boas dentro de sua vizinhança, mas não necessariamente a melhor global do problema (23, 24).

Nas subseções a seguir serão detalhados os dois algoritmos de busca local testados neste projeto: a variação *First Improvement* do algoritmo *Hill Climbing* e o *Min-Conflicts*.

2.4.1. First Improvement Hill Climbing

O *Hill Climbing* é um dos algoritmos de busca local mais populares atualmente. Sua lógica consiste em mover-se continuamente em direção a um ponto mais alto no espaço de busca, isto é, uma solução vizinha que seja melhor que a atual (24). Existem diferentes estratégias para selecionar o próximo estado, conhecidas como regras de pivoteamento (*pivoting rules*) (27). No caso do *First Improvement Hill Climbing*, especificamente, ele examina a vizinhança da solução atual, em uma ordem pré-determinada, e seleciona o primeiro vizinho encontrado que oferece alguma melhoria na função objetivo (27). As principais características dessa abordagem são:

- Simplicidade: Sua implementação é relativamente simples, exigindo apenas uma função para encontrar vizinhos e outra para avaliar a qualidade da solução (24);
- Dependência da Ordem da Vizinhança: O melhor local encontrado pelo First Improvement depende da ordem em que a vizinhança é examinada. Esta propriedade pode ser aproveitada ao utilizar uma ordem aleatória para realizar a verificação, de modo a diversificar o algoritmo. Como resultado, ele passa a ser capaz de encontrar bons locais diferentes em múltiplas execuções, mesmo partindo da mesma solução inicial (27).
- Velocidade: Por não precisar avaliar todos os estados vizinhos para definir o movimento seguinte, ele geralmente é mais rápido que métodos como o *Best Improvement* (27), tornando-se uma provável nos casos em que a vizinhança é grande;
- Vulnerabilidade a Ótimos Locais: Essa característica surge do critério de parada que utiliza, considerando que a busca termina ao alcançar um ponto onde nenhum vizinho imediato representa uma melhoria, o algoritmo pode ficar preso em um ótimo local, que é um pico maior do que cada um dos seus estados vizinhos, contudo menor do que o máximo global (24);

2.4.2. Min-Conflicts

O algoritmo *Min-Conflicts*, proposto por Jung Gu (28) em 1989 e desenvolvido de forma independente por Minton et al. (29, 30), é uma busca local projetada para resolver

Problemas de Restrições (PSR). Sua premissa é partir de uma atribuição inicial para as variáveis em um problema, mesmo que inconsistente, e buscar no espaço de possíveis reparos (30). Sendo assim, a estratégia pode ser descrita da seguinte forma:

1. Escolha aleatória de uma variável que está atualmente em conflito, ou seja, que viola uma ou mais restrições do problema;
2. Avaliação de todos os valores possíveis para essa variável;
3. Seleção do valor que minimiza o número de conflitos com as demais. Caso haja múltiplos valores em condições semelhantes, um deles é selecionado de forma aleatória;
4. Repetição do processo até que uma solução sem conflitos seja encontrada ou um número máximo de iterações seja atingido;

Sua principal característica é o foco em reparar a solução de forma incremental, concentrando-se nas inconsistências existentes (29). Para muitos problemas, a busca é rápida, pois a avaliação de cada movimento é simples e focada nas restrições diretamente associadas a variável a ser modificada.

A eficiência do método foi tanto demonstrada em problemas de escala massiva, como o de Um Milhão de Rainhas, tendo sido capaz de resolvê-lo, ainda na época, em menos de quatro minutos utilizando uma SPARCstation 1 (30), quanto na aplicação prática bem-sucedida no escalonamento de atividades para o Telescópio Espacial Hubble (23, 29). Nesse contexto, apesar de não garantir otimalidade ou completude por poder estagnar em ótimos locais, o sucesso prático do algoritmo o estabelece como uma das heurísticas mais populares para PSRs de grande porte.

3. TRABALHOS RELACIONADOS

Desde o momento em que o Sudoku se popularizou, sua resolução computacional tem sido explorada por meio de diversas abordagens, que vão desde algoritmos de busca clássicos até metaheurísticas complexas e sistemas híbridos. Essa seção apresenta os trabalhos de maior relevância que fundamentam e contextualizam a presente pesquisa, destacando seus métodos, diferenciais e resultados de desempenho.

Um dos trabalhos de referência na área é o de Mantere e Koljonen (3), que exploraram o uso de Algoritmos Genéticos (AGs) para três finalidades distintas: a resolução, a avaliação

de dificuldade e a geração de novos quebra-cabeças. Nessa abordagem, a solução é representada por um array de 81 inteiros e os operadores genéticos são projetados para nunca gerar configurações inválidas para as subgrades 3x3, o que garante a satisfação inerente dessa restrição. Como resultado, a validação do AG como método de resolução foi confirmada por sua capacidade de solucionar consistentemente um vasto conjunto de quebra-cabeças de diferentes níveis. Posteriormente, os autores foram capazes de validar sua eficácia para avaliar a dificuldade dos quebra-cabeças, ao demonstrarem que o número de gerações necessárias para a resolução dos Sudokus se correlacionou diretamente com as classificações de complexidade indicadas em suas fontes. Finalmente, sua função como ferramenta de geração de novos *puzzles* foi comprovada através do seguinte método: primeiro, o AG criava uma grade completa e válida, depois, era realiza a remoção de números para formar as pistas no novo quebra-cabeça, então, o mesmo algoritmo era utilizado para resolver o *puzzle* múltiplas vezes, a fim de testar e garantir a unicidade de sua solução.

Em um trabalho posterior (7), Mantere e Koljonen investigaram os Algoritmos Culturais, que estendem os AGs ao incluir um "espaço de crença": uma estrutura de dados $9x9x9$ que armazena informações de soluções quase ótimas com objetivo de guiar a evolução. Em termos de desempenho, o AC mostrou-se em média 8,7% mais eficiente que o AG padrão, superando-o em 11 das 15 classes de dificuldade avaliadas, com a melhoria sendo mais perceptível nos quebra-cabeças mais difíceis.

A evolução dos estudos levou ao desenvolvimento de abordagens híbridas, como a apresentada em Mantere (9), que propõe um sistema que combina AG com Otimização por Colônia de Formigas (GA/ACO). Nele, o AG atua como um buscador global, e o ACO, como um buscador local ganancioso, que utiliza trilhas de feromônio para acelerar a convergência. O diferencial desse trabalho, em comparação com as versões anteriores do mesmo autor, foi a implementação de uma nova estratégia de elitismo dinâmico, que resultou em uma melhoria de desempenho de 48,7% para o AG, 45,1% para o AC e 12,5% para o GA/ACO. Seguindo uma abordagem distinta, Wang et al. (31) desenvolveram um algoritmo chamado de LSGA, que incorpora no Algoritmo Genético uma busca local por colunas e sub-blocos e um mecanismo de aprendizado de população de elite. Esse sistema demonstrou alta performance em seus testes, superando outros algoritmos ao qual foi comparado, ao resolver 100% dos quebra-cabeças difíceis, em alguns casos, exigindo média de gerações inferior para encontrar a solução. De forma notável, ele também foi capaz de resolver o famoso "AI Escargot", considerado por muitos como um dos Sudokus mais difíceis (7). Em outra vertente de

hibridização, Amil e Mantere (13) integraram Algoritmos Evolutivos com estratégias lógicas de resolução por meio da introdução de uma etapa de pré-processamento. Essa abordagem se mostrou altamente eficaz, considerando que parte dos quebra-cabeças classificados como fáceis e médios foram resolvidos apenas nesta fase inicial, enquanto nos mais difíceis, o tempo e o número de gerações necessários para solução foram drasticamente reduzidos.

Em paralelo, outros pesquisadores se dedicaram ao estudo de algoritmos mais tradicionais. Indriyono et al. (5) realizaram uma análise comparativa entre o *backtracking* e os Algoritmos Genéticos e concluíram que nos quebra-cabeças testados, a primeira abordagem, mesmo sendo mais simples, foi mais rápida nos *puzzles* mais difíceis. Contudo, o mesmo estudo apresenta o AG como um método preciso para resolver desafios complexos de otimização que não podem ser resolvidos com métodos convencionais, sugerindo que embora eles nem sempre sejam os mais rápidos, seriam muitas vezes mais eficazes no encontro da solução de problemas de maior complexidade. De forma semelhante, Diah et al. (4) compararam os algoritmos de busca clássicos DFS e BFS e os resultados indicaram que o DFS foi significativamente mais eficiente em termos de tempo de execução.

Nesse contexto, pode-se concluir que a literatura apresentada corrobora com a ideia de que algoritmos determinísticos podem ser mais eficientes, no quesito tempo de execução, na solução de alguns problemas, contudo as abordagens meta-heurísticas e, principalmente, as híbridas, oferecem maior robustez para desafios mais complexos. Sendo assim, o algoritmo híbrido, que combina estratégias de resolução lógicas com AG, proposto neste trabalho, mostra-se alinhado com as pesquisas mais recentes na área.

4. DESENVOLVIMENTO

A presente seção detalha a metodologia e as escolhas de implementação que orientaram a construção do solucionador de Sudoku. Para contextualizar as decisões tomadas ao longo do desenvolvimento, inicialmente, será apresentada uma visão geral do algoritmo proposto, explorando sua estrutura, a justificativa para a abordagem adotada e seu fluxo de funcionamento. Em seguida, serão descritas as etapas de definição inicial, evolução e refinamento dos parâmetros e operadores do Algoritmo Genético ao longo do processo de construção e aprimoramento. Finalmente, será apresentada a versão final do algoritmo, com destaque para as configurações adotadas e uma descrição aprofundada do funcionamento das duas etapas principais do sistema: o pré-processamento, baseado em técnicas clássicas de

resolução de sudoku, e a implementação do AG, com a explicação de seus principais componentes e operadores.

4.1. VISÃO GERAL DO ALGORITMO

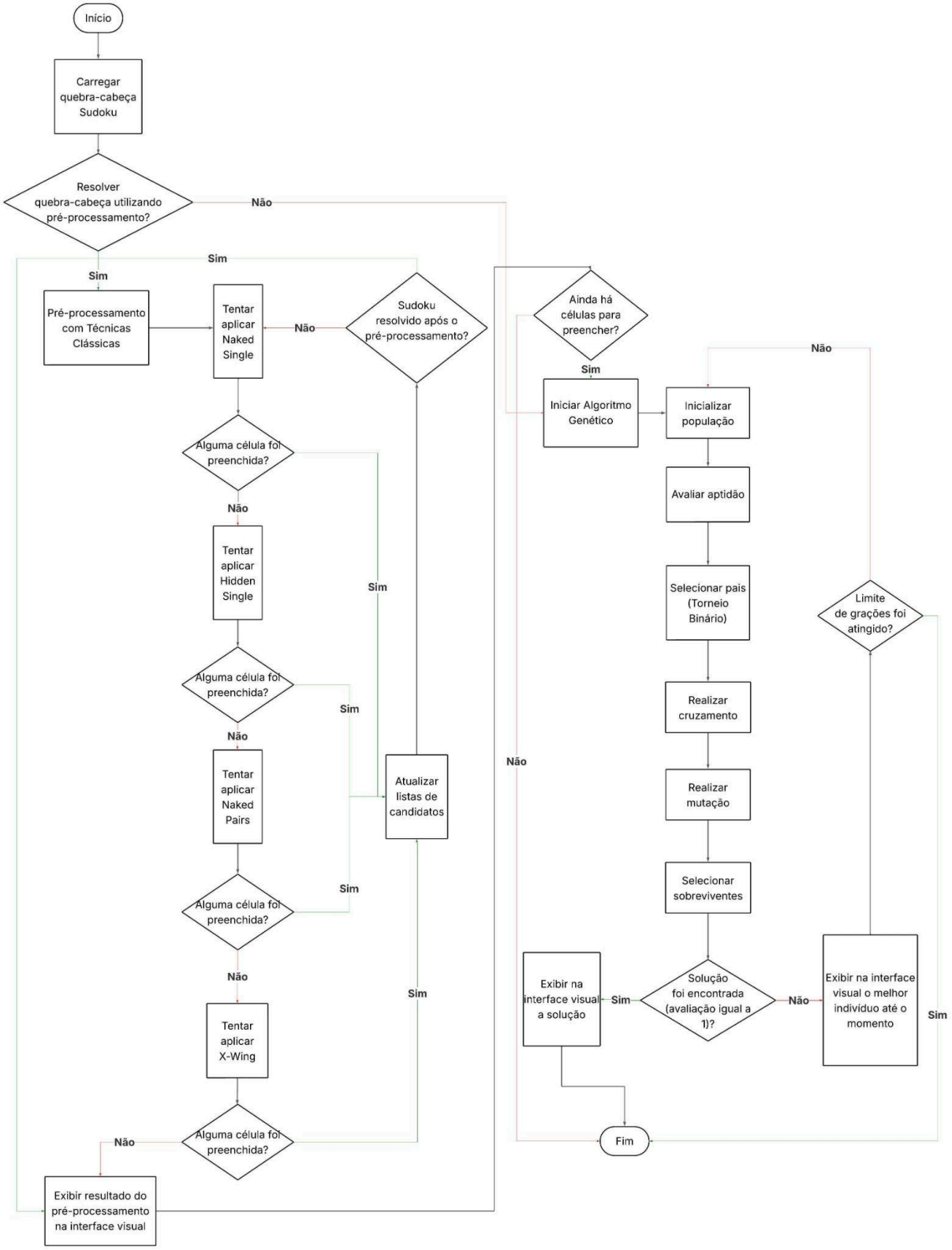
O sistema proposto neste projeto adota uma abordagem algorítmica híbrida para resolução de quebra-cabeças Sudoku, combinando estratégias determinísticas com um algoritmo meta-heurístico. O objetivo dessa combinação é a reunião dos pontos fortes de ambas as metodologias: a precisão e a agilidade das técnicas clássicas nas deduções iniciais, aliadas à robustez e adaptabilidade dos Algoritmos Genéticos na exploração de espaços de busca complexos, visando a obtenção de soluções para quebra-cabeças, especialmente, de níveis médio e difícil.

A motivação por trás da adoção da abordagem híbrida encontra-se na natureza dos quebra-cabeças Sudoku e nas limitações dos métodos puramente determinísticos ou meta-heurísticos quando utilizados isoladamente. Embora as técnicas clássicas sejam eficazes para preencher células “óbvias” e simplificar a complexidade do problema, elas frequentemente chegam em um ponto em que as deduções adicionais, necessárias para o encontro da solução, exigem a aplicação de métodos mais complexos ou específicos e/ou diversas tentativas e erros. Em contrapartida, o Algoritmo Genético, apesar de sua capacidade de explorar vastos espaços de busca, é menos eficiente para deduções iniciais e diretas, demandando mais gerações para convergir, especialmente para quebra-cabeças mais simples ou nas etapas iniciais de problemas mais complexos.

Nesse contexto, a integração do pré-processamento determinístico ao Algoritmo Genéticos procura refinar o espaço de busca ao preencher algumas células vazias que apresentam números dedutíveis, por meio da aplicação de técnicas clássicas. Assim, com um ponto de partida mais otimizado, o AG pode operar sobre uma instância simplificada, acelerando sua convergência e aumentando a eficiência e a taxa de sucesso, em comparação com os métodos puramente determinísticos ou meta-heurísticos isoladamente, sobretudo em quebra-cabeças mais desafiadores.

A Figura 7 apresenta o fluxograma do sistema desenvolvido:

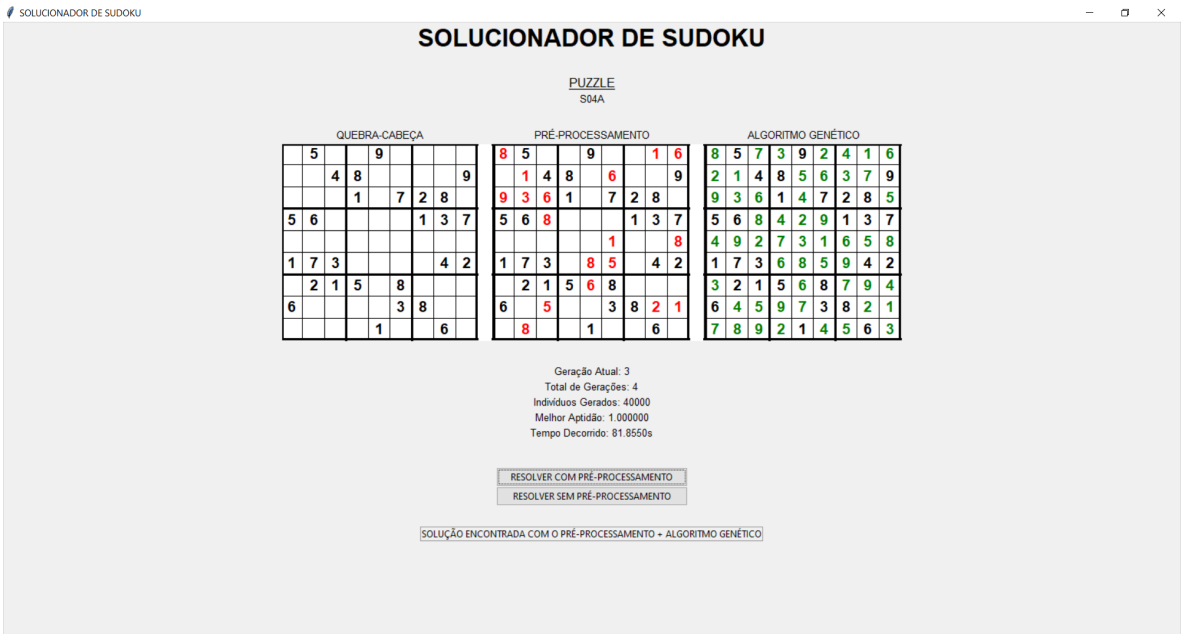
Figura 7 - Fluxograma do Sistema



Fonte: Autor (2025)

De modo a aprimorar a experiência de uso do sistema, facilitando a visualização do processo de resolução do Sudoku, o sistema foi encapsulado em uma interface, apresentada na Figura 8, que foi desenvolvida com a biblioteca interna do Python, Tkinter. Esta interface permite ao usuário, após carregar previamente o Sudoku de sua escolha a partir de um arquivo de texto, observar, em tempo real, o quebra-cabeça inicial e as duas etapas do processo, mais especificamente o resultado da aplicação das técnicas de pré-processamento e a evolução da melhor solução encontrada pelo AG a cada geração. Além das grades do Sudoku, a interface também exibe métricas importantes como o número da geração atual, a contagem de indivíduos totais, o valor da melhor aptidão no momento e o tempo de execução do processo, possibilitando um acompanhamento facilitado da resolução.

Figura 8 - Interface gráfica do sistema



Fonte: Autor (2025)

4.2. DEFINIÇÃO DE PARÂMETROS E OPERADORES

No início do desenvolvimento do solucionador, o objetivo principal dos testes realizados foi averiguar o funcionamento apropriado dos módulos que o integram, individual e em conjunto. Portanto, foram definidos alguns parâmetros e configurações iniciais com base nos resultados obtidos pelo algoritmo nesse primeiro momento. Inicialmente, foi utilizado um pequeno conjunto de quebra-cabeças de teste, constituído por apenas 5 *puzzles*, disponíveis online no site sudoku.com, que apresentavam de 23 a 36 dicas, isto é, do nível fácil ao difícil, que ajudaram a determinar alguns operadores iniciais, como o método de seleção de pais

(torneio), operador de cruzamento (PMX), o operador de mutação (troca) e a seleção de sobreviventes com elitismo.

Entretanto, com o avanço do algoritmo e a melhoria do seu desempenho, surgiu a necessidade de testar a sua performance com um conjunto mais completo de quebra-cabeças, provenientes de diferentes fontes, que apresentassem características distintas com relação à simetria e quantidade de dicas. Desse modo, passou-se a utilizar nos testes subsequentes um grupo de 46 Sudokus (nível fácil ao muito difícil), apresentados em Mantere e Koljonen (7), que apresentam suas as características listadas na Tabela 1.

Tabela 1 - Características dos Sudokus resolvidos pelo algoritmo na etapa de testes

(continua)

Dificuldade	Instâncias Sudoku A	Quantidade de Dicas (A)	Instâncias Sudoku B	Quantidade de Dicas (B)	Instâncias Sudoku C	Quantidade de Dicas (C)
1	s01a	33	s01b	36	s01c	32
2	s02a	30	s02b	28	s02c	28
3	s03a	28	s03b	26	s03c	27
4	s04a	28	s04b	27	s04c	28
5	s05a	30	s05b	28	s05c	26
E	s06a	36	s06b	39	s06c	36
C	s07a	25	s07b	25	s07c	25
D	s08a	22	s08b	23	s08c	22
SD	s09a	23	s09b	22	s09c	22
EASY	s10a	31	s10b	31	s10c	32
MEDIUM	s11a	28	s11b	26	s11c	28

Tabela 1 - Características dos Sudokus resolvidos pelo algoritmo na etapa de testes

(conclusão)

Dificuldade	Instâncias Sudoku A	Quantidade de Dicas	Instâncias Sudoku B	Quantidade de Dicas	Instâncias Sudoku C	Quantidade de Dicas
HARD	s12a	22	s12b	26	s12c	23
GA-E	s13a	32	s13b	33	s13c	37
GA-M	s14a	29	s14b	32	s14c	31
GA-H	s15a	27	s15b	27	s15c	27
AI ESCARGOT			s16			27

Fonte: Autor (2025)

Essa nova etapa de testes envolveu um processo iterativo de experimentação e ajuste de seus parâmetros e operadores, fundamental para a otimização de seu desempenho. Este processo foi dividido em fases distintas, buscando refinar a configuração do AG de forma sistemática, de acordo com as necessidades percebidas na análise dos resultados de cada execução realizada. Em cada uma dessas fases, para cada configuração de parâmetros testada, o desempenho do AG foi avaliado resolvendo-se todos os 46 *puzzles* do conjunto de testes, tanto sem pré-processamento quanto com pré-processamento, e cada combinação foi executada três vezes para garantir a robustez dos resultados. Nos parágrafos seguintes encontram-se os detalhamentos de cada fase de teste.

Inicialmente, na fase de Ajuste Inicial, o foco esteve em estabelecer uma linha de base de desempenho. Durante esse momento, 94 configurações diferentes de parâmetros e operadores foram testadas e exploradas à medida que as dificuldades enfrentadas pelos algoritmos eram percebidas, especialmente, ao analisar a taxa de sucesso alcançada por ele e os gráficos *boxplot* que indicavam a convergência da solução. Nessa etapa foram testados diferentes tamanhos de população (valores entre 150 até 3000), quantidades máximas de gerações (valores entre 25 até 500), tipos de seleção de pais (torneio binário e torneio com grupos ponderados), operadores de mutação (apenas troca, e troca com busca local do tipo *First Improvement Hill Climbing* e *Min-Conflicts)*, critérios de adaptação da mutação (baseado na regra de ⅕ de sucesso, amplitude da aptidão, aptidão média e aptidão mediana), critérios de seleção de sobrevivência (modelo geracional $\mu = \lambda$, com elitismo, de estado

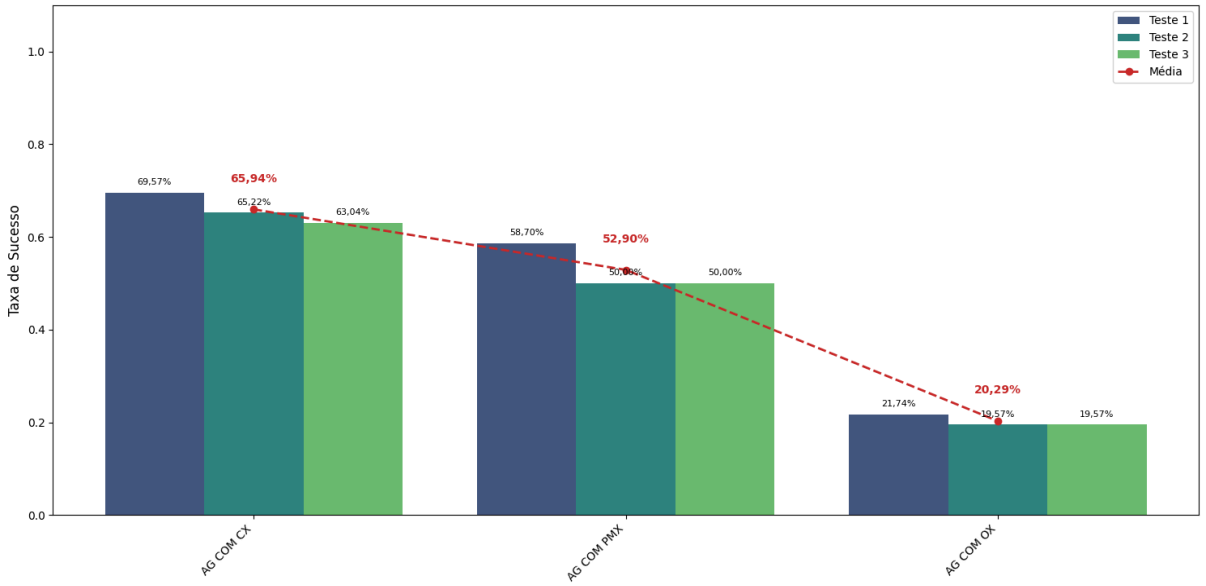
estacionário $\mu + \lambda$ com elitismo, e percentual de elitismo). Além disso, nesse momento, foi testado um reinício parcial da população para promover uma maior diversidade quando a mesma estagnava, considerando que o maior problema percebido durante esses testes foi a convergência prematura e a rápida homogeneização da população; contudo, nas etapas seguintes de testes, percebeu-se que seria possível aumentar a diversidade da população sem a necessidade do reinício, apenas aumentado a quantidade de indivíduos. Nessa fase, foi estabelecida um total de avaliações de aptidão de 75.000, o que possibilitou que o desempenho das configurações testadas pudessem ser comparadas entre si e garantiu a rápida realização dos testes nesse primeiro momento, em que buscou-se identificar tendências gerais e comportamentos do AG no novo conjunto de quebra-cabeças.

Subsequentemente, na fase de Refinamento Simplificado, os parâmetros foram ajustados com maior precisão, visando otimizar a performance com base no desempenho percebido nos testes anteriores. Nesta etapa, 51 configurações foram testadas, contudo, diferente da primeira, poucas variações em torno dos operadores genéticos foram realizadas, de modo que o objetivo principal tornou-se o ajuste de parâmetros desses operadores. Desse modo, nesse momento foram testados diferentes tamanhos de população (valores entre 1500 até 5000), quantidades máximas de gerações (valores entre 60 até 200), bem como operadores (taxa inicial, mínima, máxima, proporção de limite inferior, proporção de limite superior e passo de ajuste) e parâmetros de mutação (apenas troca e troca com busca local do tipo *Min-Conflicts)*. Além disso, o total de avaliações de aptidão foi expandido para 300.000 para que fosse observada a estabilidade e robustez do AG em um espaço de busca amplificado.

Por fim, a fase de Refinamento Expandido concentrou-se em um ajuste fino e na validação das configurações mais promissoras em cenários de maior complexidade. Nessa etapa, 7 configurações foram testadas e o número de indivíduos totais avaliados foi significamente aumentado para 750.000, permitindo a exploração do espaço de buscas de forma mais exaustiva e o levantamento das configurações que apresentaram melhor performance e consistência na resolução dos quebra-cabeças mais complexos. Sendo assim, foram testados diferentes tamanhos de população (valores entre 5000 até 10000), quantidades máximas de gerações (valores entre 75 até 150), como também operadores (taxa inicial, mínima, máxima, proporção de limite inferior, proporção de limite superior e passo de ajuste) e parâmetros de mutação (apenas troca e troca com busca local do tipo *Min-Conflicts)*.

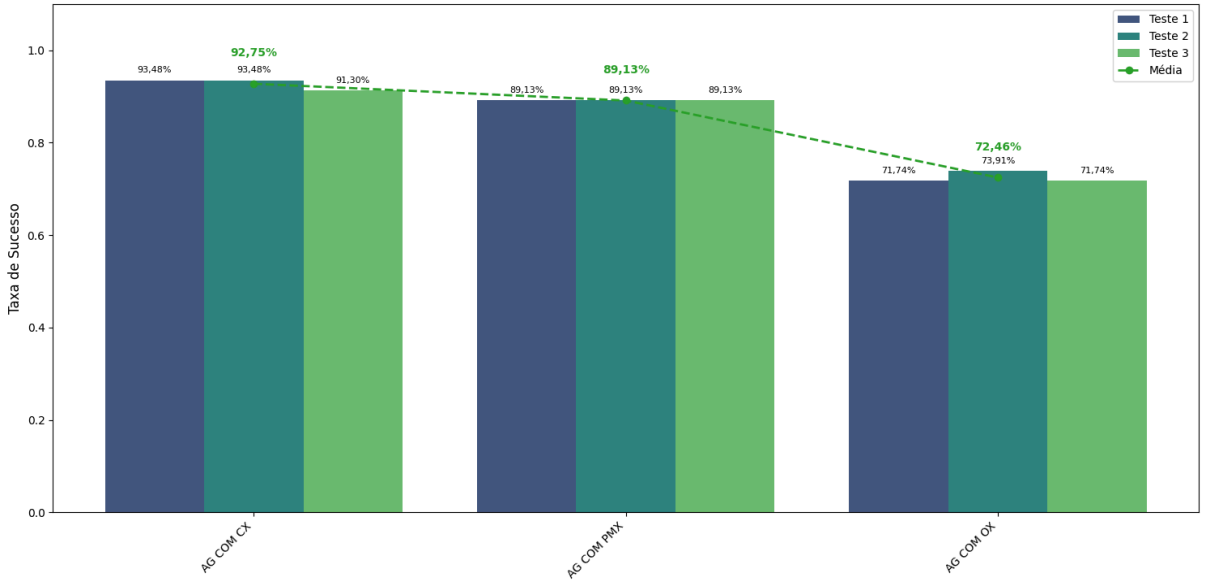
Ainda é fundamental destacar que, em todas as etapas de testes abordadas até o momento, desde o conjunto de quebra-cabeças inicial mais simples até os mais complexos, foram aplicadas na fase de pré-processamento as técnicas *Naked Single* (que também identifica o *Full House*), *Hidden Single* e *Naked Pairs*, devido ao fato de serem métodos simples, cujo implementação apresenta baixo custo computacional, capazes de reduzir significativamente a complexidade da maioria dos quebra-cabeças disponíveis. Após a escolha da configuração final, foi realizado um teste final complementar com diferentes operadores de cruzamento PMX, OX e CX, que haviam sido testados nas etapas iniciais do desenvolvimento ainda com o conjunto reduzido de cinco quebra-cabeças, mas que não haviam sido analisados em um conjunto mais completo. Por meio desse último teste foi possível identificar um aumento de cerca de 13% sem pré-processamento e 3% com pré-processamento na taxa de resolução com o uso do CX em comparação ao PMX; já com relação ao OX, o desempenho reduziu em quase 32% sem pré-processamento e 16% com pré-processamento, como visto nas Figuras 9 e 10, que correspondem ao desempenho do algoritmo com e sem pré-processamento, respectivamente. Baseado nesse resultado, o operador de cruzamento escolhido para a configuração final foi o CX.

Figura 9 - Taxas de sucesso obtidas nos três testes realizados com o AG sem pré-processamento utilizando os cruzamentos CX, PMX e OX.



Fonte: Autor (2025)

Figura 10 - Taxas de sucesso obtidas nos três testes realizados com o AG com pré-processamento utilizando os cruzamentos CX, PMX e OX.



Fonte: Autor (2025)

Ainda para auxiliar o Algoritmo Genético a obter uma maior taxa de sucesso na resolução foi incluída a técnica *X-Wings*. A sua adição não foi capaz de fazer com que o pré-processamento resolvesse isoladamente mais algum quebra-cabeça além dos 12 que já haviam sido solucionados apenas por ele, contudo passou a ser capaz de preencher algumas células vazias a mais em determinados quebra-cabeças, o que representou um avanço positivo considerando que Sudokus como o s09b e o s12a não haviam sido resolvidos por nenhuma configuração até o momento e passaram a ser solucionados em algumas das tentativas.

4.3. APRESENTAÇÃO DO ALGORITMO FINAL

O algoritmo final é o resultado de todos os testes empíricos realizados durante o desenvolvimento e refinamento que exploraram a influência dos operadores e parâmetros na capacidade do AG de convergir para uma solução ótima. Desse modo, a configuração final escolhida foi aquela que representou uma maior otimização, demonstrando um equilíbrio entre exploração do espaço de busca, pressão seletiva e a capacidade de superar ótimos locais. A seguir encontra-se uma explicação detalhada acerca do funcionamento das duas etapas principais que constituem o sistema, o pré-processamento e o Algoritmo Genético.

4.3.1. Etapa 1: Pré-Processamento com Técnicas Clássicas

A etapa de pré-processamento, neste solucionador, consiste na aplicação sistemática de um conjunto de técnicas de dedução lógica clássicas, com objetivo de simplificar o

quebra-cabeça sudoku antes da aplicação do Algoritmo Genético. Essas técnicas são aplicadas de forma determinística, buscando preencher o maior número de candidatos potenciais para as células vazias restantes. Assim, ao diminuir a complexidade do problema na fase inicial, a carga computacional para a etapa seguinte, em que o AG é aplicado, é minimizada, contribuindo para um processo de solução mais eficiente.

Para atingir esse objetivo, o módulo de pré-processamento incorpora várias estratégias já conhecidas de resolução de Sudoku. A seleção desses métodos baseou-se em sua eficácia na dedução lógica de números e na redução das listas de candidatos sem a necessidade de recorrer a tentativas e erros. Essas técnicas representam um equilíbrio entre a simplicidade de implementação e a capacidade de reduzir o espaço de busca. As principais técnicas escolhidas foram: *Naked Single, Hidden Single, Naked Pairs* e *X-Wings*.

No algoritmo do fluxograma da figura 7, a lógica de pré-processamento encontra-se encapsulada na classe *PreProcessing* (código *pre_processing.py*[2]). Nela, o método *get_candidates()* responsável por identificar o conjunto de candidatos possíveis para cada célula é chamado, considerando os números já presentes na linha, coluna e sub-bloco 3x3 correspondentes. O *loop* principal do pré-processamento encontra-se no método *preprocess()*, que aplica iterativamente o método *analyze_cell()* a cada célula vazia do Sudoku. O *analyze_cell()* busca por *Naked Singles*, *Naked Pairs* e *Hidden Singles* e caso um deles seja encontrado, o número é preenchido na célula vazia e/ou a lista de candidatos é atualizada, de acordo com o funcionamento da técnica. Enquanto isso, o *search_x_wing()* é aplicado após a análise célula a célula, ele é responsável por identificar *X-Wings*, identificando e eliminando candidatos. Durante esse processo, caso alguma modificação seja encontrada, as informações necessárias são atualizadas e o *looping* principal reinicia. Assim, todo o processo segue até que nenhuma outra alteração possa ser feita na grade, indicando que todas as deduções lógicas possíveis foram aplicadas.

4.3.2. Etapa 2: Algoritmo Genético

No fluxo padrão do solucionador, a segunda etapa, que corresponde ao Algoritmo Genético, é acionada apenas se o quebra-cabeça não for completamente resolvido pela Etapa 1 (pré-processamento). Nesse cenário, o papel do AG é atuar sobre o estado parcialmente preenchido do tabuleiro, que já contém tanto as pistas originais quanto os números inferidos na primeira parte, buscando os valores para as células que permaneceram vazias. De forma

[2] https://github.com/mariliagrcosta/evolutionary-sudoku-solver/blob/main/core/pre_processing.py

alternativa, o sistema oferece a flexibilidade de contornar a primeira etapa, permitindo que o usuário opte por aplicar o AG diretamente sobre o quebra-cabeça original. Os parágrafos seguintes descrevem detalhadamente o funcionamento do algoritmo.

A representação de cada solução potencial (indivíduo) no AG é implementada pela classe *Candidate* (código *individual.py*[3]), cuja estrutura foi projetada para refletir a abordagem flexível do sistema. A base dela é um array NumPy $9x9$, no qual as células são tratadas de duas formas diferentes:

- Células Fixas: Correspondem aos valores que o algoritmo não pode alterar. Este conjunto sempre inclui as pistas do problema original e, no caso da execução do fluxo híbrido, também engloba os números inferidos pela Etapa 1. Em ambos os cenários, estes valores são gerenciados pela classe *Fixed* (código *individual.py*[4]) para garantir sua imutabilidade;

- Células Variáveis: São as posições que permanecem vazias após a definição das células fixas, e portanto, são as únicas células que o AG otimiza, representando os genes de cada indivíduo na população.

A geração da população inicial é um passo fundamental para o bom desempenho da metaheurística e foi projetada para que cada indivíduo já satisfaça a restrição de linha, isto é, cada linha sempre conterá os números de um até nove sem duplicatas. Para isso, o algoritmo implementa um método elaborado para que esses indivíduos sejam devidamente criados em um processo que pode ser descrito da seguinte forma: uma lista de candidatos possíveis para cada célula vazia, considerando os valores fixos presentes nas linhas, colunas e sub-blocos correspondentes, é definida e cada espaço é preenchido com uma escolha aleatória dessa lista. Após o preenchimento, o algoritmo verifica se há duplicadas na linha e, em caso afirmativo, tenta atribuir novamente os valores com novas escolhas aleatórias para a linha inteira. Essa estratégia, apesar de mais custosa, é capaz de gerar uma população inicial de maior qualidade, o que resulta em uma taxa de sucesso consistentemente mais alta.

Vale ressaltar que, durante o desenvolvimento, uma abordagem alternativa mais simples e rápida foi considerada. Esse método consistia em listar os números de 1 a 9 que faltavam para completar a linha, embaralhar essas lista e usá-la para preencher, em ordem, as células vazias. Contudo, mesmo sendo mais feliz, a abordagem foi descartada pois, ao não

[3] https://github.com/mariliagrcosta/evolutionary-sudoku-solver/blob/main/core/individual.py
[4] Ibid.

considerar as restrições nas demais unidades, acaba gerando indivíduos iniciais de baixa qualidade e com maior dificuldade de convergir para uma solução válida, o que resultou em uma queda na taxa de sucesso.

Com relação à aptidão de cada indivíduo, ela é calculada pelo método *update_fitness() dentro da classe Candidate* (código *individual.py*[5]). A função de aptidão é projetada para avaliar o quão "boa" é a solução candidata, penalizando as violações das regras dos Sudoku, especialmente nas restrições de coluna e sub-bloco, já que as linhas foram projetadas para permanecerem válidas. Sendo assim, formalmente, tem-se: seja X a matriz $9x9$ que representa o indivíduo, para cada coluna j define-se U_j como o número de elementos distintos na coluna j, e para cada bloco b define-se V_b como o número de elemento distintos no bloco b. Então, com base nesses valores, calculam-se as somas $S_{col} = \sum_{j=1}^{9} U_J$ e $S_{blo} = \sum_{b=1}^{9} V_b$ que são normalizados por 81, obtendo $\overline{S}_{col} = \frac{S_{col}}{81}$ e $\overline{S}_{blo} = \frac{S_{blo}}{81}$, de modo que a aptidão final é o produto dessas quantidades:

$$f(X) = \overline{S}_{col} \times \overline{S}_{blo} = \frac{(\sum_{j=1}^{9} U_j)}{81} \times \frac{(\sum_{b=1}^{9} V_b)}{81} \quad (2)$$

Nesse contexto, conforme a definição formal do problema na Equação 1, uma solução ótima é encontrada ao minimizar a função $f(X)$, que representa o número de violações das restrições. No presente trabalho, essa lógica é adaptada: a função objetivo expressa pela Equação 1 é representada pela função de aptidão descrita na Equação 2. Desta forma, a busca por uma solução com zero violações ($f(X) = 0$) é traduzida como a maximização da função de aptidão, cujo valor pertence ao intervalo $f(X) \in (0, 1]$ e atinge seu máximo de 1,0 somente para indivíduos que satisfazem integralmente as restrições de colunas e blocos, isto é, correspondem a uma solução válida.

O ciclo evolutivo do Algoritmo Genético, gerenciado pelo método, *solve()* na classe *Sudoku* (código *solver.py*[6]), refina iterativamente a população de soluções candidatas ao longo de um número máximo pré definido de gerações, *MAX_GENERATIONS*. Esse ciclo envolve a aplicação de vários operadores genéticos fundamentais: seleção de pais, cruzamento, mutação e estratégia de sobrevivência.

[5] https://github.com/mariliagrcosta/evolutionary-sudoku-solver/blob/main/core/individual.py
[6] https://github.com/mariliagrcosta/evolutionary-sudoku-solver/blob/main/core/solver.py

A seleção de pais é realizada utilizando um mecanismo de Seleção por Torneio, implementado pela classe *Tournament* (código *genetic_operators.py*[7]). Em cada evento de seleção, uma dupla de candidatos é selecionada aleatoriamente da população atual, e o indivíduo com maior aptidão entre eles é selecionado como pai. A escolha desse método se baseou na sua capacidade de introduzir um grau de pressão seletiva, ao mesmo tempo que preserva certa diversidade na população.

A operação de cruzamento é implementada pela classe *CXCrossover* (código *genetic_operators.py*[8]), e aplica o operador Cycle Crossover (CX). O cruzamento CX foi escolhido por se mostrar adequado para problemas baseado em permutação, como o Sudoku, por não resultar em descendentes inválidos, com valores duplicados e/ou ausentes, que violariam as restrições fundamentais do problema. Então, para cada linha dos pais selecionados, o método *cx_row_segment()* é chamado, o qual identifica "ciclos" entre as duas linhas parentais. Os valores fixos do Sudoku, que são os números iniciais do quebra-cabeça, são inerentemente preservados durante esse processo, pois eles ocupam a mesma posição em ambos os pais, considerando que as soluções candidatas são inicializadas com esses valores inalterados. Consequentemente, ao identificar os ciclos e gerar os valores para a linha do descendente, o CX garante que esses valores fixos mantenham sua posição original, pois eles não participam das trocas dentro dos ciclos. Uma vez que os "ciclos" são identificados, os valores para a linha do descendentes herdam características de ambos os pais mantendo permutações válidas dentro de cada linha.

Já a mutação é tratada pela função *mutate()* (código *genetic_operators.py*[9]) e emprega uma estratégia de Mutação por Troca. Este operador é responsável por introduzir variações aleatórias no indivíduo, evitando a convergência prematura e ajudando na exploração de novas regiões do espaço de busca. Baseado em uma determinada taxa de mutação, é analisado se um indivíduo sofrerá mutação ou não. Caso ele seja selecionado, uma linha é escolhida de forma aleatória. Então, para a linha selecionada, duas posições mutáveis são escolhidas também de maneira aleatória, e os valores nessas duas posições são então trocados. Devido ao seu modo de funcionamento, esta operação garante que a linha permaneça sendo uma permutação válida após a sua realização.

[7] https://github.com/mariliagrcosta/evolutionary-sudoku-solver/blob/main/core/genetic_operators.py
[8] Ibid.
[9] Ibid.

A taxa de mutação, *mutation_rate*, apresenta uma valor inicial, *INITIAL_MUTATION_RATE*, contudo é ajustada adaptativamente a cada geração com base na aptidão mediana da população. O funcionamento dessa adaptação ocorre da seguinte maneira: o algoritmo calcula a aptidão máxima, *max_fitness*, e a aptidão mediana, *median_fitness,* da população atual e a partir dessas valores define os limites superior, *upper_bound*, e inferior, *lower_bound*, para a aptidão mediana, que são adaptativos a performance geral da população através de um redutor de amplitude, *amplitude_reduction*. Então, caso *median_fitness* exceder o *upper_bound,* indicando homogeneidade na população e risco de estagnação, o *mutation_rate* é aumentado em um passo, segundo a variável *MUTATION_RATE_ADJUSTMENT_STEP*, estimulando a exploração. Por outro lado, se *median_fitness* cai abaixo do *lower_bound*, sugerindo que a população está muito dispersa, a *mutation_rate* é diminuída no mesmo passo, incentivando a explotação de soluções promissoras. Entretanto, a taxa de mutação é sempre contida entre um valor mínimo, *MIN_MUTATION_RATE* , e máximo, *MAX_MUTATION_RATE*, para garantir estabilidade no processo evolutivo.

Após a geração dos descendentes, o algoritmo determina quais indivíduos formarão a próxima geração, através de uma estratégia de sobrevivência com elitismo, de modo que uma porcentagem dos indivíduos de melhor desempenho da população atual é diretamente transferida para a próxima, garantindo que as melhores possíveis soluções encontradas até o momento não sejam perdidas entre as gerações. Os demais espaços, além daqueles ocupados pelos indivíduos de elite, são preenchidos selecionando-se a partir de um grupo que combina os indivíduos não-elite da população atual e todos os descendentes recém gerados. Para a seleção dos indivíduos nesse conjunto é aplicado uma Seleção por Torneio, semelhante à utilizada na seleção dos pais, de modo que o melhor indivíduo entre os dois escolhidos é removido da *pool* e segue para a próxima geração.

4.3.3. Configuração Final do Algoritmo

A configuração final do algoritmo consolida, especialmente, os operadores e parâmetros utilizados na etapa do Algoritmo Genético, cujas justificativas e mecanismos foram detalhados nas seções anteriores. Essa configuração representa o melhor equilíbrio otimizado entre exploração e explotação que foi alcançado através dos testes empíricos realizados. Dessa maneira, esses valores encontram-se consolidados na Quadro 1.

Quadro 1 - Configuração final dos operadores e parâmetros do Algoritmo Genético

Categoria	Operador e/ou parâmetro	Estratégia e/ou valor
Geral	Tamanho da população (POPULATION_SIZE)	10.000 indivíduos
	Máximo de gerações (MAX_GENERATIONS)	100 gerações
	Total de avaliações de aptidão	1.000.000 (10.000 indivíduos * 100 gerações)
Seleção de pais	Mecanismo de seleção	Torneio
	Tamanho do torneio (TOURNAMENT_SIZE)	2
Cruzamento	Operador de cruzamento	Cycle Crossover (CX)
Mutação com taxa adaptativa	Estratégia de mutação	Mutação por Troca (Swap Mutation)
	Taxa de mutação inicial (INITIAL_MUTATION_RATE)	0,06
	Taxa de mutação mínima (MIN_MUTATION_RATE)	0,01
	Taxa de mutação máxima (MAX_MUTATION_RATE)	0,40
	Proporção limite superior de aptidão mediana (MEDIAN_FITNESS_UPPER_BOUND_RATIO)	0,70
	Proporção limite inferior de aptidão mediana (MEDIAN_FITNESS_LOWER_BOUND_RATIO)	0,50
	Passo de ajuste da mutação (MUTATION_RATE_ADJUSTMENT_STEP)	0,005
Seleção de sobreviventes	Mecanismo de seleção	Conjunto combinado de pais e filho (Elitismo + Torneio)
	Percentual de elite (ELITE_PERCENTAGE)	0,01 (1%)

Fonte: Autor (2025)

5. RESULTADOS

Esta seção dedica-se a apresentar e analisar os resultados experimentais obtidos com a aplicação do algoritmo híbrido proposto. O principal objetivo dessa fase foi a avaliação empírica do desempenho do solucionador e o entendimento do impacto da etapa de pré-processamento na otimização da resolução de quebra-cabeças Sudoku.

Para a realização desses testes foi utilizado o conjunto de 46 instâncias de Sudoku apresentados em Mantere e Kolkonen (7), que abrangem diferentes níveis de dificuldade com

características diversas não apenas relacionadas à quantidade de dicas informadas, mas à simetria do quebra cabeça. Semelhantemente à metodologia descrita na etapa de ajuste e refinamento de parâmetros, cada instância foi executada nove vezes, o que representa um aumento de seis execuções em comparação com o refinamento, em duas modalidades distintas: com pré-processamento e sem pré-processamento, totalizando 552 execuções para cada abordagem.

Os dados gerais demonstram uma superioridade expressiva da abordagem híbrida, considerando que o sistema com pré-processamento alcançou uma taxa de sucesso de 94,56%, resolvendo 522 execuções de 552, o que corresponde a um aumento de mais de 28,8% em comparação com a taxa de 65,76% do Algoritmo Genético, como pode ser visto na Tabela 2.

Tabela 2 - Desempenho do Algoritmo Genético com e sem a etapa de pré-processamento.

Algoritmo	Total de execuções	Não resolvidos	Resolvido apenas com pré-processamento	Resolvido com algoritmo genético	Sucesso total
PRÉ-PROCESSAMENTO + ALGORITMO GENÉTICO	552	30 (5,43%)	408 (73,91%)	114 (20,65%)	522 (94,56%)
ALGORITMO GENÉTICO	552	189 (34,24%)	0	363 (65,76%)	363 (65,76%)

Fonte: Autor (2025)

5.1. ANÁLISE DA EFICÁCIA DO PRÉ-PROCESSAMENTO

A primeira etapa do algoritmo híbrido, responsável por realizar a aplicação das estratégias lógicas de resolução, provou ser fundamental para o desempenho geral do sistema, pois conforme mostrado na Tabela 2, das 552 execuções realizadas com o algoritmo híbrido, 408 foram completamente solucionadas apenas na fase de pré-processamento, o que corresponde a cerca de 73,91% do total. Esse fato indica que 34 dos 46 quebra-cabeças do conjunto, classificados em suas fontes como de dificuldade fácil a difícil foram resolvidos de forma determinística com tempo médio de 0,01 segundos.

Agora, com relação aos quebra-cabeças mais complexos que não foram completamente resolvidos nesta fase, a etapa de pré-processamento ainda desempenhou um papel importante ao reduzir significativamente o espaço de busca. Isso pode ser percebido analisando a Tabela 3, que indica para cada quebra-cabeça não solucionado de forma

autônoma pela primeira etapa, qual é a quantidade de células que foram preenchidas. Na instância s09b, por exemplo, que é considerada como difícil, foram complementadas 13 das 59 vazias, enquanto que em s15c foram 24 das 57, o que permitiu que o AG operasse em um problema de menor complexidade, otimizando sua capacidade de convergência.

Tabela 3 - Quantidade de células preenchidas pela etapa de pré-processamento nos quebra-cabeças em que não foram solucionados apenas por ela

Quebra-cabeça	Dicas	Células preenchidas pelo pré-processamento	Células vazias restantes
s04a.txt	28	18	35
s04c.txt	28	6	47
s05b.txt	28	6	47
s05c.txt	26	18	37
s09a.txt	23	13	45
s09b.txt	22	13	46
s09c.txt	22	26	33
s12b.txt	26	13	42
s15a.txt	27	25	29
s15b.txt	27	9	45
s15c.txt	27	24	30
s16.txt	27	1	53

Fonte: Autor (2025)

5.2. ANÁLISE COMPARATIVA: ABORDAGEM HÍBRIDA VS. ALGORITMO GENÉTICO

A eficácia da abordagem híbrida torna-se ainda mais evidente ao analisar os casos em que o algoritmo híbrido foi necessário, ou seja, nos quebra-cabeças que não foram inteiramente resolvidos pela etapa de pré-processamento. A integração de estratégias lógicas

não apenas elevou a taxa de sucesso de forma substancial, mas também acelerou a busca pela solução, otimizando tanto o tempo quanto os recursos computacionais. Alguns dos benefícios observados foram:

- Capacidade de Solucionar *Puzzles* Inviáveis para o AG: O impacto mais notável do pré-processamento foi capacitar o AG a resolver instâncias que se mostravam consistentemente insolúveis para ele. O *puzzle* s15a, por exemplo, não foi resolvido em nenhuma das 12 tentativas do AG. Contudo, na abordagem híbrida, com o preenchimento de 25 células e aplicação subsequente do AG, o algoritmo conseguiu encontrar a solução em 100% das execuções. Um comportamento similar ocorreu com o s05c, cuja taxa de sucesso saltou de 16,67% com AG para 100% com abordagem híbrida.
- Aceleração da Convergência: Mesmo nos casos em que o AG sozinho foi capaz de encontrar a solução, a abordagem híbrida demonstrou ser significativamente mais eficiente. No quebra-cabeça s04c, por exemplo, a média das gerações, nos casos em que ele foi resolvido apenas pelo AG, foi de 21,38 para 12,62 gerações, o que representa uma redução de mais de 8000 avaliações de aptidão de indivíduos. De forma semelhante, para a instância 12b.txt a redução na média de geração nos casos resolvidos foi de 19,25 para 11,42.

As Tabelas 4 e 5 evidenciam o desempenho do AG e do algoritmo híbrido, respectivamente, na resolução dos mesmos quebra-cabeças, detalhando a taxa de sucesso e a média de gerações necessárias para o alcance de solução, nos casos em que ela pode ser atingida sem o pré-processamento.

Tabela 4 - Desempenho do AG na resolução de alguns dos quebra-cabeças do conjunto de 46 *puzzles* em 12 tentativas.

(continua)

Quebra-cabeça	Tentativas de resolução	Quantidade de sucessos obtidos	Quantidade mínima de gerações	Quantidade máxima de gerações	Média de gerações	Taxa de sucesso
s04a.txt	12	2	14	20	17,00	16,67%
s04c.txt	12	8	13	35	22,38	66,67%
s05b.txt	12	0	0	0	0,00	0,00%

Tabela 4 - Desempenho do AG na resolução de alguns dos quebra-cabeças do conjunto de 46 *puzzles* em 12 tentativas.

(conclusão)

Quebra-cabeça	Tentativas de resolução	Quantidade de sucessos obtidos	Quantidade mínima de gerações	Quantidade máxima de gerações	Média de gerações	Taxa de sucesso
s05c.txt	12	2	21	26	23,50	16,67%
s09a.txt	12	0	0	0	0,00	0,00%
s09b.txt	12	2	25	27	26,00	16,67%
s09c.txt	12	6	16	29	22,17	50,00%
s12b.txt	12	8	16	29	20,25	66,67%
s15a.txt	12	0	0	0	0,00	0,00%
s15b.txt	12	4	21	27	23,50	33,33%
s15c.txt	12	1	34	34	34,00	8,33%
s16.txt	12	0	0	0	0,00	0,00%

Fonte: Autor (2025)

Tabela 5 - Desempenho do algoritmo híbrido na resolução de alguns dos quebra-cabeças do conjunto de 46 *puzzles* em 12 tentativas.

(continua)

Quebra-cabeça	Tentativas de resolução	Quantidade de sucessos obtidos	Quantidade mínima de gerações	Quantidade máxima de gerações	Média de gerações	Taxa de sucesso
s04a.txt	12	12	1	7	3,67	100,00%
s04c.txt	12	8	11	18	13,62	66,67%
s05b.txt	12	11	9	21	13,82	91,67%
s05c.txt	12	12	6	28	12,75	100,00%

Tabela 5 - Desempenho do algoritmo híbrido na resolução de alguns dos quebra-cabeças do conjunto de 46 *puzzles* em 12 tentativas.

(conclusão)

Quebra-cabeça	Tentativas de resolução	Quantidade de sucessos obtidos	Quantidade mínima de gerações	Quantidade máxima de gerações	Média de gerações	Taxa de sucesso
s09a.txt	12	7	10	17	13,43	58,33%
s09b.txt	12	7	11	16	13,71	58,33%
s09c.txt	12	12	4	8	6,42	100,00%
s12b.txt	12	12	10	15	12,42	100,00%
s15a.txt	12	12	3	15	06,08	100,00%
s15b.txt	12	11	12	27	16,27	91,67%
s15c.txt	12	10	2	18	9,50	83,33%
s16.txt	12	0	0	0	0,00	0,00%

Fonte: Autor (2025)

5.3. ANÁLISE DOS CASOS DE FALHA

Apesar do elevado índice de sucesso, o algoritmo não foi capaz de resolver todas as instâncias em todas as execuções, tendo as falhas se concentrado, em linha com o foco do projeto, na solução de quebra-cabeças de nível médio e difícil, nos *puzzles* de mais alta complexidade. O maior exemplo dessa situação é o quebra-cabeça s16, conhecido como “AI Escargot”, que é considerado um dos mais difíceis (7) e não foi solucionado em nenhuma das 12 tentativas totais, tanto com a aplicação das técnicas de resolução antes do Algoritmo Genético, quanto sem. No caso dele, a etapa de pré-processamento foi capaz de preencher apenas 1 das 58 células vazias, de modo que o espaço de busca remanescente mostrou-se demasiadamente complexo para que o AG convergisse para a solução ótima no limite de gerações estabelecidos, o que pode indicar que seria necessário uma quantidade de gerações maior para que a meta-heurística fosse capaz de chegar em uma resolução.

Outras instâncias, como s04c e s15b, também apresentaram falhas pontuais, mesmo na abordagem híbrida. Para esses casos, infere-se que o algoritmo pode ter convergido prematuramente para ótimos locais, dos quais não conseguiu escapar. Essa situação se relaciona profundamente à natureza estocástica dos operadores genéticos que implica que, embora a probabilidade de encontrar soluções sejam consideráveis, ela não é garantida em todas as execuções, especialmente em cenários de alta complexidade.

Em suma, os casos de falhas foram pontuais e percebidos em quebra-cabeças de maior complexidade, como o AI Escargot. Considerando que o foco do projeto era desenvolver uma ferramenta para Sudokus de nível médio e difícil, as falhas observadas nas instâncias mais difíceis estão alinhadas com as expectativas traçadas. Isso se deve não apenas a dificuldade intrínseca desses desafios, mas também a natureza do AG, que não garante a convergência para a solução ótima em todas as execução, fazendo com que os resultados obtidos estejam dentro do esperado.

5.4. COMPARAÇÃO COM OUTROS ESTUDOS

Torna-se muito difícil comparar o desempenho do algoritmo proposto devido às diferenças de Sudokus testados, que mesmo indicados como do mesmo nível podem representar desafios diferentes para o sistema, e as especificidades de *hardware* utilizados durante os testes que ofertam capacidades distintas de processamento. Contudo, ainda é possível identificar similaridades e diferenças na implementação e interpretar o quão próximo os resultados se encontram.

O estudo de Indriyono et al. (5) busca comparar o resultado do método *backtracking* com o Algoritmo Genético na resolução de Sudoku. Nele foram testados 10 quebra-cabeças de diferentes níveis que apresentavam de 43 a 45 células vazias e, a partir dos resultados obtidos, os autores concluíram que o número de casas vazias não é diretamente proporcional ao tempo necessário para resolução ou à quantidade de processo *backtracking*. Também foi observado que houve uma diferença no tempo de conclusão, considerando que em um nível difícil, o algoritmo de *backtracking* apresentou um tempo relativamente mais rápido do que o algoritmo genético. Entretanto, não é possível ter certeza se os resultados atingidos seriam os mesmos ou se os algoritmos obteriam êxito na resolução, caso o desempenho desses algoritmos fosse averiguado em um conjunto de testes mais complexo, semelhante ao utilizado no presente trabalho. Além disso, como os quebra-cabeças não foram disponibilizados, não é possível averiguar a capacidade de resolução do algoritmo híbrido

proposto neste trabalho, contudo, baseado na quantidade de dicas fornecidas e no desempenho da solução em desafios de mesmo porte, pode-se inferir que o algoritmo proposto pode ser capaz de resolver de forma autônoma com pré-processamento ou ao menos reduzir consideravelmente o espaço de busca a ser explorado pelo AG.

Já a pesquisa de Diah et al. (4) realizou uma análise comparativa entre as abordagens de busca em largura (BFS), busca em profundidade (DFS) e a resolução humana para quebra-cabeças Sudoku. Nela, os *puzzles* foram selecionados com base na quantidade de dicas, variando entre 32 e 35, e os resultados obtidos na resolução desses desafios indicaram que o DFS foi significativamente mais rápido e eficiente que o BFS. Ademais, também observou-se que quando a solução obtida pelos algoritmos coincidia com a solução humana, isso sugere que o Sudoku possuía uma solução única. Em comparação com o trabalho proposto, a solução de Diah et al. (4) focou em um nível de dificuldade que muitas vezes foi resolvido rapidamente de forma autônoma pelo pré-processamento do sistema apresentado neste estudo ou, pelo menos, foi consideravelmente simplificado por ele. Desse modo, não é possível analisar ou comparar a robustez dessa estratégia na resolução de quebra-cabeças mais complexos, pois não foram feitos testes com desafios maiores, como alguns dos presentes no conjunto de 46 Sudokus.

Com relação ao trabalho de Wang et al. (31) foi proposto um Algoritmo Genético aprimorado com busca local (LSGA) para resolver Sudokus, incorporando uma codificação baseada em matriz, uma nova estratégia de busca local por coluna e sub-bloco, e um mecanismo de aprendizado de elite. Esse sistema demonstrou alta performance, resolvendo 100% dos quebra-cabeças difíceis testados e, em alguns casos, exigindo uma média inferior de gerações para encontrar a solução em comparação a outras soluções com as quais ele foi comparado. Notavelmente, o LSGA foi capaz de resolver o “AI Escargot” em suas tentativas, o que o destaca como uma solução promissora. Ao compará-lo com a abordagem proposta neste estudo, percebe-se que ambos buscam eficiência na resolução de Sudokus complexos, contudo enquanto Wang et al. (31) focam em aprimoramentos internos do Algoritmo Genético, o presente projeto utiliza um pré-processamento lógico que busca para simplificar o problema antes da aplicação do AG.

Por fim, a proposta de Amil e Mantere (13) investigou a otimização na resolução de Sudokus por algoritmos evolutivos (Algoritmos Genéticos, Algoritmos Culturais e híbridos de AG/Otimização por Colônia de Formigas), através da aplicação de um pré-processamento

inicial com métodos lógicos de resolução, de modo semelhante a esta pesquisa. Os resultados obtidos por eles mostram que essa etapa foi capaz de solucionar completamente alguns dos Sudokus mais fáceis e, para os mais difíceis, reduziu drasticamente o número de posições a serem preenchidas, acelerando de forma significativa a solução pelos algoritmos evolutivos. Neste contexto, o trabalho proposto neste estudo se alinha diretamente com a abordagem proposta por Amil e Mantere (13), pois partem da mesma ideia para otimizar o encontro da solução, mesmo que através da aplicação de diferentes métodos. Sendo assim, foi possível reforçar as conclusões obtidas pelos autores acerca da eficiência combinação de deduções lógicas e métodos evolutivos, além de apresentar o impacto dessa etapa no processo de solução de forma quantitativa.

6. CONSIDERAÇÕES FINAIS

O presente trabalho se propôs a desenvolver e avaliar um algoritmo evolutivo híbrido na otimização de quebra-cabeças Sudoku. A solução final integrou uma etapa de pré-processamento, constituída por técnicas lógicas de revolução como *Naked Singles*, *Hidden Singles*, *Naked Pairs* e *X-Wi*ngs, a um Algoritmo Genético.

Os resultados experimentais consolidados a partir de um conjunto de 12 execuções para cada um dos 46 quebra-cabeças, validaram com sucesso a hipótese de que a abordagem híbrida é significativamente mais eficaz que apenas o Algoritmo Genético. O sistema desenvolvido alcançou uma taxa de sucesso global de 91,06%, superando o desempenho do AG em mais de 25%. A análise dos resultados demonstrou que o pré-processamento não apenas resolveu de forma autônoma 73,91% dos *puzzles* testados, mas também refinou o espaço de busca de maneira a tornar a convergência do AG mais rápida, necessitando de uma quantidade menor de gerações. Este desempenho encontra-se em concordância com a proposta do trabalho, que previa foco na resolução de *puzzles* de nível médio e difícil, com as falhas se concentrando, como esperado, nos casos de complexidade mais elevada.

Diante do que foi apresentado, pode-se afirmar que os objetivos delineados para esse trabalho foram satisfatoriamente atendidos e que as contribuições deste projeto residem na concepção do modelo híbrido, na análise empírica que quantifica seus ganhos de desempenho e na disponibilização do algoritmo em código aberto.

Ademais, é fundamental ressaltar que esta pesquisa possui uma dupla relevância que vai além da simples resolução do quebra-cabeça. Primeiramente, o Sudoku serve como um

benchmark ideal para a validação de algoritmos de otimização, já que por se tratar de um problema NP-completo em um ambiente controlado, permite que algoritmos, como o híbrido aqui proposto, sejam testados e refinados antes de serem adaptados e aplicados para problemas do mundo real de maior complexidade e incerteza. Em segundo lugar, a própria estrutura do Sudoku possui aplicações diretas e práticas em diferentes áreas, de modo que alguns exemplos notáveis incluem: uso como ferramenta pedagógica para o desenvolvimento do raciocínio lógico e de habilidades de resolução de problemas na educação (14); otimização do arranjo de painéis fotovoltaicos para redução do sombreamento mútuo (16) e o design de formas de onda de radar (17) na engenharia; uso como base em técnicas de esteganografia para segurança da informação (15). Dessa forma, fica evidente que a importância do estudo aprofundado de Sudokus vai além da resolução de quebra-cabeças, pois atua tanto como um laboratório para o avanço de algoritmos quanto como uma fonte de inspiração para soluções tecnológicas práticas.

Por fim, com base nos resultados atingidos e nas limitações identificadas foi possível traçar algumas direções para a continuidade da pesquisa:

- Melhoria na estratégia de criação da população inicial de candidatos: A elaboração de uma estratégia de criação da população inicial de candidatos eficaz, que apresenta uma capacidade igual ou superior à abordagem atual de solução, mas que seja menos custosa e garanta sucesso.
- Realização de testes mais aprofundados utilizando o cruzamento Cycle Crossover: Os testes desenvolvidos no presente trabalho focaram, especialmente, no operador de cruzamento PMX, contudo, diante do desempenho satisfatório percebido nos testes realizados com o CX, torna-se válida a realização de novos testes com diferentes parâmetros, a fim de averiguar o impacto deste operador em comparação com os demais métodos.
- Análise de estratégias para contornar ótimos locais: Considerando a dificuldade enfrentada pelo algoritmo em contornar ótimos locais, sobretudo, em *puzzles* de maior complexidade, torna-se necessário a análise e o desenvolvimento de estratégias capazes de perceber e contornar ótimos locais, guiando o AG ao encontro de um ótimo global, em problemas permutacionais de alta dimensionalidade e com restrições, como os abordados no presente trabalho.

REFERÊNCIAS

1 FELGENHAUER, B.; JARVIS, F. Mathematics of Sudoku I. **Mathematical Spectrum**, v. 39, 2006.

2 MCGUIRE, G.; TUGEMANN, B.; CIVARIO, G. There is no 16-clue Sudoku: solving the Sudoku minimum number of clues problem via hitting set enumeration. **Experimental Mathematics**, v. 23, n. 2, p. 190–217, 2014. DOI: https://doi.org/10.1080/10586458.2013.870195. Acesso em: jul. 2025.

3 MANTERE, T.; KOLJONEN, J. Solving, rating and generating Sudoku puzzles with GA. In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, 2007, Singapore. **Anais eletrônicos** [...]. Singapore: IEEE, 2007. p. 1382-1389. DOI: 10.1109/CEC.2007.4424632. Acesso em: jul. 2025.

4 DIAH, N. M. et al. Sudoku solutions: a comparative analysis of breadth-first search, depth-first search, and human approaches. **International Journal of Evaluation and Research in Education (IJERE)**, v. 19, n. 1, 2025. DOI: https://doi.org/10.11591/edulearn.v19i1.21214. Acesso em: jul. 2025.

5 INDRIYONO, B. et al. Comparative analysis of the performance testing results of the backtracking and genetics algorithm in solving Sudoku games. **International Journal of Artificial Intelligence & Robotics (IJAIR)**, v. 5, n. 1, p. 29–35, 2023. DOI: https://doi.org/10.25139/ijair.v5i1.6501. Acesso em: jul. 2025.

6 THENMOZHI, M. et al. Analysis of Sudoku solving algorithms. **International Journal of Engineering and Technology**, v. 9, n. 3, p. 1745–1749, 2017. DOI: https://doi.org/10.21817/ijet/2017/v9i3/170903043. Acesso em: jul. 2025.

7 MANTERE, T.; KOLJONEN, J. Solving and analyzing Sudokus with cultural algorithms. In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, 2008, Hong Kong. **Anais eletrônicos** [...]. Hong Kong: IEEE, 2008. p. 4053–4060. DOI: https://doi.org/10.1109/CEC.2008.4631350. Acesso em: jul. 2025.

8 LLOYD, H. **Solving Sudoku with Ant Colony Optimization**. 2005. Dissertação (Mestrado em Inteligência Artificial) – School of Informatics, University of Edinburgh, Edimburgo, 2005.

9 MANTERE, T. Improved ant colony genetic algorithm hybrid for Sudoku solving. In: WORLD CONGRESS ON INFORMATION AND COMMUNICATION TECHNOLOGIES (WICT), 3., 2013, Hanói. **Anais eletrônicos** [...]. Hanói: IEEE, 2013. p. 274–279. DOI: https://doi.org/10.1109/WICT.2013.7113148. Acesso em: jul. 2025.

10 AARONSON, L. Sudoku science. **IEEE Spectrum**, v. 43, n. 2, p. 16–17, fev. 2006. DOI: https://doi.org/10.1109/MSPEC.2006.1584356. Acesso em: jul. 2025.

11 YATO, T.; SETA, T. Complexity and completeness of finding another solution and its application to puzzles. **IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences**, v. E86-A, n. 5, p. 1222–1230, 2003.

12 LYNCE, I.; OUAKNINE, J. Sudoku as a SAT problem. In: INTERNATIONAL SYMPOSIUM ON ARTIFICIAL INTELLIGENCE AND MATHEMATICS, 9., 2006, Fort Lauderdale. **Anais** [...]. Fort Lauderdale: [s.n.], 2006.

13 AMIL, Pedro Redondo; MANTERE, Timo. Solving sudoku's by evolutionary algorithms with pre-processing. In: MATOUŠEK, Radek (ed.). **Recent advances in soft computing**: proceedings of 23rd International Conference on Soft Computing (MENDEL 2017), held in Brno, Czech Republic, June 20-22, 2017. Cham: Springer, 2019. p. 3-15. (Advances in Intelligent Systems and Computing, v. 837). DOI: https://doi.org/10.1007/978-3-319-97888-8_1. Acesso em: jul. 2025.

14 JOSE, S.; ABRAHAM, R. Influence of chess and Sudoku on cognitive abilities of secondary school students. **Issues and Ideas in Education**, v. 7, n. 1, p. 27–34, 2019. DOI: https://doi.org/10.15415/iie.2019.71004. Acesso em: jul. 2025.

15 JANA, S.; MAJI, A. K.; PAL, R. K. A novel SPN-based video steganographic scheme using Sudoku puzzle for secured data hiding. **Innovations in Systems and Software Engineering**, v. 15, p. 65–73, 2019. DOI: https://doi.org/10.1007/s11334-019-00324-8. Acesso em: jul. 2025.

16 HOROUFIANY, M.; GHANDEHARI, R. Optimization of the Sudoku based reconfiguration technique for PV arrays power enhancement under mutual shading conditions. **Solar Energy: The Official Journal of the International Solar Energy Society**, v. 159, p. 103–113, 2017. DOI: https://doi.org/10.1016/j.solener.2017.05.059. Acesso em: jul. 2025.

17 BUFLER, T. D.; NARAYANAN, R. M.; SHERBONDY, K. D. Sudoku inspired designs for radar waveforms and antenna arrays. **Electronics**, v. 6, n. 1, p. 13, 2017. DOI: https://doi.org/10.3390/electronics6010013. Acesso em: jul. 2025.

18 WEAVER, C. **Strategies and Algorithms of Sudoku**. 2020. Trabalho de Conclusão de Curso (Bacharelado em Matemática) – Louisiana Tech University, Louisiana, 2020. Disponível em: https://digitalcommons.latech.edu/mathematics-senior-capstone-papers/22. Acesso em: jul. 2025.

19 DAVIS, T. The mathematics of Sudoku. In: SHUBIN, T.; HAYES, D. F.; ALEXANDERSON, G. L. (org.). **Expeditions in Mathematics**. Washington, D.C.: Mathematical Association of America, 2011. p. 31–59.

20 EIBEN, A. E.; SMITH, J. E. **Introduction to Evolutionary Computing.** 2. ed. Cham: Springer, 2015.

21 HOLLAND, J. H. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence**. Cambridge: The MIT Press, 1992. DOI: https://doi.org/10.7551/mitpress/1090.001.0001. Acesso em: jul. 2025.

22 GOLDBERG, David E.; LINGLE, Robert. Alleles, Loci, and the Traveling Salesman Problem. In: GREFENSTETTE, John J. (ed.). **Proceedings of the First International Conference on Genetic Algorithms and their Applications.** New York: Psychology Press, 1985. p. 154-159. DOI: https://doi.org/10.4324/9781315799674. Acesso em: jul. 2025.

23 DAVIS, L. **Handbook of Genetic Algorithms**. New York: Van Nostrand Reinhold, 1991.

24 RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 4. ed. Hoboken: Pearson, 2020.

25 PAPADIMITRIOU, C.; STEIGLITZ, K. **Combinatorial Optimization: Algorithms and Complexity**. Englewood Cliffs, NJ: Prentice-Hall, 1982.

26 HOOS, H. H.; STÜTZLE, T. **Stochastic Local Search: Foundations and Applications**. Burlington: Morgan Kaufmann Publishers, 2004.

27 HOOS, Holger H.; STÜTZLE, Thomas. Stochastic Local Search Algorithms: An Overview. In: KACPRZYK, Janusz; PEDRYCZ, Witold (Eds.). **Springer Handbook of Computational Intelligence**. Berlin, Heidelberg: Springer, 2015. p. 1085-1105. DOI: https://doi.org/10.1007/978-3-662-43505-2_54. Acesso em: jul. 2025.

28 GU, J. **Parallel Algorithms and Architectures for Very Fast AI Search**. 1989. Tese (Doutorado em Ciência da Computação) – University of Utah, Utah, 1989.

29 MINTON, S. et al. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. **Artificial Intelligence**, v. 58, n. 1–3, p. 161–205, 1992. DOI: https://doi.org/10.1016/0004-3702(92)90007-K. Acesso em: jul. 2025.

30 MINTON, S. et al. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 8., 1990, Boston. **Anais** [...]. Boston: AAAI Press, 1990. v. 1, p. 17–24.

31 WANG, C. et al. A novel evolutionary algorithm with column and sub-block local search for Sudoku puzzles. **IEEE Transactions on Games**, v. 16, n. 1, p. 162–172, mar. 2024. DOI: https://doi.org/10.1109/TG.2023.3236490. Acesso em: jul. 2025.