UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS DE ARAÚJO TORRES

ALGORITMOS GENÉTICOS PARA O PROBLEMA DA DOMINAÇÃO ROMANA 2-FORTE EM GRAFOS

QUIXADÁ

2026

LUCAS DE ARAÚJO TORRES

ALGORITMOS GENÉTICOS PARA O PROBLEMA DA DOMINAÇÃO ROMANA 2-FORTE EM GRAFOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Atílio Gomes Luiz.

QUIXADÁ

2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

T646a Torres, Lucas.
Algoritmos Genéticos para o Problema de Dominação Romana 2-Forte em Grafos / Lucas Torres. – 2026.
78 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2026.
Orientação: Prof. Dr. Atílio Gomes Luiz.

1. otimização. 2. grafos. 3. algoritmos. I. Título.

CDD 004

LUCAS DE ARAÚJO TORRES

ALGORITMOS GENÉTICOS PARA O PROBLEMA DA DOMINAÇÃO ROMANA 2-FORTE EM GRAFOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: 20 de janeiro de 2026

BANCA EXAMINADORA

Prof. Dr. Atílio Gomes Luiz (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Fábio Carlos Sousa Dias
Universidade Federal do Ceará (UFC)

Prof. Dr. Wladimir Araújo Tavares
Universidade Federal do Ceará (UFC)

Aos meus familiares, amigos e professores. E, com eterno carinho, ao meu amigo Maycon Mendes, que hoje me assiste do céu e a quem dedico esta e todas as minhas vitórias.

AGRADECIMENTOS

Agradeço primeiramente aos meus familiares. À minha mãe, ao meu pai e à minha avó, que sempre cuidaram de mim e me ofereceram suporte incondicional. Ao meu tio materno, pela educação exemplar que me proporcionou desde a infância. Agradeço também aos meus tios, tias e primos, que sempre estiveram ao meu lado. De forma especial, dedico este trabalho aos meus irmãos mais novos, que são minhas maiores motivações para chegar até aqui.

Dedico também aos meus amigos, que sempre me deram suporte, oferecendo bons conselhos, direcionamentos e me acompanhando nesta caminhada.

Um agradecimento especial aos mentores e parceiros que ajudaram a trilhar minha jornada acadêmica e profissional. Ao Prof. Atílio Gomes Luiz, por ter moldado minha trajetória acadêmica com paciência e sabedoria, sempre me conduzindo por bons caminhos e servindo como inspiração de disciplina. Dedico também ao meu antigo professor Vladymir, por ter sido meu mentor pioneiro e se manter presente até hoje, e ao meu amigo João Feitoza, pela parceria de trabalho e por ter contribuído significativamente para o meu crescimento.

Por fim, e mais importante, agradeço a Deus por traçar meus caminhos e me fortalecer com o tempo. Sem Ele, nada disso seria possível.

“A ciência é muito mais do que um corpo de conhecimento. É uma maneira de pensar.”
(Carl Sagan)

RESUMO

O Problema de Dominação Romana k-Forte, introduzido em 2020, é uma generalização do problema clássico de dominação romana, cujo objetivo é determinar uma estratégia eficiente para defender um grafo sob múltiplos ataques simultâneos, por meio da alocação de tropas em seus vértices. Nesse contexto, cada vértice do grafo recebe um rótulo inteiro não negativo que representa a quantidade de tropas posicionadas. Vértices rotulados com 1 são considerados protegidos, enquanto vértices com rótulo maior ou igual a 2 são considerados suficientemente guarnecidos para se proteger e, adicionalmente, defender uma quantidade limitada de vizinhos vulneráveis — aqueles rotulados com 0. O problema consiste em atribuir os rótulos de modo que todos os vértices estejam protegidos ou passíveis de proteção, minimizando a soma total dos rótulos, de modo que se houver k ataques simultâneos, todos os vértices atacados possam ser protegidos. Este trabalho concentra-se no caso em que $k = 2$, no qual se assume que o grafo pode ser atacado simultaneamente em dois vértices distintos. Acredita-se que o problema de determinar a soma mínima dos rótulos em uma dominação romana k-forte seja NP-completo. Diante dessa complexidade, diferentes abordagens heurísticas têm sido propostas na literatura científica para a obtenção de soluções aproximadas eficientes, como formulações por Programação Inteira (PI), técnicas de decomposição de Benders, algoritmos do tipo L-Shaped e métodos baseados em busca em vizinhança variável (VNS). Neste trabalho, investiga-se o uso de heurísticas baseadas em algoritmos genéticos para abordar o Problema de Dominação Romana 2-Forte. Duas abordagens são propostas: algoritmos genéticos tradicionais e o BRKGA (*Biased Random-Key Genetic Algorithm*). As propostas geram soluções de boa qualidade para o problema, fornecendo aproximações eficientes para a soma mínima dos rótulos em uma dominação romana 2-forte de um grafo, e compara esses resultados com os previamente obtidos na literatura.

Palavras-chave: dominação romana 2-forte, teoria dos grafos, algoritmos genéticos, programação linear inteira, heurísticas, BRKGA.

ABSTRACT

The k-Strong Roman Domination Problem, introduced in 2020, is a generalization of the classical Roman domination problem, whose objective is to determine an efficient strategy to defend a graph under multiple simultaneous attacks by allocating troops to its vertices. In this context, each vertex of the graph receives a non-negative integer label representing the number of troops stationed there. Vertices labeled with 1 are considered protected, while vertices labeled with a value greater than or equal to 2 are considered sufficiently fortified to protect themselves and additionally defend a limited number of vulnerable neighbors—those labeled with 0. The problem consists of assigning labels such that all vertices are protected or defensible, minimizing the total sum of labels, so that if there are k simultaneous attacks, all attacked vertices can be protected. This work focuses on the case where $k = 2$, assuming the graph can be attacked simultaneously at two distinct vertices. It is believed that the problem of determining the minimum sum of labels in a k-strong Roman domination is NP-complete. Given this complexity, different heuristic approaches have been proposed in the scientific literature to obtain efficient approximate solutions, such as Integer Programming (IP) formulations, Benders decomposition techniques, L-Shaped algorithms, and methods based on Variable Neighborhood Search (VNS). In this work, the use of heuristics based on genetic algorithms to address the 2-Strong Roman Domination Problem is investigated. Two approaches are proposed: traditional genetic algorithms and the BRKGA (Biased Random-Key Genetic Algorithm). The proposals generate good quality solutions for the problem, providing efficient approximations for the minimum sum of labels in a 2-strong Roman domination of a graph, and compare these results with those previously obtained in the literature.

Keywords: 2-strong Roman domination, graph theory, genetic algorithms, integer linear programming, heuristics, BRKGA

LISTA DE SÍMBOLOS

γ_R	Número de Dominação Romana
γ_{R2F}	Número de Dominação Romana 2-Forte
$\mathscr{P}$	Conjunto Enumerado de Todos os Padrões de Ataques
δ	Grau Mínimo do Grafo
Δ	Grau Máximo do Grafo
γ	Número de Dominação

SUMÁRIO

1 INTRODUÇÃO

No final do século III e início do século IV da Era Comum, o Império Romano dominava vastas regiões, abrangendo grande parte da Europa, estendendo-se para o Leste até o Oriente Próximo — incluindo áreas da atual Turquia, Mesopotâmia — e também o norte da África. Nesse período, Roma enfrentava uma grave crise e intensa instabilidade política, marcada por guerras constantes e disputas territoriais. Um dos principais rivais era o Império Sassânida, que ameaçava os domínios romanos no Oriente Próximo e no Egito. Ao norte, tribos germânicas também representavam uma ameaça, disputando terras no norte e centro da Europa (Heather, 2006).

No período de 306 d.C. a 337 d.C., Constantino foi o imperador de Roma. Durante seu reinado, o Império Romano enfrentava constantes ataques direcionados a diversas províncias espalhadas por seu vasto território. O império era composto de 8 províncias, sendo elas: Gália, Bretanha, Ibéria, Norte da África, Constantinopla, Roma, Egito e Ásia Menor. No entanto, devido à crise que assolava o império, Constantino dispunha de um número limitado de tropas para defender todas as 8 províncias. Diante desse cenário, ele adotou medidas estratégicas, alocando suas forças de maneira a maximizar a proteção das províncias e garantir a segurança contra os ataques inimigos. Esta narrativa foi construída com base na adaptação proposta por (Stewart, 1999), que modela esse contexto como um problema de dominação em grafos.

A estratégia de defesa do Império Romano durante o reinado de Constantino contava com dois principais componentes: as *tropas locais*, responsáveis pela proteção imediata de cada província, e os *exércitos de campo*, que possuíam maior mobilidade e podiam ser deslocados rapidamente para regiões vizinhas em caso de ataque. Essa estrutura permitia uma resposta mais flexível às ameaças, mesmo diante da limitação de recursos. No entanto, devido à crise militar e econômica enfrentada pelo império na época, Constantino dispunha de apenas quatro exércitos de campo para cobrir todo o vasto território romano. Com um número tão restrito de unidades móveis, a alocação eficiente dessas tropas tornou-se essencial para garantir a segurança das províncias, exigindo decisões estratégicas sobre onde posicioná-las para maximizar a cobertura defensiva.

A fim de assegurar a proteção das províncias, o imperador Constantino estabeleceu duas regras para a alocação de tropas. Ele estabeleceu que uma província seria considerada *segura* se satisfizesse uma das seguintes condições:

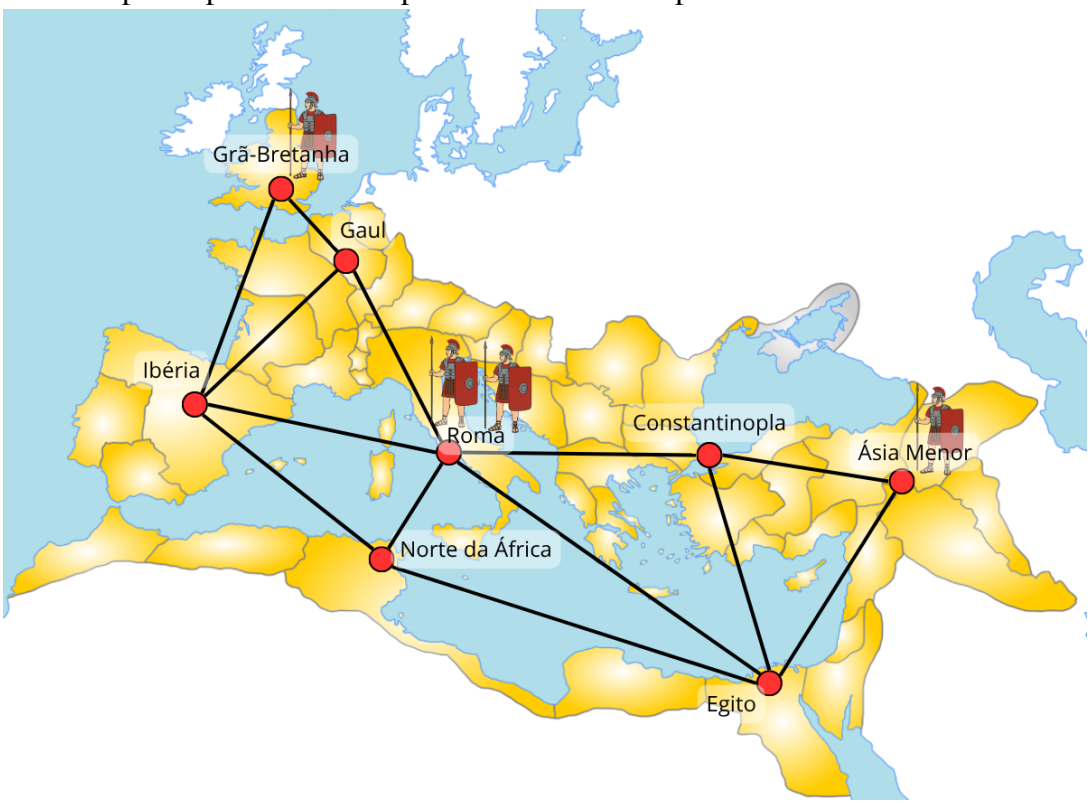
(1) Possuísse pelo menos uma tropa estacionada diretamente nela.

(2) Caso não possuísse tropa alocada para ela, que fosse vizinha de uma província que possuísse pelo menos duas tropas.

A segunda condição representa uma estratégia defensiva, onde províncias fortemente guarnecidas (com tropas excedentes) podem oferecer apoio a regiões vizinhas sem tropas alguma, sem comprometer sua própria segurança. Assim, caso uma província sem tropas viesse a ser atacada, uma vizinha sua que possuísse pelo menos duas tropas poderia enviar uma de suas tropas para protegê-la, permanecendo ainda com pelo menos uma tropa em seus domínios, por precaução.

Na Figura 1, é apresentada uma versão simplificada do mapa do Império Romano, com a divisão de suas principais províncias, utilizada como base para a modelagem do problema de dominação abordado neste trabalho. Observe que as tropas estão distribuídas no mapa de forma a garantir a proteção de todas as cidades. Estrategicamente, duas tropas estão posicionadas na província de Roma, uma na Bretanha e uma na Ásia Menor.

Figura 1 – Mapa simplificado do Império Romano e suas províncias.



Fonte: Adaptado a partir de (Genially, 2023)

Por volta de 2004, Cockayne et al. (Cockayne *et al.*, 2004) modelaram esse problema

de alocação de tropas como um problema em teoria dos grafos, apresentado a seguir. Seja um grafo $G = (V(G), E(G))$, onde $V(G)$ é o conjunto de vértices, que representa cada província do império, e $E(G)$ é o conjunto de arestas, que representa o conjunto de vias (terrestres ou marítimas) que interligam as províncias do império. O processo de alocação de tropas é modelado como uma função $f\colon V(G) \to \{0,1,2\}$, denominada *função de dominação romana (FDR)*, que atribui *rótulos* 0, 1 ou 2 aos vértices do grafo e que deve obedecer a uma única restrição: todo vértice com rótulo 0 deve ter pelo menos um vértice vizinho com rótulo 2. O *peso* de uma FDR f é a soma dos rótulos de todos os vértices do grafo sob a função f. Denominamos *número de dominação romana* o menor peso que uma FDR de um grafo pode ter e denotamos esse número por $\gamma_R(G)$.

Outras variantes do problema de dominação romana foram introduzidas ao longo dos anos, como a dominação romana dupla (Beeler *et al.*, 2016), tripla (Abdollahzadeh Ahangar *et al.*, 2021), perfeita (Henning; Klostermeyer, 2018), entre outras. A variante de interesse neste trabalho é denominada *Dominação Romana 2-Forte*, que é uma versão restrita do problema de Dominação Romana k-Forte, originalmente proposto por (Liu *et al.*, 2020). A principal diferença entre a Dominação Romana e a Dominação Romana 2-Forte é que a primeira considera que no máximo uma província pode vir a ser atacada, enquanto que a segunda considera que até duas províncias podem ser atacadas simultaneamente, aumentando assim a complexidade da proteção. A seguir, definimos formalmente este problema conforme ele está definido no artigo de Nikolić *et al.* (2024).

Dado um grafo $G = (V(G), E(G))$ com n vértices, definimos um *padrão de ataque* como um subconjunto de vértices $P = \{v_1, v_2\}$ tal que $v_1, v_2 \in V(G)$, ou seja, um subconjunto contendo dois vértices que podem ser atacados simultaneamente. Note que, pela definição, existem $\binom{n}{2}$ possíveis padrões de ataque diferentes em um grafo. Denotamos por $\mathscr{P}(G) = \{P_1, \ldots, P_{\binom{n}{2}}\}$ o conjunto enumerado de todos os padrões de ataque no grafo G.

Uma função $f : V(G) \to \{0,1,2,3\}$ é chamada de *função de dominação romana 2-forte* (FDR2F) se, para cada padrão de ataque $P \in \mathscr{P}(G)$, as seguintes condições forem satisfeitas:

(1) Todo vértice rotulado com $f(v) \geq 2$ pode defender, no máximo, $f(v) - 1$ vértices vizinhos pertencentes a P que estejam rotulados com 0;

(2) Cada vértice do padrão de ataque P que possui rótulo 0 pode ser defendido por pelo menos um de seus vizinhos.

Observa-se que vértices com rótulo diferente de zero já estão automaticamente protegidos (possuem pelo menos uma tropa alocada neles). Assim, o foco principal está em proteger os vértices rotulados com zero que fazem parte de cada padrão de ataque. O objetivo do problema é encontrar uma função f que satisfaça as condições acima para todos os padrões de ataque de $\mathscr{P}(G)$, minimizando o somatório dos rótulos atribuídos aos vértices. O *peso* de uma FDR2F f de G, denotado por $\omega(f,G)$, é a soma dos rótulos dos vértices de G sob f, ou seja, $\omega(f,G) = \sum_{v \in V(G)} f(v)$. O menor peso que uma FDR2F de G pode ter é chamado de *número de dominação romana 2-forte* de G e é denotado por $\gamma_{R2F}(G)$.

A Figura 2 ilustra um grafo G com vértices $V(G) = \{a,b,c,d,e\}$ submetido a um padrão de ataque $P = \{b,d\}$. A Figura 3 apresenta uma função de dominação romana 2-forte aplicada ao mesmo grafo, cuja rotulação é capaz de proteger contra o padrão de ataque anterior. Nota-se que essa função também é válida para qualquer padrão de ataque envolvendo dois vértices, satisfazendo as condições de dominância exigidas em todos os casos.

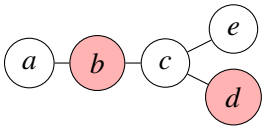


Figura 2 – Grafo com um padrão de ataque nos vértices em vermelho
Fonte: Elaborado pelo autor.

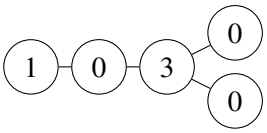


Figura 3 – Rotulação FDR2F ótima para todos padrões de ataques no grafo

O conceito de Dominação Romana 2-Forte foi inicialmente proposto por Liu *et al.* (2020). Na verdade, esses autores apresentaram o problema mais geral de Dominação Romana k-Forte, válido para qualquer $k \geq 2$. Eles abordaram o problema sob uma perspectiva prática, desenvolvendo dois algoritmos exatos: um baseado em Programação Inteira (PI) e outro utilizando a técnica de decomposição de Benders.

Posteriormente, Nikolić et al. (Nikolić *et al.*, 2024) realizaram o primeiro estudo teórico sobre esse problema, sob a ótica da teoria dos grafos. Nesse trabalho, os autores estabeleceram conexões com outros problemas clássicos de dominação, além de apresentarem limites superiores e inferiores para o parâmetro $\gamma_{R2F}(G)$ em diferentes classes de grafos, tanto gerais quanto específicas.

O estudo mais recente envolvendo a dominação romana k-forte é o de Djukanović et al. (Djukanović *et al.*, 2025), no qual os autores propõem uma heurística baseada na busca em vizinhança variável (VNS). Essa abordagem explora o conceito de quase-viabilidade e distingue

os tipos de ataque em dois grupos distintos, buscando melhorar a qualidade das soluções obtidas para o problema. A heurística proposta foi testada e comparada com alguns resultados já realizados no trabalho de (Liu *et al.*, 2020).

O conceito de Dominação Romana 2-Forte possui aplicações práticas. Por exemplo, em guerras, os bombardeios são geralmente realizados simultaneamente em mais de uma região, dificultando ainda mais a proteção. Outro exemplo mais lúdico é o jogo de estratégia *War*, no qual devemos proteger e conquistar territórios no tabuleiro, podendo sofrer múltiplos ataques dos jogadores. Nessas situações, torna-se fundamental adotar estratégias de defesa que maximizem a cobertura com o menor número possível de recursos.

1.1 Contribuição Deste Trabalho

Embora a literatura sobre Dominação Romana k-Forte tenha avançado com métodos algorítmicos exatos e heurísticos para k arbitrário, o caso específico $k = 2$, relevante em cenários de dupla ameaça simultânea, ainda carece de abordagens especializadas e eficientes. Este trabalho concentra-se no Problema de Dominação Romana 2-Forte (PDR2F). Investigamos meta-heurísticas evolutivas, como algoritmos genéticos, que permanecem pouco exploradas para esta variante, mas com potencial para obter soluções de qualidade em tempo viável, especialmente em instâncias maiores.

Neste contexto, este trabalho contribui com as seguintes iniciativas:

1. Proposta de dois algoritmos meta-heurísticos baseadas em algoritmos genéticos:
   - Um algoritmo genético clássico (GA) com codificação direta baseada em valores, inspirado em abordagens consolidadas para problemas de dominação em grafos, mas adaptado às especificidades do PDR2F.
   - Um algoritmo genético de chaves aleatórias enviesadas (BRKGA), variante que tem se mostrado eficaz em problemas de otimização combinatória devido ao seu mecanismo de busca indireta e elitismo enviesado.

   Ambos foram desenvolvidos em Python, utilizando bibliotecas especializadas, e seus operadores (seleção, cruzamento, mutação e decodificação) foram projetados visando a viabilidade e a qualidade das soluções geradas.
2. Proposta de um novo modelo de Programação Linear Inteira: Com base em uma reformulação da definição do problema, proposta neste trabalho, nós propomos e implementamos um novo modelo de programação linear inteira para o PDR2F, incorporando ajustes que

melhoram sua eficiência computacional e permitem a obtenção de soluções ótimas em um conjunto mais amplo de instâncias.

Em resumo, este trabalho preenche uma lacuna na literatura ao explorar e comparar o desempenho de algoritmos genéticos na resolução do PDR2F.

1.2 Estrutura do Trabalho

Este trabalho está organizado da seguinte forma: o Capítulo 1 apresenta a Introdução, com o Problema da Dominação Romana 2-Forte, suas motivações e aplicações. O Capítulo 2 traz a Fundamentação Teórica, abordando conceitos de teoria dos grafos, dominação romana, algoritmos genéticos e programação linear inteira. O Capítulo 3 revisa os Trabalhos Relacionados ao tema. O Capítulo 4 detalha a proposta e implementação de um Algoritmo Genético clássico. O Capítulo 5 descreve o BRKGA (*Biased Random-Key Genetic Algorithm*) adaptado ao problema. O Capítulo 6 apresenta a implementação do modelo de Programação Linear Inteira. O Capítulo 7 expõe os Experimentos realizados, com análise comparativa dos métodos. Por fim, o Capítulo 8 traz as Conclusões e propostas para trabalhos futuros.

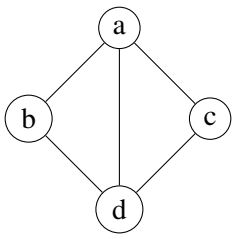
2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, apresentam-se os conceitos teóricos essenciais para a compreensão do problema abordado nesta pesquisa. Inicialmente, são introduzidas as noções fundamentais sobre dominação em grafos, com ênfase na dominação romana e em suas principais variantes. Em seguida, discute-se em detalhe o problema de dominação romana 2-forte, foco central deste trabalho, explorando suas definições, propriedades e motivações práticas, tanto em cenários aplicados quanto em contextos teóricos. Além dos aspectos combinatórios, o capítulo também fundamenta as abordagens computacionais utilizadas na modelagem e resolução do problema, com destaque para os algoritmos genéticos, o método BRKGA (Algoritmo Genético com Chaves Aleatórias Enviesadas) e a programação inteira. Cada uma dessas técnicas é apresentada com a respectiva fundamentação teórica e explicações sobre seu funcionamento, evidenciando suas vantagens e limitações no contexto da otimização combinatória.

2.1 Conceitos básicos de teoria dos grafos

Um grafo $G = (V(G), E(G))$ é uma estrutura matemática composta por um conjunto finito e não vazio $V(G)$ de elementos chamados *vértices* e um conjunto $E(G)$ de elementos denominados *arestas*, onde cada aresta é um par não ordenado de vértices distintos $u, v \in V(G)$. A definição de grafo aqui apresentada é comumente chamada de *grafo simples*; porém, chamamos simplesmente de grafo durante todo o texto. Quando $uv \in E(G)$, dizemos que u e v são *extremidades* da aresta uv e que u e v são *adjacentes*; neste caso, dizemos também que a aresta uv *incide* nos vértices u e v e vice-versa. Grafos podem ser desenhados no plano de modo que cada vértice seja representado como um ponto e as arestas sejam representadas como linhas ligando os dois pontos que são suas extremidades. A Figura 4 ilustra um exemplo de um grafo $G = (V(G), E(G))$, na qual $V(G) = \{a, b, c, d\}$ e $E(G) = \{ab, ac, ad, bd, cd\}$.

Figura 4 – Representação de um grafo G com 4 vértices



Fonte: modificado pelo autor

A *ordem* de um grafo G, denotada por n, corresponde à quantidade de vértices de G, ou seja, $n = |V(G)|$. Já o *tamanho* do grafo, denotado por m, representa a quantidade de arestas, isto é, $m = |E(G)|$. Por exemplo, na Figura 4, temos que $n = 4$ e $m = 5$.

O *grau* de um vértice v é a quantidade de arestas que incidem em v, sendo denotado por $d_G(v)$. Na Figura 4, por exemplo, temos que $d_G(a) = 3$. O *grau mínimo* de G é denotado por $\delta(G)$ e definido como $\delta(G) = \min\{d_G(v) \mid v \in V(G)\}$, enquanto o *grau máximo* é denotado por $\Delta(G)$ e definido como $\Delta(G) = \max\{d_G(v) \mid v \in V(G)\}$. Dizemos que G é um grafo *k-regular* se todos os seus vértices possuem grau igual a k, ou seja, $d_G(v) = k$ para todo $v \in V(G)$.

Um (u,v)-*passeio* $W = (u = v_0, v_1, \ldots, v_k = v)$ em um grafo G é uma sequência de vértices de G na qual quaisquer dois vértices consecutivos nessa sequência são adjacentes em G. Essa sequência inicia no vértice u e termina no vértice v. O passeio é dito *fechado* se $u = v$, e *aberto* caso contrário. Um *caminho* P no grafo é um passeio em que não há repetição de vértices. Um grafo G é *conexo* se, para todo par de vértices de G, existe ao menos um caminho em G que os conecta. Caso exista ao menos um par de vértices em G entre os quais não há caminho, dizemos que o grafo G é *desconexo*.

Um subconjunto $D \subseteq V$ é dito *dominante* se todo vértice $v \in V(G) \setminus D$ é adjacente a pelo menos um vértice de D (Haynes *et al.*, 1998). A cardinalidade do menor conjunto dominante de um grafo G é chamada de *número de dominação* de G, denotado por $\gamma(G)$.

A vizinhança aberta de um vértice $v \in V(G)$ é o conjunto de todos os vértices adjacentes a v, sendo denotada por $N_G(v) = \{u \in V(G) \mid uv \in E(G)\}$. Já a vizinhança fechada de v inclui o próprio vértice v juntamente com sua vizinhança aberta, sendo representada por $N_G[v] = N_G(v) \cup \{v\}$. Esses conceitos são amplamente utilizados em problemas de dominação, onde frequentemente analisamos se um vértice é dominado por um conjunto dominante a partir da sua vizinhança.

Algumas classes básicas de grafos são comumente estudadas em problemas de dominação. A seguir, definimos algumas delas. Um *caminho* com n vértices, denotado por P_n, é um grafo tal que $V(P_n) = v_1, v_2, \ldots, v_n$, e o conjunto de arestas é dado por $E(P_n) = v_i v_{i+1} \mid 1 \leq i < n$. Um *ciclo*, denotado por C_n, é um grafo tal que $V(C_n) = \{v_1, v_2, \ldots, v_n\}$, e suas arestas são dadas por $E(C_n) = \{v_i v_{i+1} \mid 1 \leq i < n\} \cup \{v_n v_1\}$. Um grafo G é denominado *k-regular* se todos os seus vértices possuem o mesmo grau k. Formalmente, isso significa que para todo vértice $v \in V(G)$, o grau de v é igual a k (isto é, $d(v) = k$). Uma classe importante derivada desta definição são os grafos *cúbicos*, que correspondem exatamente aos grafos 3-regulares.

Vale notar também que o ciclo C_n, definido anteriormente, é um exemplo de grafo 2-regular.

As definições apresentadas nesta seção, bem como outros conceitos da teoria dos grafos, podem ser encontrados com maiores detalhes na obra de Diestel (2017).

2.2 Dominação Romana 2-Forte

Como já discutido no Capítulo 1, Liu et al. (Liu *et al.*, 2020) introduziram o Problema da Dominação Romana k-Forte por volta de 2020, propondo algoritmos exatos para o problema. Em seguida, Nikolić *et al.* (2024) investigaram o problema de forma teórica. A seguir, apresentamos a definição do problema de acordo com o trabalho de Nikolić *et al.* (2024), considerando o contexto restrito $k = 2$, que é o foco deste trabalho de conclusão de curso.

Seja $G = (V(G), E(G))$ um grafo com n vértices. Seja $f : V(G) \to \{0,1,2,3\}$ uma função que associa a cada vértice de G um valor inteiro (*rótulo* ou *peso*) entre 0 e 3. Para cada $j \in \{0,1,2,3\}$, definimos o conjunto de vértices $V_j = \{v \in V(G) \mid f(v) = j\}$, e também $V_{\geq j} = \{v \in V(G) \mid f(v) \geq j\}$. Observe que existe uma correspondência biunívoca entre cada função $f : V(G) \to \{0,1,2,3\}$ e a partição ordenada (V_0, V_1, V_2, V_3). Assim, alternativamente, podemos representar a função f pela quádrupla (V_0, V_1, V_2, V_3). O *peso* de uma função f de um grafo G, como acima definida, é definido como $\omega(f,G) = \sum_{v \in V(G)} f(v)$, ou seja, é a soma dos pesos de todos os vértices do grafo G sob a função f.

Dado um grafo G e uma função $f : V(G) \to \{0,1,2,3\}$ que atribui peso aos vértices de G, dizemos que um vértice $v \in V(G)$ *pode defender* um vizinho $w \in N_G(v)$ se pudermos subtrair uma unidade do peso de v e adicionar uma unidade ao peso de w e ainda resultar, ao final, com um peso positivo no vértice v. Um vértice $v \in V(G)$ é dito *protegido* ou que *está defendido* se $f(v) \geq 1$. Por estas definições, temos que um vértice $v \in V(G)$ com rótulo $f(v) = 1$ está protegido mas não pode defender nenhum de seus vizinhos.

Para um grafo G, definimos um *padrão de ataque* $P = \{v_1, v_2\}$ tal que $v_1, v_2 \in V(G)$, ou seja, um subconjunto contendo dois vértices que podem ser atacados simultaneamente. Veja que, pela definição, existem $\binom{n}{2}$ possíveis padrões de ataque em um grafo. Denotamos por $\mathscr{P}(G) = \{P_1, \ldots, P_{\binom{n}{2}}\}$ o conjunto enumerado de todos os padrões de ataque no grafo G.

Denomina-se *função de dominação romana 2-forte* ($FDR2F$) qualquer função $f : V(G) \to \{0,1,2,3\}$ tal que, para todo padrão de ataque $P \in \mathscr{P}$, sejam asseguradas as seguintes propriedades:

(1) Todo vértice rotulado com $f(v) \geq 2$ pode defender, no máximo, $f(v) - 1$ vértices vizinhos

pertencentes a P que estejam rotulados com 0;

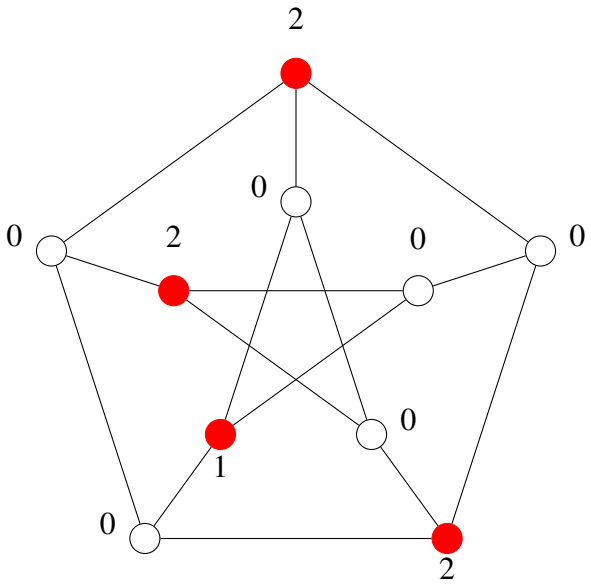
(2) Cada vértice do padrão de ataque P que possui rótulo 0 pode ser defendido por pelo menos um de seus vizinhos.

Pelas definições, vértices com peso diferente de zero já estão automaticamente protegidos, conforme as propriedades estabelecidas anteriormente. Uma $FDR2F$ trivial em um grafo é aquela em que todos os vértices recebem peso 1.

O *Problema da Dominação Romana* 2*-Forte* (PDR2F) em um grafo G consiste em determinar o menor peso $\omega(f,G)$ possível de uma FDR2F f de G. O menor valor possível do peso de tal função é denominado *número de dominação romana* 2*-forte*, e é representado por $\gamma_{R2F}(G)$. Formalmente, temos $\gamma_{\mathrm{R2F}}(G) = \min\{\omega(f,G) \mid f \text{ é uma função FDR2F válida em } G\}$. Uma FDR2F com o menor peso possível é dita *ótima*.

A Figura 5 apresenta uma função de dominação romana 2-forte ótima para o grafo de Petersen. Nessa solução, três vértices são rotulados com peso 2, um vértice com peso 1, e os demais estão rotulados com peso 0. Essa configuração garante que, para quaisquer dois ataques simultâneos, os vértices em vermelho conseguem proteger e dominar todo o grafo.

Figura 5 – Grafo de Petersen com uma função de dominação romana 2-forte ótima.



Fonte: elaborado pelo autor

Um padrão de ataque contendo apenas vértices rotulados com um valor positivo é considerado protegido, pois cada um desses vértices é autoprotegido. Portanto, estamos interessados apenas em padrões de ataque que contenham ao menos um vértice rotulado com zero. Logo, o desafio reside em garantir que todos os padrões de ataque contendo vértices

rotulados com 0 sejam adequadamente defendidos. Note que todo subconjunto $B \subset V_0$ com $1 \leq |B| \leq 2$ pertence a pelo menos um padrão de ataque. Para cada subconjunto B, deve existir um conjunto $A \subset V_{\geq 2}$ tal que cada vértice de B seja defendido por pelo menos um vizinho em A, respeitando os limites impostos pela função de dominação romana 2-forte f. Isso implica que $\sum_{v \in A} f(v) \geq |A| + |B|$. Essa relação de cobertura impõe restrições adicionais sobre a distribuição dos pesos no grafo, dificultando a construção de soluções triviais e tornando o problema computacionalmente desafiador. Como consequência, a determinação de uma $FDR2F$ de peso mínimo exige considerar cuidadosamente os padrões de ataque possíveis e o potencial defensivo de cada vértice com $f(v) \geq 2$.

2.2.1 Uma definição equivalente para a dominação romana 2-forte

A definição clássica apresentada anteriormente baseia-se na verificação de todos os $\binom{n}{2}$ padrões de ataque possíveis, o que pode se tornar computacionalmente custoso ou inviável para grafos densos ou de grande porte. No entanto, é possível reformular as condições de segurança, observando apenas as propriedades locais da vizinhança de cada vértice. Portanto, logo abaixo, nós propomos uma definição equivalente para o problema.

Nesta nova abordagem, definimos o conjunto de *protetores* de um vértice v, denotado por $PN(v)$, como o conjunto $PN(v) = \{w \in N(v) : f(w) \geq 2\}$. Dessa forma, uma função $f : V(G) \to \{0,1,2,3\}$ é considerada uma *função de dominação romana 2-forte* (FDR2F) se, e somente se, satisfizer as duas condições locais abaixo:

1. Se $f(v) = 0$, então v deve possuir pelo menos um vizinho com rótulo maior ou igual a 2 (isto é, $PN(v) \neq \emptyset$);
2. Se $f(v) = 2$, então, dentre todos os vizinhos de v rotulados com 0, no máximo um deles pode ter v como seu único protetor. Ou seja, existe no máximo um vizinho w tal que $PN(w) = \{v\}$.

A equivalência entre as definições é explicada a seguir. Note que um vértice com peso 2 pode defender no máximo 1 vizinho em um ataque duplos simultâneo, já que uma das duas tropas deve permanecer estacionada nele mesmo. Se um vértice v com peso 2 fosse o único protetor de dois vizinhos distintos, digamos u_1 e u_2 (ambos com peso 0), um padrão de ataque direcionado a $\{u_1, u_2\}$ seria indefensável, pois v só poderia salvar um deles. Portanto, a condição (b) desta nova definição garante exatamente que isso não ocorra: um vértice de peso 2 pode ser o protetor exclusivo de apenas um vizinho; quaisquer outros vizinhos conectados a ele

devem possuir outros vizinhos fortes para garantir a defesa em ataques simultâneos.

Essa formulação alternativa simplifica a verificação da viabilidade de uma solução, transformando um problema de análise combinatória de pares em uma verificação linear de restrições de vizinhança, o que é fundamental para a modelagem eficiente do problema.

2.2.2 Resultados teóricos

Nesta subseção, são apresentados alguns resultados que estabelecem limitantes superiores e inferiores para o número de dominação romana 2-forte, conforme apresentado no artigo de Nikolić *et al.* (2024).

O teorema abaixo apresenta um limitante inferior para o parâmetro $\gamma_{R2F}(G)$ restrito a grafos 3-regulares, que é uma das classes que consideraremos neste trabalho.

Teorema 1 (Nikolić *et al.* (2024))**.** *Se $G = (V(G), E(G))$ é um grafo 3-regular com n vértices, então $\gamma_{R2F}(G) \geq \frac{2n}{3}$.*

A seguir, é apresentado um teorema que estabelece um limitante superior para $\gamma_{R2F}(G)$ com base no número de dominação romana $\gamma(G)$ de um grafo G.

Teorema 2 (Nikolić *et al.* (2024))**.** *Se G é um grafo, então $\gamma_{R2F}(G) \leq 3\gamma(G)$.*

Demonstração. O resultado decorre do fato de que cada vértice rotulado com 3 pode proteger até 2 vértices adjacentes. Portanto, se $S \subseteq V(G)$ é um conjunto dominante, é suficiente rotular cada vértice de S com 3, e os demais vértices com 0. □

O Lema 3 estabelece um limitante superior e inferior para os valores da função de dominação k-romana forte, considerando dois inteiros distintos k_1 e k_2 tais que $k_1 < k_2$.

Lema 3 (Nikolić *et al.* (2024))**.** *Se k_1 e k_2 são inteiros tais que $k_1 < k_2$, então*

$\gamma_{Rk_1F}(G) \leq \gamma_{Rk_2F}(G) \leq |V(G)|$.

Demonstração. O resultado decorre diretamente do fato de que qualquer estratégia de defesa viável contra um padrão de ataque P, com $|P| = k_2$, garante a proteção contra qualquer subpadrão de ataque $P' \subset P$, com $|P'| = k_1$. □

A seguir, apresentamos um lema que mostra como o aumento na quantidade de arestas de um grafo pode influenciar a minimização da função de dominação romana 2-forte. Em

particular, ele evidencia que adicionar arestas tende a facilitar a defesa do grafo, potencialmente reduzindo o valor de γ_{R2F}.

Lema 4 (Nikolić *et al.* (2024))**.** *Para grafos $G = (V(G), E(G))$ e $H = (V(G), E(H))$, tais que $E(G) \subset E(H)$, garante que $\gamma_{R2F}(H) \leq \gamma_{R2F}(G)$.*

Demonstração. Seja f uma função de dominação romana 2-forte ótima para o grafo G. Essa função fornece uma rotulação dos vértices que determina uma estratégia de defesa viável contra $k = 2$ ataques no grafo G. Como o grafo H possui o mesmo conjunto de vértices que G e preserva todas as conexões existentes entre os vértices de G, a estratégia de defesa especificada continua válida para o grafo H. Em outras palavras, a função f também é uma função de dominação $R2F$ própria para o grafo H. Como consequência, temos que $\gamma_{R2F}(H) \leq \omega(f, G)$. Essa desigualdade vale para qualquer função de dominação $R2F$ f do grafo G e, em particular, temos que $\gamma_{R2F}(H) \leq \gamma_{R2F}(G)$. □

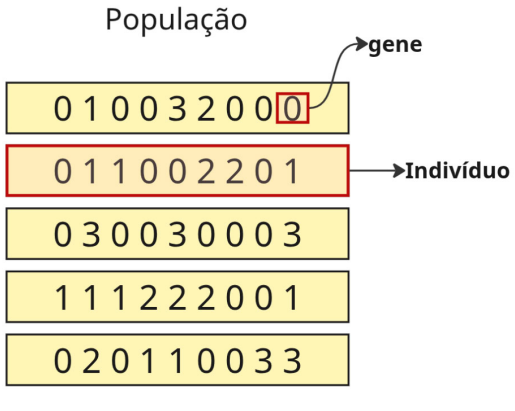
2.3 Algoritmos genéticos

Os *Algoritmos Genéticos* (AG) são técnicas de busca e otimização inspiradas nos princípios da evolução natural e da genética biológica. Propostos inicialmente por John Holland na década de 1970 e popularizados por David E. Goldberg nos anos 1980, os AGs operam por meio de um processo iterativo que simula a seleção natural, cruzamento (recombinação) e mutação de uma população de possíveis soluções (Goldberg, 1989). Ele simula o processo evolutivo proposto por Darwin, no qual os indivíduos com melhores características tendem a sobreviver e se reproduzir na natureza (Darwin, 1859). Esse processo natural é guiado pela seleção natural, onde variações fenotípicas vantajosas — resultantes de mutações ou recombinações genéticas — conferem maior sucesso reprodutivo aos organismos portadores dessas características. Ao longo das gerações, essas variações benéficas se acumulam na população, enquanto características desfavoráveis tendem a desaparecer, promovendo a adaptação das espécies ao ambiente.

Em termos gerais, um algoritmo genético trabalha com uma população de indivíduos, onde cada indivíduo representa uma solução candidata para um problema de otimização. Cada solução é codificada como um vetor de genes — tradicionalmente binário, embora codificações baseadas em números inteiros, reais ou permutacionais também possam ser utilizadas, dependendo da natureza do problema (ver Figura 6). A qualidade de cada indivíduo é medida por uma função de aptidão, também chamada função de *fitness*, que avalia o desempenho da solução em

relação ao objetivo do problema. Essa função atua como um critério de seleção natural: se o objetivo da solução ótima for a maximização, os indivíduos com maior valor de *fitness* terão maior probabilidade de serem selecionados para reprodução e sobrevivência nas próximas gerações; caso contrário, serão favorecidos aqueles com menor valor. A evolução dessas soluções ocorre ao longo de diversas gerações, com o objetivo de aprimorar progressivamente a população.

Figura 6 – Estrutura de uma população em um AG.

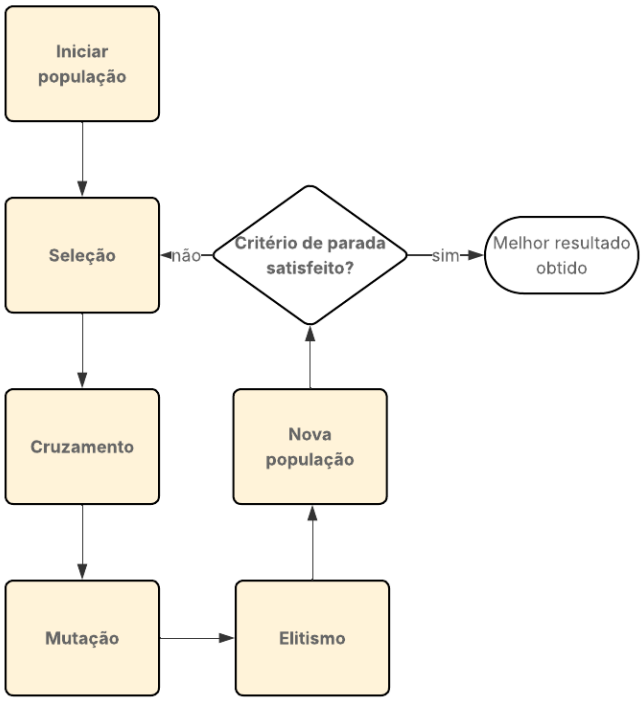


Fonte: Modificado pelo autor

Inicialmente, o algoritmo gera aleatoriamente uma população inicial de soluções, ou pode gerar essa população provinda de uma heurística gulosa. A partir dela, são aplicadas operações inspiradas na biologia evolutiva, como seleção, cruzamento (*crossover*) e mutação. A seleção favorece os indivíduos mais aptos — ou seja, aqueles com maior valor de *fitness* — que terão mais chances de transmitir seus genes para a próxima geração. Essa aptidão é determinada por meio de uma função específica do problema, a qual avalia se o indivíduo possui genes com valores que compõem uma solução viável, atribuindo a ele uma medida de desempenho. O cruzamento combina características de dois ou mais indivíduos para gerar novos descendentes. Esse processo ocorre por meio da troca de determinados genes ou sequências de genes entre os indivíduos, resultando em cromossomos recombinados e potencialmente mais adaptados. A forma como essa troca é realizada depende de uma função de cruzamento previamente definida, que varia conforme a modelagem do algoritmo genético e a natureza do problema. Em seguida, a mutação introduz pequenas variações aleatórias na população, geralmente alterando o valor de um gene em posições aleatórias de alguns indivíduos, com baixa probabilidade. Essas variações mantêm a diversidade genética e evitam a convergência prematura da população para soluções subótimas. Além disso, adota-se o mecanismo de elitismo, no qual uma porcentagem dos

melhores indivíduos (por exemplo, os 10% mais aptos) é preservada inalterada na próxima geração, garantindo que as melhores soluções não se percam ao longo do processo evolutivo. A repetição desse ciclo ao longo das gerações permite ao algoritmo buscar soluções cada vez mais adaptadas, convergindo, idealmente, para uma solução ótima ou satisfatória do problema (ver Figura 7).

Figura 7 – Fluxograma das etapas de um Algoritmo Genético.



Fonte: Modificado pelo autor

Um algoritmo genético (AG) clássico generalista inicia-se com a geração de uma população aleatória contendo N indivíduos. A partir dessa população inicial, o algoritmo inicia sua busca por uma solução ótima, ou pelo menos satisfatória, utilizando como critério de parada um número máximo de iterações M. A cada iteração, o AG executa as três operações fundamentais já mencionadas: seleção, cruzamento e mutação, aplicadas sobre um subconjunto p de indivíduos. Parâmetros como a probabilidade de cruzamento, a estratégia de seleção, a taxa de mutação e o método de sobrevivência para a próxima geração podem ser definidos previamente no algoritmo. Esse último está associado ao elitismo, definido anteriormente, que garante a preservação dos indivíduos mais aptos entre as gerações. Entretanto, por se tratar de um AG generalista, é possível descrever sua estrutura de forma abstrata, sem necessariamente detalhar

esses parâmetros. O processo de cruzamento ocorre selecionando pares de indivíduos, denotados aqui como C_1 e C_2, que trocam entre si determinados genes, originando descendentes. Seja um desses descendentes O, ele é então submetido a uma operação de mutação, com determinada probabilidade, produzindo um novo indivíduo O'. Esse procedimento se repete ao longo das gerações até que o critério de parada seja atingido. O pseudocódigo 11 descreve o funcionamento de um AG clássico, baseado no algoritmo apresentado por a revisão teórica realizada por Sourabh Katoch et al. Katoch *et al.* (2021) em sua revisão da literatura de algoritmos genéticos.

Algorithm 1 Algoritmo Genético Clássico

Entrada: Tamanho da população N, máximo de iterações M
Saída: Melhor indivíduo encontrado como solução

```
Inicializar uma população P com N indivíduos gerados aleatoriamente
i ← 0
while i < M do
    Selecione pares de individuos da população baseado no fitness
    Aplicar o operador de cruzamento sobre os pares selecionados, gerando descendentes
    Aplicar o operador de mutação sobre os descendentes com probabilidade pré-definida
    Formar a nova população P' a partir dos descendentes e/ou dos melhores indivíduos da
    geração anterior
    P ← P'
    i ← i + 1
end while
Retornar o melhor indivíduo encontrado como solução em p
```

Os AGs são particularmente úteis para problemas de otimização complexos, não lineares, com múltiplos ótimos locais e espaços de busca grandes ou mal definidos. Ao contrário de algoritmos determinísticos, eles realizam uma busca estocástica, o que permite escapar de ótimos locais e explorar amplamente o espaço de soluções.

Apesar de sua natureza aleatória, os AGs são guiados pela pressão seletiva imposta pela função de aptidão, o que tende a conduzir a população a regiões promissoras do espaço de busca. Eles têm sido aplicados com sucesso em diversas áreas, como engenharia, logística, inteligência artificial, bioinformática e problemas combinatórios complexos.

2.4 Algoritmo Genético com Chaves Aleatórias Enviesadas

O *Biased Random-Key Genetic Algorithm* (BRKGA), que traduzimos para o português como *Algoritmo Genético com Chaves Aleatórias Enviesadas*, é uma extensão dos algoritmos genéticos com chaves aleatórias (do inglês, *Random-Key Genetic Algorithm* (RKGA)),

originalmente propostos por Bean (Bean, 1994) e posteriormente aprimorados por Gonçalves e Resende (Goncalves; Resende, 2011). O BRKGA apresenta uma estratégia de cruzamento com viés direcionado à propagação de características dos indivíduos mais aptos, sendo particularmente eficiente em problemas de otimização combinatória. O BRKGA é um GA no qual a codificação dos cromossomos e os operados de seleção, cruzamento, mutação e até mesmo o elitismo já estão todos definidos. Essa variante é apresentada como segue.

No BRKGA, cada solução é representada por um *indivíduo*, que nada mais é do que um vetor de números reais gerados aleatoriamente no intervalo $[0,1]$. Cada posição do vetor corresponde a um *gene*. Um *decodificador* determinístico é responsável por mapear qualquer vetor desse tipo para uma solução viável do problema em questão, permitindo a avaliação da aptidão associada. Esse processo de mapeamento é essencial, pois o algoritmo realiza a busca no hipercubo contínuo $[0,1]^n$, enquanto a avaliação da qualidade da solução ocorre no espaço discreto do problema. A população de cada geração é composta por p indivíduos e é dividida em três subconjuntos: um grupo de p_e indivíduos *elite*, que possuem as melhores aptidões; um grupo de p_m *mutantes*, gerados aleatoriamente com os mesmos critérios da população inicial; e um grupo de $p - p_e - p_m$ *descendentes* produzidos por cruzamento. O mecanismo de *elitismo* garante que todos os indivíduos elite de uma geração sejam preservados inalterados na próxima geração, o que assegura a retenção das melhores soluções já encontradas.

O processo de cruzamento é central no BRKGA e incorpora um viés a favor dos indivíduos elite. Para gerar um novo indivíduo, seleciona-se um pai do conjunto elite e outro do conjunto não-elite. A herança de cada gene do descendente é determinada por um parâmetro bias $\rho_e > 0.5$. Para cada posição i do vetor, o gene do descendente será igual ao gene do pai elite com probabilidade ρ_e e igual ao gene do pai não-elite com probabilidade $1 - \rho_e$. Essa escolha é feita por meio da geração de um número real aleatório em $[0,1]$, que simula o lançamento de uma moeda viciada. Se o valor gerado for menor ou igual a ρ_e, o gene do descendente é herdado do pai elite; caso contrário, é herdado do outro pai.

Por exemplo, considere dois indivíduos com quatro genes cada, sendo o primeiro do conjunto elite e o segundo do conjunto não-elite, e suponha $\rho_e = 0.7$. Nesse caso, há maior probabilidade de que o descendente herde três ou mais genes do pai elite, fazendo com que ele se assemelhe mais a esse do que ao não-elite. Essa preferência intencional pelo material genético dos melhores indivíduos favorece a convergência da população para regiões promissoras do espaço de busca.

A introdução de mutantes a cada geração também é uma característica importante do BRKGA, pois permite a manutenção da diversidade populacional e reduz a chance de o algoritmo ficar preso em ótimos locais. Os mutantes são indivíduos gerados aleatoriamente, com cada gene escolhido de forma independente e uniforme em $[0, 1]$. Como todo vetor pode ser decodificado, os mutantes sempre resultam em soluções válidas. Ao fim de cada geração, a população completa é avaliada e novamente particionada em indivíduos elite e não-elite, reiniciando o ciclo. O BRKGA, portanto, realiza uma busca eficiente e indireta no espaço de soluções do problema, utilizando o decodificador como elo entre o espaço contínuo onde ocorre a evolução e o espaço discreto onde se mede a qualidade das soluções. A Figura 8 esquematiza o algoritmo BRKGA.

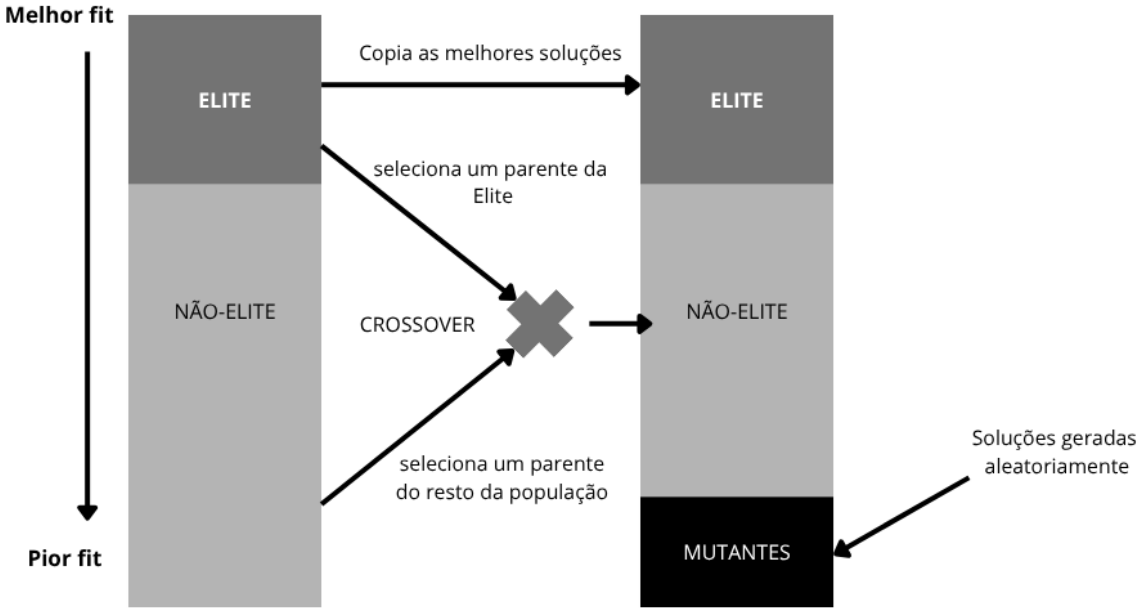


Figura 8 – Evolução da população entre gerações consecutivas em um BRKGA. Adaptado de (Silva *et al.*, 2015).

Uma característica fundamental dos algoritmos BRKGA é a separação entre a parte genérica do algoritmo e a parte que depende diretamente do problema a ser resolvido. Essa separação está centrada no processo de decodificação. A codificação, isto é, a geração de indivíduos como vetores de números reais no intervalo $[0, 1]$, é completamente independente do problema e faz parte do núcleo genérico do algoritmo. No entanto, para que esses indivíduos possam ser utilizados efetivamente, é necessário convertê-los em soluções viáveis no espaço do problema combinatório em questão. Essa conversão é realizada pelo decodificador, cuja lógica é

específica para cada tipo de problema.

Essa distinção entre o núcleo genérico do algoritmo e o componente dependente do problema pode ser visualizada no fluxograma apresentado na Figura 9. Nele, o retângulo destacado representa a etapa de decodificação. Esse bloco é o único elemento do processo que deve ser implementado de forma personalizada para cada novo problema, enquanto todas as demais etapas do BRKGA — como a geração da população, seleção, cruzamento, elitismo e mutação — permanecem invariantes. Isso torna o BRKGA uma meta-heurística de aplicação ampla e modular, onde a adaptação a diferentes contextos se concentra unicamente na definição da função de decodificação.

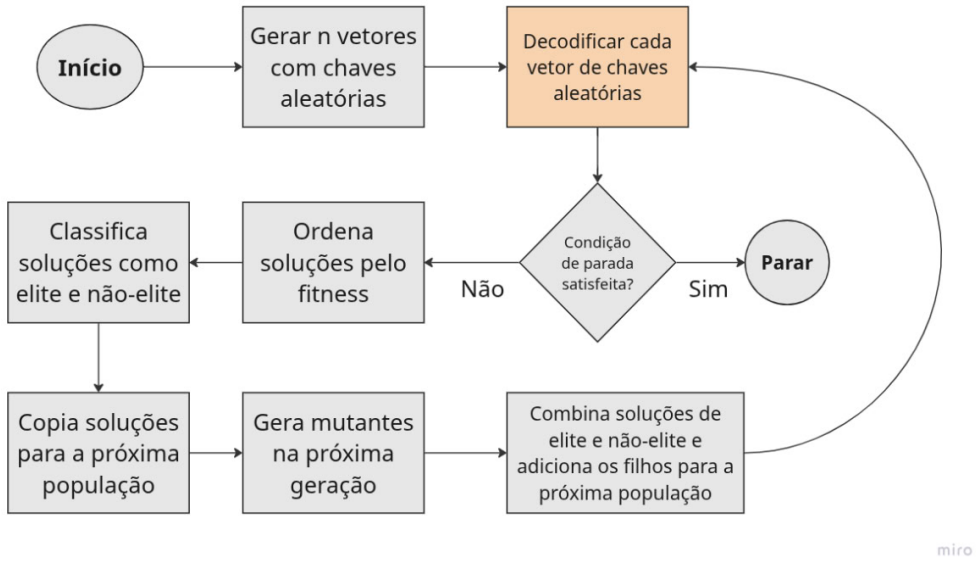


Figura 9 – Fluxograma do BRKGA adaptado de (Goncalves; Resende, 2011).

2.5 Programação Inteira

A *programação linear* (PL) é uma técnica de otimização combinatória que se propõe a resolver de forma exata a maximização ou minimização de uma função objetivo linear, sujeita a um conjunto de restrições também lineares. Ela é amplamente utilizada em problemas de alocação de recursos, planejamento de produção, logística, entre outros. A formulação geral de um problema de PL é dada por:

$$\begin{aligned} &\text{Maximizar:} \quad c^\top x \\ &\quad \text{sujeito a:} \\ &\qquad Ax \leq b \\ &\qquad x \geq 0, \end{aligned} \tag{2.1}$$

onde $x \in \mathbb{R}^n$ é o *vetor de variáveis de decisão*, $c \in \mathbb{R}^n$ representa os *coeficientes* da função objetivo, $A \in \mathbb{R}^{m \times n}$ é a *matriz de coeficientes* das restrições, e $b \in \mathbb{R}^m$ é o *vetor de constantes*.

Em muitos casos práticos, no entanto, exige-se que as variáveis assumam apenas valores inteiros — por exemplo, quando se está lidando com quantidades discretas de produtos, pessoas ou máquinas. Nestes casos, utiliza-se a *programação inteira* (PI), em que impõe-se que $x \in \mathbb{Z}^n$. A formulação passa então a ser:

$$\begin{aligned} &\text{Maximizar :} \quad c^\top x \\ &\quad \text{sujeito a :} \\ &\qquad Ax \leq b \\ &\qquad x \in \mathbb{Z}_+^n. \end{aligned} \tag{2.2}$$

Problemas de PI são, em geral, mais difíceis de resolver do que os problemas de PL, pois a introdução da restrição de integralidade das variáveis torna o problema não convexo, eliminando a aplicabilidade direta do método do *simplex* clássico. Para resolvê-los, técnicas como *branch-and-bound*, *cutting planes* e métodos híbridos têm sido desenvolvidas ao longo das últimas décadas (Wolsey, 1998). A programação linear inteira é uma ferramenta poderosa para modelar problemas reais de forma precisa, embora apresente desafios computacionais que requerem algoritmos especializados e, muitas vezes, heurísticas para grandes instâncias.

3 TRABALHOS RELACIONADOS

Nesta seção, são discutidos trabalhos anteriores que estabeleceram bases teóricas, metodológicas ou técnicas para o desenvolvimento deste projeto, destacando tanto os avanços alcançados quanto as lacunas que esta pesquisa busca abordar. A análise dessas referências permite posicionar o presente trabalho no cenário científico atual e evidenciar sua originalidade e contribuições.

3.1 Artigo: *On the k-Strong Roman Domination Problem*

Este artigo de Liu *et al.* (2020) foi o artigo seminal que introduziu o conceito de dominação romana em grafos considerando ataques simultâneos. Até então, os modelos existentes tratavam apenas cenários com um único vértice sendo atacado por vez. Neste trabalho, os autores generalizam essa abordagem ao considerar a possibilidade de até k ataques ocorrendo simultaneamente, resultando em um total de $\binom{n}{k}$ possíveis padrões de ataque.

Para encontrar boas soluções e valores ótimos para o parâmetro $\gamma_{R2F}(G)$ os autores propuseram um modelo de Programação Inteira (PI), além de abordagens estocásticas baseadas na decomposição de Benders. Também foram desenvolvidos dois algoritmos inspirados no método *L*-shaped. Os resultados evidenciam que a complexidade computacional do problema cresce rapidamente conforme o tamanho das instâncias aumenta.

Este artigo é fundamental para o desenvolvimento deste trabalho, tanto pela primeira definição formal do problema quanto pelas abordagens exatas propostas. Em especial, sua formulação em PI e sua análise estrutural foram utilizadas como referência para a construção dos experimentos e para a validação das soluções obtidas pelas heurísticas aqui implementadas.

3.2 Artigo: *Theoretical Studies of the k-Strong Roman Domination Problem*

No trabalho desenvolvido por (Nikolić *et al.*, 2024), os autores adotam uma abordagem mais voltada para a teoria dos grafos, buscando generalizar teoremas existentes e estabelecer limites superiores e inferiores para o número de dominação romana k-forte de grafos arbitrários. O estudo estabelece conexões relevantes entre esse problema e outras variantes de dominação em grafos, além de determinar o valor exato do número de dominação romana k-forte para classes específicas como caminhos, ciclos, grafos bipartidos completos, grafos grade, entre outras estruturas particulares. Pode-se considerar este artigo como uma abordagem mais teórica

do problema de dominação romana k-forte.

Este artigo contribui para este trabalho ao fornecer um embasamento teórico e uma referência de limites esperados para o problema. Sua análise estrutural e os resultados exatos para classes específicas de grafos foram considerados durante o desenvolvimento dos algoritmos, auxiliando na compreensão do problema e na verificação da viabilidade de soluções em casos particulares durante a fase de testes.

3.3 Artigo: *Graph Protection Under Multiple Simultaneous Attacks: A Heuristic Approach*

O artigo de Djukanović *et al.* (2025) propõe uma meta-heurística baseada na busca por vizinhança variável (VNS) para resolver o problema de dominação romana k-forte. Os autores iniciam com um panorama geral do problema, fundamentado nos dois trabalhos citados anteriormente, e descrevem como a técnica de VNS foi utilizada para encontrar soluções viáveis, avaliadas por meio do conceito de viabilidade relaxada. Além disso, o artigo apresenta experimentos computacionais, construindo benchmarks e comparando os resultados obtidos com os do estudo de (Liu *et al.*, 2020).

O trabalho de (Djukanović *et al.*, 2025) constitui uma referência central para esta pesquisa, estabelecendo o estado da arte na resolução heurística do problema. Suas contribuições foram essenciais para a validação experimental deste estudo, fornecendo tanto os conjuntos de instâncias de teste (*benchmarks*) quanto os resultados computacionais do algoritmo VNS e métodos exatos. Esses dados serviram como linha de base para avaliar a competitividade, a qualidade das soluções e a eficiência das abordagens evolutivas (GA e BRKGA) desenvolvidas neste trabalho.

3.4 Comparativo entre os Trabalhos Relacionados

O quadro 1 apresenta um comparativo entre os trabalhos que abordaram diretamente o problema de dominação romana k-forte por meio de algoritmos, o que permite uma comparação mais direta com este trabalho. Cada abordagem contribui de forma complementar, e a proposta aqui se posiciona como uma alternativa focada no caso $k = 2$, com o objetivo de encontrar soluções de boa qualidade em tempo viável.

Quadro 1 – Comparativo entre os trabalhos relacionados

Trabalhos	**(Liu *et al.*, 2020)**	**(Djukanović *et al.*, 2025)**	**Este trabalho**
Tipo de problema	Dominação Romana k-forte	Dominação Romana k-forte	Dominação Romana 2-forte ($k = 2$)
Abordagem principal	Programação Inteira (PI) e decomposição de Benders	Busca por Vizinhança Variável (VNS) com viabilidade relaxada	Algoritmo genético com elitismo, operador adaptado e BRKGA
Base de grafos utilizada	Grafos aleatórios com 10 e 30 vértices	Random, Wireless e Real (168 instâncias GeoJSON)	Caminhos, ciclos, estrelas, grades, grafos 3-regulares, DIMACS, Matrix Market, Random, Wireless e Real
Foco principal	Formulação e análise do problema *DRkF*	Heurística VNS aplicada ao *DRkF*	Avaliação de heurísticas baseadas em algoritmo genético e BRKGA para o caso $k = 2$

Fonte: Elaborado pelo autor.

4 ALGORITMO GENÉTICO

Este capítulo descreve o Algoritmo Genético desenvolvido especificamente para o Problema de Dominação Romana 2-Forte (PDR2F). Este algoritmo incorpora estratégias híbridas de inicialização, múltiplos operadores de cruzamento, e um mecanismo de reparação de soluções que garante a viabilidade de todos os indivíduos. A seguir são detalhados os parâmetros, representação cromossômica, e os componentes fundamentais do algoritmo.

4.1 Parâmetros do Algoritmo Genético

O algoritmo recebe como entrada um conjunto de sete parâmetros principais que controlam seu comportamento exploratório e de intensificação:

- `graph`: Instância do grafo que representa o problema a ser resolvido. Esta é a entrada principal sobre a qual todas as operações são realizadas.
- `population_size`: Número total de indivíduos (soluções candidatas) mantidos em cada geração. O tamanho da população permanece constante ao longo da execução, seguindo o modelo canônico de algoritmos genéticos.
- `gene_mutation_rate`: Probabilidade de que um gene específico (rótulo de um vértice) sofra mutação durante a fase de mutação da população.
- `n_mutants`: Fração da população que será submetida ao operador de mutação. Diferente de abordagens tradicionais onde cada indivíduo tem uma chance de mutação, este parâmetro controla quantos indivíduos serão efetivamente modificados.
- `crossover_rate`: Probabilidade de aplicação do operador de cruzamento entre pares de pais selecionados. Quando o cruzamento não é aplicado, os filhos são cópias exatas dos pais, preservando características elitistas.
- `elitism_rate`: Proporção dos melhores indivíduos da geração anterior que são preservados intocados para a próxima geração. Este mecanismo assegura que soluções de alta qualidade não sejam perdidas durante o processo evolutivo.

A saída do algoritmo é a população final após o número especificado de gerações, com o melhor indivíduo (aquele com menor soma de rótulos) representando a solução aproximada para o PDR2F.

4.2 Codificação da Solução

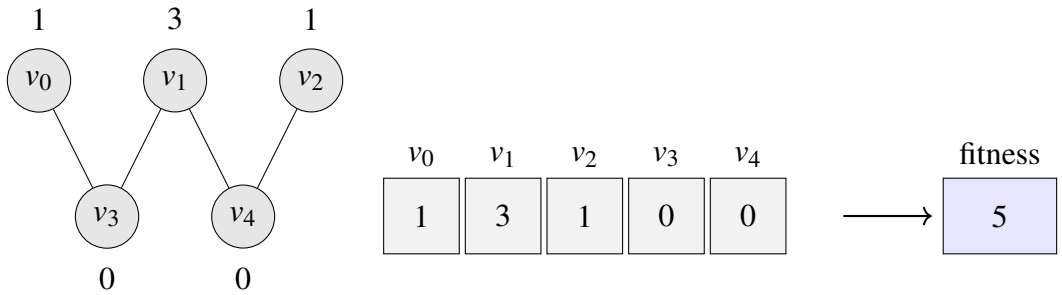
Para um grafo G com conjunto de vértices $V(G) = \{v_0, v_1, \ldots, v_{n-1}\}$, uma solução (cromossomo) para o Problema de Dominação Romana 2-Forte (PDR2F) é representada internamente como um vetor de inteiros de tamanho n, onde cada posição i corresponde ao vértice v_i e armazena um valor no conjunto $\{0,1,2,3\}$, indicando o número de tropas alocadas naquele vértice segundo a definição do PDR2F.

Formalmente, um cromossomo C é definido como $C = [c_0, c_1, \ldots, c_{n-1}]$, onde $c_i \in \{0,1,2,3\}$ para todo $i = 0, \ldots, n-1$.

O valor de c_i representa a imagem $f(v_i)$ da função de dominação romana 2-forte associada. A qualidade da solução é medida pela função de *fitness*, denotada por $F(C)$, calculada como a soma dos elementos do vetor: $F(C) = \sum_{i=0}^{n-1} c_i$.

Dado que o objetivo do PDR2F é minimizar o número total de tropas, soluções com menor valor de $F(C)$ são consideradas melhores. No entanto, nem todo vetor C constitui uma solução viável para o problema, pois deve satisfazer as condições de defesa contra todos os padrões de ataque com dois vértices. Para garantir a viabilidade, toda solução gerada ou modificada pelo algoritmo é submetida a uma função de reparação (`fix_instance`), que a transforma em uma função de dominação romana 2-forte válida antes da avaliação do fitness.

A Figura 10 ilustra a relação entre um grafo, uma função de dominação 2-forte válida e sua representação como cromossomo no algoritmo.



(a) Grafo G com função de dominação 2-forte f

(b) Cromossomo correspondente: vetor $[1,3,1,0,0]$, fitness = 5

Figura 10 – Representação visual de uma solução viável para o PDR2F e sua codificação como cromossomo no algoritmo genético.
Fonte: Elaborado pelo autor.

Essa representação direta e de baixa complexidade permite operações genéticas (cruzamento, mutação) eficientes, enquanto a etapa de reparação garante que apenas soluções válidas sejam avaliadas, alinhando a busca evolutiva com as restrições combinatórias do problema.

4.3 Visão Geral do Algoritmo Genético Personalizado

O algoritmo genético implementado neste trabalho adota uma abordagem híbrida que integra estratégias evolutivas clássicas com adaptações específicas para o PDR2F. A arquitetura proposta combina inicialização inteligente da população, operadores genéticos especializados e um mecanismo de reparação que garante a viabilidade de todas as soluções avaliadas. O fluxo principal do algoritmo segue essa estrutura:

1. Inicialização Híbrida: A população inicial é formada pela combinação de indivíduos gerados por uma heurística gulosa adaptada ao PDR2F e indivíduos gerados aleatoriamente. Esta abordagem busca equilibrar a qualidade inicial das soluções com a diversidade necessária para uma exploração eficiente do espaço de busca.
2. Ciclo Evolutivo: Em cada geração, o algoritmo aplica sequencialmente:
   - Cruzamento: Utiliza dois operadores distintos (cruzamento de um ponto e cruzamento uniforme), selecionados estocasticamente conforme a taxa de cruzamento configurada.
   - Mutação Direcionada: Aplica alterações específicas aos rótulos dos vértices, sempre no sentido de reduzir o peso total da solução, alinhada com o objetivo de minimização do PDR2F.
   - Avaliação Paralela: Todos os indivíduos são submetidos à função de reparação `fix_instance` e têm seu fitness calculado (soma dos rótulos) utilizando processamento paralelo para maior eficiência.
3. Seleção com Elitismo: A nova população é formada combinando os melhores indivíduos da geração atual com os da elite da geração anterior, preservando assim tanto progresso recente quanto soluções consolidadas.
4. Critério de Parada: A execução é interrompida após um número predeterminado de gerações.

Uma característica fundamental deste algoritmo é a separação clara entre a busca evolutiva e as restrições de viabilidade do problema. Enquanto os operadores genéticos atuam livremente no espaço de todas as rotulações possíveis $\{0,1,2,3\}^n$, a função `fix_instance` atua como um filtro determinístico que converte qualquer indivíduo em uma solução viável antes da avaliação. Esta abordagem permite que o algoritmo explore regiões do espaço de busca que poderiam ser inacessíveis em formulações mais restritivas.

O Algoritmo 2 apresenta o pseudocódigo do laço principal deste algoritmo genético personalizado.

Algorithm 2 Algoritmo Genético

Function *AlgoritmoGenetico(Grafo, TamanhoPop, NumGeracoes, TaxaMutacao, FracaoMutantes, TaxaCruzamento, TaxaElitismo)***:**

Input: Grafo do problema, tamanho da população, número de gerações, taxa de mutação por gene, fração de indivíduos mutantes, taxa de cruzamento, taxa de elitismo
Output: Melhor solução viável encontrada
PopGulosa ← InicializarPopulacaoGulosa(Grafo, $\lceil$TamanhoPop / 2$\rceil$)
PopAleatoria ← InicializarPopulacaoAleatoria(Grafo, $\lfloor$TamanhoPop / 2$\rfloor$)
Populacao ← PopGulosa ∪ PopAleatoria
for *geracao* ← 1 **to** *NumGeracoes* **do**
    PopulacaoAnterior ← Populacao
    Populacao ← CruzamentoPopulacao(Populacao, TaxaCruzamento)
    Populacao ← MutacaoPopulacao(Populacao, TaxaMutacao, FracaoMutantes)
    Populacao ← AvaliarPopulacao(Populacao, PopulacaoAnterior, Grafo, TaxaElitismo)
end
MelhorSolucao ← indivíduo de Populacao com menor valor de aptidão
return MelhorSolucao

A implementação resultante representa uma abordagem balanceada entre a flexibilidade da computação evolutiva e as especificidades combinatórias do PDR2F, permitindo a obtenção de soluções de qualidade em tempo computacional viável mesmo para instâncias de médio e grande porte.

4.4 Heurísticas de Geração de População inicial

A população inicial exerce um papel fundamental no desempenho de algoritmos genéticos, pois influencia diretamente a diversidade das soluções exploradas e a velocidade de convergência do algoritmo. Estratégias de inicialização puramente aleatórias tendem a gerar indivíduos com baixa qualidade inicial, enquanto abordagens excessivamente determinísticas podem limitar a exploração do espaço de busca.

Com o objetivo de equilibrar diversidade e qualidade inicial, o algoritmo genético proposto adota uma estratégia híbrida de inicialização da população, composta por duas heurísticas distintas: uma heurística gulosa e uma inicialização aleatória. Cada uma dessas abordagens é responsável por gerar aproximadamente metade da população total.

4.4.0.1 Inicialização aleatória

Como visto anteriormente, cada indivíduo (cromossomo) é representado por um vetor de tamanho n, onde n corresponde ao número de vértices do grafo. Na inicialização aleatória, cada posição do vetor recebe um valor inteiro escolhido aleatoriamente no conjunto $\{1, 2, 3\}$, representando o peso atribuído ao respectivo vértice. Essa abordagem garante diversidade estrutural na população inicial, permitindo que o algoritmo explore regiões distintas do espaço de busca desde as primeiras gerações.

4.4.0.2 Heurística Gulosa

A heurística gulosa criada no trabalho de (Djukanović *et al.*, 2025), utilizada para a geração de parte da população inicial, constrói indivíduos de forma iterativa, explorando diretamente a estrutura do grafo. O objetivo é atribuir pesos aos vértices de modo a garantir a cobertura de todos os vértices do grafo, priorizando aqueles que apresentam maior capacidade de cobertura em cada iteração do processo.

Inicialmente, todos os vértices são considerados não cobertos e o vetor solução é inicializado com valores nulos. A cada iteração, o algoritmo avalia apenas os vértices ainda não cobertos, calculando, para cada um deles, a quantidade de novos vértices que seriam cobertos caso esse vértice fosse selecionado. Esse conjunto é formado pelo próprio vértice e por seus vizinhos no grafo.

O vértice que maximiza essa quantidade de cobertura adicional é selecionado. O peso atribuído ao vértice escolhido depende do número de vértices recém-cobertos e de sua condição de cobertura, sendo limitado superiormente por um valor máximo previamente definido. Após a atribuição do peso, o conjunto de vértices cobertos é atualizado. O procedimento é repetido até que todos os vértices do grafo estejam cobertos.

Para promover diversidade entre os indivíduos gerados, a ordem de avaliação dos vértices é embaralhada aleatoriamente a cada execução da heurística gulosa. O funcionamento detalhado dessa heurística é formalizado no Algoritmo 4.4.0.2, que descreve o processo de construção iterativa dos indivíduos da população inicial.

Algorithm 3 Pseudocódigo da heurística gulosa para geração da população inicial

Function *InicializarPopulacaoGulosa(Grafo, NumIndividuos)*:
    Input: Grafo, número de indivíduos a serem gerados
    Output: Conjunto de indivíduos gerados pela heurística gulosa
    Populacao $\leftarrow \emptyset$
    $n \leftarrow$ número de vértices de Grafo
    for $k \leftarrow 1$ **to** *NumIndividuos* **do**
        Solucao $\leftarrow$ vetor de tamanho n inicializado com zeros
        Cobertos $\leftarrow \emptyset$
        Vertices $\leftarrow$ vértices de Grafo embaralhados aleatoriamente
        while $|Cobertos| < n$ **do**
            foreach $v \in Vertices$ **do**
                if $v \notin Cobertos$ **then**
                    $Vizinhos \leftarrow N(v) \cup \{v\}$
                    Ganho$(v) \leftarrow |Vizinhos \setminus Cobertos|$
                end
            end
            $v^* \leftarrow$ vértice com maior Ganho
            NovosCobertos $\leftarrow$ vizinhos de $v^* \cup \{v^*\} \setminus Cobertos$
            if $v^* \in NovosCobertos$ **then**
                Indicador $\leftarrow 0$
            end
            else
                Indicador $\leftarrow 1$
            end
            Solucao$[v^*] \leftarrow \min(3,\ \text{Ganho}(v^*) + \text{Indicador})$
            Cobertos $\leftarrow Cobertos \cup NovosCobertos$
        end
        Populacao $\leftarrow Populacao \cup \{\text{Solucao}\}$
    end
    return Populacao

4.4.0.3 Composição da população inicial

A população inicial é formada pela união das duas subpopulações geradas. Formalmente, seja N o tamanho total da população, a população inicial P é definida como: $P = P_{\text{gulosa}} \cup P_{\text{aleatória}}$, onde $|P_{\text{gulosa}}| = \lceil N/2 \rceil$ e $|P_{\text{aleatória}}| = \lfloor N/2 \rfloor$.

Essa estratégia híbrida permite que o algoritmo genético inicie a busca com indivíduos de boa qualidade, sem comprometer a diversidade necessária para evitar convergência prematura.

4.5 Função de Conserto de Viabilidade

No algoritmo genético desenvolvido, é fundamental que toda solução candidata atenda às condições do Problema de Dominação Romana 2-Forte. No entanto, durante a geração e recombinação aleatória de indivíduos, é comum que algumas soluções não atendam a essas regras. Para resolver isso, criamos uma rotina de *conserto de viabilidade* (`fix_instance`) que, dada uma atribuição de rótulos aos vértices, verifica se a solução é válida e, caso não seja, faz ajustes até que todas as condições sejam satisfeitas. Essa abordagem permite que a população permaneça sempre viável, sem precisar descartar soluções promissoras que possam ter pequenas falhas.

4.5.1 Cálculo de Protetores Potenciais

Antes de verificar a validade de uma solução, é necessário identificar, para cada vértice desprotegido (rótulo 0), quais de seus vizinhos possuem capacidade defensiva suficiente para protegê-lo. Definimos o *conjunto de protetores potenciais* (AP — do inglês *available protectors*) de um vértice v como: $\mathrm{AP}(v) = \{u \in N(v) \mid f(u) \geq 2\}$, onde $N(v)$ é o conjunto de vizinhos de v e $f(u)$ é o rótulo atribuído ao vértice u.

A função `calculate_ap` percorre todos os vértices do grafo e, para aqueles com rótulo 0, constrói seu conjunto AP a partir da inspeção dos rótulos dos vizinhos. Vértices com rótulo diferente de zero recebem um conjunto AP vazio, pois já estão autoprotegidos. Essa estrutura auxiliar é fundamental para as verificações subsequentes, pois evita a repetição de consultas à vizinhança e permite identificar rapidamente quais vértices podem receber apoio defensivo.

O pseudocódigo 4 detalha a lógica dessa etapa.

Algorithm 4 Cálculo dos protetores potenciais (AP) para cada vértice

Function *calculate_ap(G)*:
- **foreach** *vértice* $v \in V(G)$ **do**
  - **if** *v possui peso igual a 0* **then**
    - Conjunto de protetores $AP(v) \leftarrow \emptyset$
    - **foreach** *vizinho* $u \in N(v)$ **do**
      - **if** *u possui peso maior ou igual a 2* **then**
        - $AP(v) \leftarrow AP(v) \cup \{u\}$
      - **end**
    - **end**
  - **else**
    - $AP(v) \leftarrow \emptyset$
  - **end**
- **end**
- **return** grafo G atualizado com a função $AP(\cdot)$ para cada vértice

Para ilustrar o funcionamento da função `calculate_ap`, a Figura 11 apresenta um exemplo visual do processo de verificação dos vizinhos de um vértice v desprotegido. Nela, é possível observar como o algoritmo filtra apenas os vértices com capacidade de defesa suficiente para compor o conjunto de protetores potenciais.

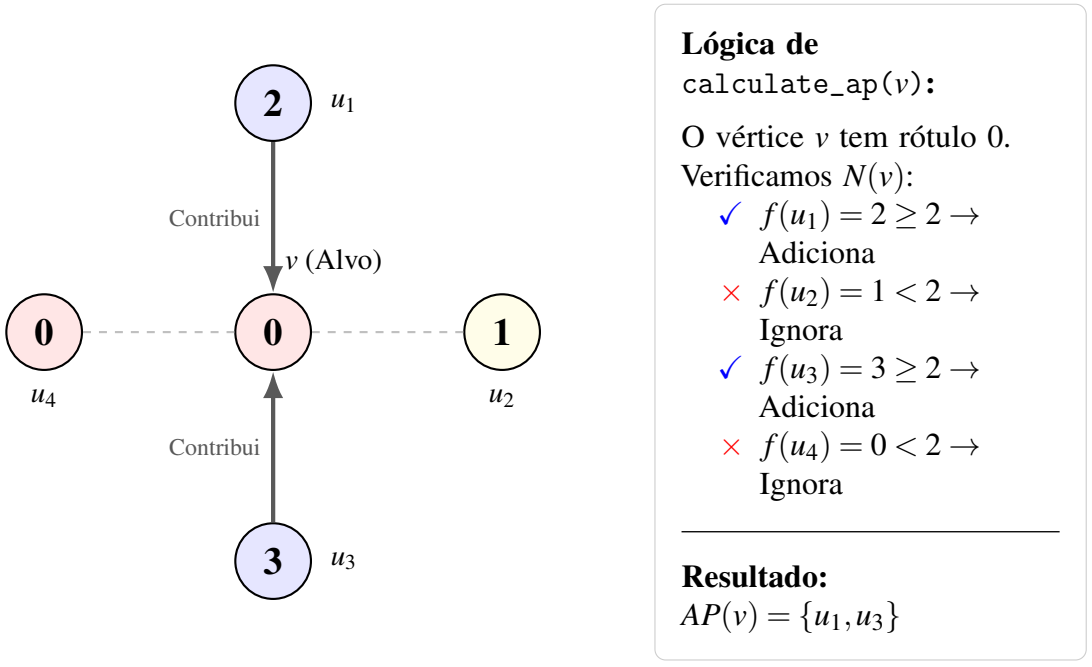


Figura 11 – Exemplo visual da execução da função `calculate_ap`. As setas sólidas indicam vizinhos que satisfazem a condição $f(u) \geq 2$ e, portanto, são adicionados ao conjunto de protetores potenciais do vértice v.
Fonte: Elaborado pelo autor.

4.5.2 Algoritmo de Reparo de Soluções

Após o cálculo dos conjuntos de protetores potenciais (AP), o algoritmo de conserto, denominado `fix_instance`, percorre os vértices do grafo para identificar e corrigir violações das regras de dominação romana 2-forte. O procedimento trata dois tipos principais de inviabilidade:

1. Vértices desprotegidos: Caso um vértice possua rótulo 0 e não tenha nenhum vizinho com rótulo maior ou igual a 2 (isto é, seu conjunto AP seria vazio se calculado naquele momento), ele precisa ser alterado. A heurística adota uma estratégia baseada no grau do vértice: se o vértice possui grau superior a 3, seu rótulo é alterado para 3, tornando-o um forte defensor para seus vizinhos. Caso contrário, seu rótulo é alterado para 1, tornando-o autoprotegido. Essa distinção visa aumentar a eficiência da solução, privilegiando vértices de maior grau para atuarem como defensores.
2. Defensores sobrecarregados: A definição de dominação romana 2-forte estabelece que um vértice com rótulo 2 pode defender, no máximo, um vizinho sob ataque simultâneo (capacidade de defesa $f(v) - 1$). O algoritmo verifica se algum vértice com rótulo 2 é o único protetor de dois ou mais vizinhos (ou seja, vizinhos cujo conjunto AP possui tamanho 1). Se essa situação ocorrer, o rótulo do defensor é incrementado para 3, aumentando sua capacidade de defesa para cobrir a demanda.

Dessa forma, a rotina assegura que o resultado final seja sempre uma solução viável. Se a atribuição inicial de rótulos já for válida, o algoritmo a preserva integralmente; caso contrário, as correções são aplicadas de maneira pontual, priorizando estratégias que evitem o aumento excessivo do custo total da solução.

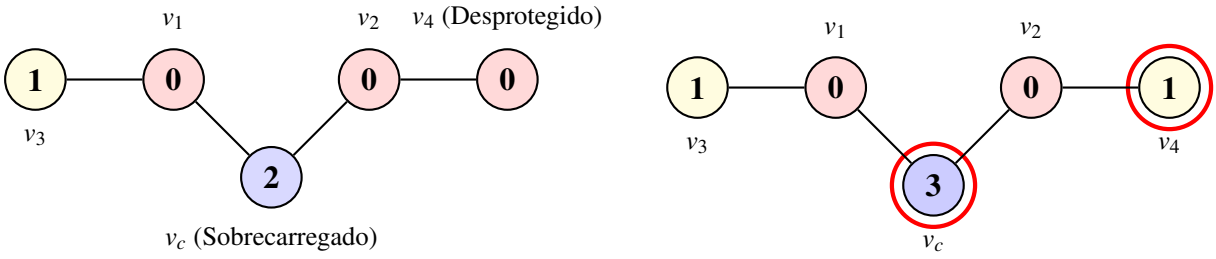
O Algoritmo 5 detalha essa lógica computacional.

Algorithm 5 Algoritmo de reparo de viabilidade (Fix Instance)

Function *fix_instance(G)*:
```
G ← calculate_ap(G)
foreach vértice v ∈ V(G) do
    // Caso 1: Vértice com peso 0 (Vulnerável)
    if f(v) == 0 then
        if não existe u ∈ N(v) tal que f(u) ≥ 2 then
            if grau de v > 3 then
                f(v) ← 3
                Atualizar AP dos vizinhos de v
            else
                f(v) ← 1
            end
        end
    end
    // Caso 2: Vértice com peso 2 (Defensor)
    else if f(v) == 2 then
        contagem_dependentes ← 0
        foreach vizinho u ∈ N(v) do
            if f(u) == 0 e |AP(u)| == 1 then
                contagem_dependentes ← contagem_dependentes + 1
            end
        end
        if contagem_dependentes > 1 then
            f(v) ← 3
        end
    end
end
return G
```

Para ilustrar a atuação prática da rotina de reparo descrita no Algoritmo 5, a Figura 12 apresenta um comparativo entre uma solução inicial inválida e sua versão corrigida. Na Figura 12a, a instância é considerada inviável por dois fatores principais: o vértice v_4 encontra-se totalmente desprotegido, sem vizinhos capazes de defendê-lo, e o vértice central v_c está sobrecarregado. Neste último caso, embora v_c tenha rótulo 2, sua capacidade de defesa extra $f(v_c) - 1 = 1$ é insuficiente para atender à demanda simultânea de dois vizinhos dependentes, v_1 e v_2.

A correção automática é observada na Figura 12b. O algoritmo detectou a vulnerabilidade de v_4 e, devido ao seu baixo grau, alterou seu rótulo para 1, garantindo autoproteção econômica. Simultaneamente, o rótulo de v_c foi incrementado para 3, elevando sua capacidade de defesa para cobrir validamente ambos os vizinhos e assegurar a viabilidade global da solução.



(a) Instância inviável antes do conserto.

(b) Instância viável após o conserto.

Figura 12 – Exemplo de aplicação da função `fix_instance`. Em (a), o grafo apresenta violações. Em (b), os vértices v_c e v_4 (destacados em vermelho) tiveram seus rótulos ajustados pelo algoritmo para garantir a viabilidade da solução.
Fonte: Elaborado pelo autor.

4.6 Estratégia de Elitismo

A estratégia de elitismo foi adotada para preservar as características dos indivíduos de maior qualidade ao longo das gerações, reduzindo o risco de perda de boas soluções devido à estocasticidade dos operadores genéticos. O funcionamento dessa etapa é regido pelo parâmetro `elitism_rate`, que define a porcentagem exata da população atual que deve ser mantida intacta para a próxima iteração.

Na prática, o algoritmo ordena todos os indivíduos da geração atual com base em sua aptidão (do menor para o maior peso). Em seguida, seleciona-se os melhores classificados até atingir a quantidade estipulada e os copia diretamente para a nova geração. As posições restantes na população são, então, preenchidas pelos descendentes gerados pelos processos de cruzamento e mutação. Com isso, assegura-se que o melhor indivíduo encontrado durante todo o processo nunca seja descartado.

O Algoritmo 6 detalha o procedimento de composição da nova geração, combinando a preservação da elite histórica com a inserção dos novos descendentes.

Algorithm 6 Estratégia de Elitismo e Composição da Nova Geração

Function *AplicarElitismo*($Pop_{anterior}$, $Pop_{descendentes}$, $elitism\_rate$)**:**

Input: População da geração anterior, população de descendentes (reparada), taxa de elitismo `elitism_rate`

Output: Nova população composta

$N \leftarrow$ Tamanho($Pop_{anterior}$)

`// Calcula o tamanho da elite`

$Qtd_{elite} \leftarrow \lfloor N \times elitism\_rate \rfloor$

`// O restante será preenchido pelos novos descendentes`

$Qtd_{novos} \leftarrow N - Qtd_{elite}$

$Elite \leftarrow$ `SelecionarMelhores`($Pop_{anterior}$, Qtd_{elite})

$Novos \leftarrow$ `SelecionarMelhores`($Pop_{descendentes}$, Qtd_{novos})

return $Elite \cup Novos$

4.7 Operador de Seleção para Cruzamento

Diferentemente de abordagens clássicas que aplicam pressão seletiva na escolha dos progenitores (como os métodos de Roleta ou Torneio), este trabalho adota uma estratégia de Seleção por Acasalamento Aleatório (*Random Mating*). Nesta abordagem, todos os indivíduos da população atual possuem igual probabilidade de serem escolhidos para a reprodução, independentemente de sua aptidão (*fitness*).

O processo consiste em permutar aleatoriamente a ordem dos indivíduos na população e, em seguida, agrupá-los em pares sequenciais para a aplicação do operador de cruzamento. A escolha dessa estratégia justifica-se pelo forte mecanismo de elitismo empregado na etapa de sobrevivência (descrita na Seção 3.6). Como o algoritmo já descarta agressivamente as piores soluções ao final de cada geração, a seleção aleatória de pais atua como um contrapeso fundamental, promovendo a diversidade genética e permitindo que genes de indivíduos menos aptos sejam recombinados, evitando a convergência para ótimos locais.

A Figura 13 ilustra o fluxo deste operador.

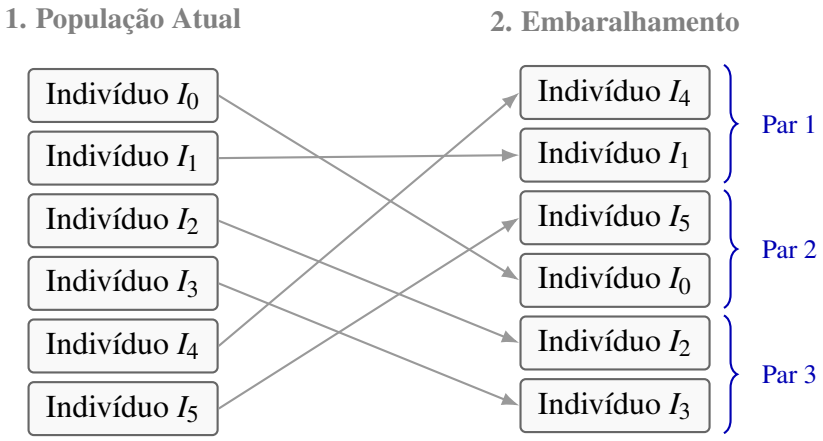


Figura 13 – Representação do processo de seleção aleatória. A população é embaralhada e, em seguida, os pais são formados por pares adjacentes sequenciais, sem viés de aptidão nesta etapa.
Fonte: Elaborado pelo autor.

4.8 Operadores de Cruzamento

Neste trabalho, optou-se por uma abordagem dinâmica de operação de cruzamento (*crossover*) que alterna aleatoriamente entre dois tipos clássicos de operadores: o *Cruzamento de Ponto Único* e o *Cruzamento Uniforme*.

Essa estratégia híbrida visa equilibrar a capacidade de exploração do algoritmo. Enquanto o cruzamento de ponto único tende a preservar blocos adjacentes de genes, o cruzamento uniforme promove uma mistura mais granular dos cromossomos.

A rotina de cruzamento opera da seguinte forma: para cada par de pais selecionados, verifica-se se a recombinação ocorrerá com base na taxa de cruzamento (P_c). Caso positivo, um dos dois operadores é sorteado equiparavelmente para gerar os filhos. Caso contrário, os pais são clonados diretamente para a próxima etapa.

4.8.1 Cruzamento de Ponto Único

O operador de cruzamento de ponto único fundamenta-se na troca de segmentos contíguos de genes entre dois cromossomos. O processo inicia-se com a escolha aleatória de um ponto de corte k ao longo do cromossomo (variando entre o primeiro e o penúltimo gene). Os descendentes são gerados combinando-se as partes: o primeiro filho herda os genes do Pai 1 anteriores ao corte e os genes do Pai 2 posteriores a ele, enquanto o segundo filho realiza a composição inversa. Essa técnica tem como principal característica a preservação da ordem e da adjacência das informações genéticas, mantendo intactos blocos de genes que podem ser

benéficos para a solução. Veja a figura 14 que ilustra um pouco da ideia do crossover.

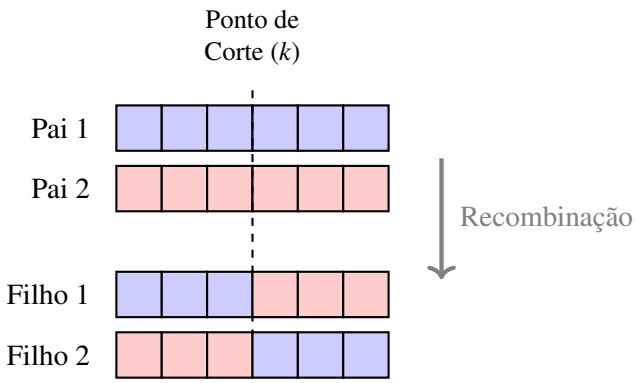


Figura 14 – Esquema do Cruzamento de Ponto Único. O corte divide os pais em dois segmentos, que são trocados para formar os descendentes.
Fonte: Elaborado pelo autor.

Os Algoritmo 7 detalha o processo de crossover de ponto único.

Algorithm 7 Operador de Cruzamento de Ponto Único

Function *CrossoverPontoUnico*(P_1, P_2)**:**
- **Input:** Dois indivíduos pais P_1 e P_2 de tamanho L
- **Output:** Dois indivíduos filhos F_1 e F_2
- `// Escolhe ponto de corte entre 1 e L-1`
- $k \leftarrow \text{RandomInteiro}(1, L-1)$
- $F_1 \leftarrow \text{concatena}(P_1[0 \ldots k-1], P_2[k \ldots L-1])$
- $F_2 \leftarrow \text{concatena}(P_2[0 \ldots k-1], P_1[k \ldots L-1])$
- **return** F_1, F_2

4.8.2 Cruzamento Uniforme

O operador de cruzamento uniforme adota uma abordagem mais granular em comparação ao de ponto único, tratando cada gene de forma independente. O processo percorre todo o comprimento do cromossomo e, para cada posição, decide probabilisticamente (com 50% de chance para cada pai) qual gene será herdado pelo primeiro descendente, sendo o gene restante atribuído ao segundo filho. Essa estratégia promove uma mistura intensa do material genético e evita o viés de posição, permitindo a recombinação de genes distantes entre si, embora apresente uma tendência maior a romper blocos construtivos já estabelecidos.

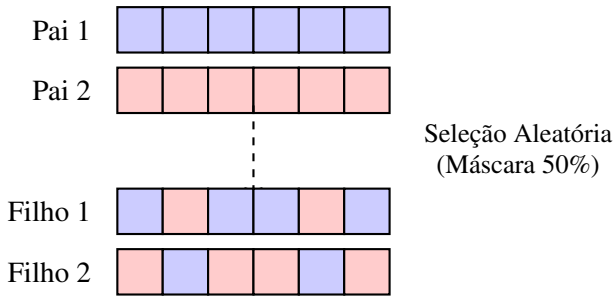


Figura 15 – Esquema do Cruzamento Uniforme. Cada gene é selecionado independentemente de um dos pais, gerando descendentes com alta variabilidade genética.
Fonte: Elaborado pelo autor.

O pseudocódigo 8 mostra como essa rotina é implementada.

Algorithm 8 Operador de Cruzamento Uniforme

Function *CrossoverUniforme*(P_1, P_2)**:**
  Input: Dois indivíduos pais P_1 e P_2 de tamanho L
  Output: Dois indivíduos filhos F_1 e F_2
  $F_1 \leftarrow$ vetor vazio de tamanho L
  $F_2 \leftarrow$ vetor vazio de tamanho L
  for $i \leftarrow 0$ **to** $L-1$ **do**
    `// Máscara aleatória para cada gene (50% chance)`
    if *Random(0, 1)* < 0.5 **then**
      $F_1[i] \leftarrow P_1[i]$
      $F_2[i] \leftarrow P_2[i]$
    else
      $F_1[i] \leftarrow P_2[i]$
      $F_2[i] \leftarrow P_1[i]$
    end
  end
  return F_1, F_2

4.8.3 Rotina Geral de Cruzamento da População

A etapa de reprodução é coordenada por uma rotina principal, detalhada no Algoritmo 9. O processo começa com o embaralhamento de toda a população, garantindo que os pares de pais sejam formados de maneira aleatória e imparcial.

Para cada par selecionado, o algoritmo verifica se o cruzamento deve ocorrer com base na probabilidade definida pelo parâmetro `crossover_rate`. Se o cruzamento for confirmado, o sistema escolhe dinamicamente qual operador utilizar, sorteando entre o Ponto Único e o Uniforme com chances iguais. Essa alternância automática permite que o algoritmo combine, em uma mesma execução, a capacidade de preservação de blocos do Ponto Único com o poder

de mistura do Uniforme. Caso o cruzamento não ocorra, os pais são apenas copiados para a próxima etapa. O pseudocódigo 9 mostra esse processo.

Algorithm 9 Rotina Geral de Cruzamento da População

```
Function ExecutarCruzamento(Populacao, crossover_rate):
    Input: População atual, taxa de cruzamento crossover_rate
    Output: Nova população de descendentes
    NovaPop ← ∅
    Populacao ← Embaralhar(Populacao)
    // Processa os indivíduos em pares
    for i ← 0 to Tamanho(Populacao) - 1 passo 2 do
        P1 ← Populacao[i]
        P2 ← Populacao[i+1]
        if Random(0, 1) < crossover_rate then
            // Sorteia qual operador utilizar
            if RandomChoice({SinglePoint, Uniform}) == SinglePoint then
                F1, F2 ← CrossoverPontoUnico(P1, P2)
            else
                F1, F2 ← CrossoverUniforme(P1, P2)
            end
        else
            // Sem cruzamento: clonagem
            F1, F2 ← P1, P2
        end
        Adicionar F1, F2 em NovaPop
    end
    return NovaPop
```

4.9 Operador de Mutação

O operador de mutação desempenha o papel fundamental de introduzir diversidade genética na população, permitindo que o algoritmo escape de ótimos locais e explore novas regiões do espaço de busca. Neste trabalho, implementou-se uma estratégia de *mutação gene a gene com viés de redução*, aplicada apenas a uma fração da população.

O processo ocorre em duas etapas. Primeiramente, seleciona-se aleatoriamente um subconjunto dos indivíduos da população atual. A quantidade de indivíduos a serem mutados é determinada pelo parâmetro `n_mutants` (definido anteriormente como a fração da população sujeita à mutação). Para cada indivíduo selecionado, percorre-se todos os seus genes e, com base na probabilidade de mutação por gene definida pelo parâmetro `gene_mutation_rate`, aplica-se uma alteração no valor do gene.

Diferentemente de abordagens clássicas que atribuem um valor aleatório ao gene

mutado, este operador atua especificamente reduzindo os pesos dos vértices: rótulos 3 são alterados para 2, e rótulos 1 são alterados para 0. O objetivo dessa estratégia agressiva é forçar a redução do custo total da solução. Embora isso possa gerar soluções momentaneamente inviáveis, confia-se na etapa subsequente de reparo (descrita na Seção 4.5) para restaurar a viabilidade da estrutura, muitas vezes encontrando configurações mais econômicas do que as originais.

A Figura 16 ilustra a alteração dos valores nos genes selecionados.

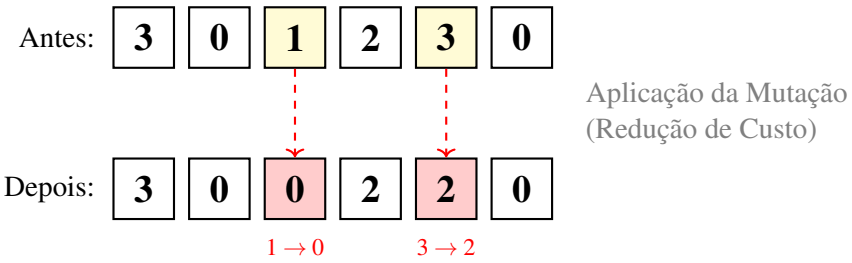


Figura 16 – Exemplo do operador de mutação redutora. Os genes selecionados (destacados) têm seus pesos diminuídos para tentar minimizar o custo da função objetivo.
Fonte: Elaborado pelo autor.

O procedimento computacional para a implementação deste operador é apresentado no Algoritmo 10.

Algorithm 10 Operador de Mutação com Viés de Redução

Function *MutacaoPopulacao(Populacao, gene_mutation_rate, n_mutants)*:

Input: População atual, taxa de mutação por gene `gene_mutation_rate`, fração de mutantes `n_mutants`

Output: População com indivíduos mutados

$N \leftarrow$ Tamanho(*Populacao*)

`// Calcula quantos indivíduos sofrerão mutação`

$Qtd_{mutantes} \leftarrow \lceil N \times n\_mutants \rceil$

`// Seleciona aleatoriamente quais indivíduos da população serão afetados`

Indices $\leftarrow$ EscolherAleatoriamente($0 \ldots N-1$, $Qtd_{mutantes}$)

for *i* ***em*** *Indices* **do**

  for $j \leftarrow 0$ **to** *TamanhoCromossomo - 1* **do**

    `// Verifica probabilidade de mutação para cada gene`

    if *Random(0, 1)* $<$ *gene_mutation_rate* **then**

      if $Populacao[i][j] == 3$ **then**

        $Populacao[i][j] \leftarrow 2$

      else if $Populacao[i][j] == 1$ **then**

        $Populacao[i][j] \leftarrow 0$

    end

  end

end

return *Populacao*

5 ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS ENVIESADAS

Conforme detalhado na revisão teórica apresentada no Capítulo 2, o algoritmo BRKGA (*Biased Random-Key Genetic Algorithm*) distingue-se por separar a parte evolutiva da parte dependente do problema. Enquanto os operadores de seleção, cruzamento enviesado e mutação operam de maneira padronizada sobre cromossomos de chaves aleatórias (vetores de números reais $x \in [0,1]$), a especificidade da aplicação reside inteiramente no decodificador (*decoder*).

Desta forma, este capítulo omite a discussão do BRKGA já fundamentada anteriormente na Seção 2.4 e concentra-se exclusivamente na definição da estratégia de codificação e no algoritmo de decodificação desenvolvido para o PDR2F. O objetivo é demonstrar como o vetor de chaves aleatórias é mapeado para uma configuração de pesos no grafo e como a viabilidade dessa solução é assegurada.

5.1 Codificador e Decodificador

No contexto deste algoritmo, cada indivíduo é representado por um cromossomo (vetor contendo N genes), onde N corresponde ao número total de vértices do grafo analisado. Cada gene g_i é um número real gerado no intervalo contínuo $[0,1]$. O papel fundamental do componente decodificador é transformar esses valores de ponto flutuante em uma configuração discreta de pesos $w_i \in \{0,1,2,3\}$, compatível com a definição do Problema de Dominação Romana.

Para realizar essa transformação, o intervalo unitário foi particionado em quatro subintervalos de igual amplitude. A regra de mapeamento associa cada faixa de valores a um peso específico, conforme definido pela Equação 5.1:

$$w_i = \begin{cases} 0 & \text{se } 0,00 \leq g_i < 0,25 \\ 1 & \text{se } 0,25 \leq g_i < 0,50 \\ 2 & \text{se } 0,50 \leq g_i < 0,75 \\ 3 & \text{se } 0,75 \leq g_i \leq 1,00 \end{cases} \tag{5.1}$$

Essa abordagem permite que o algoritmo explore o espaço de busca, enquanto o decodificador atua como uma interface para o espaço de soluções discretas. A Figura 17 ilustra

visualmente o particionamento da reta real utilizado para a determinação dos pesos.

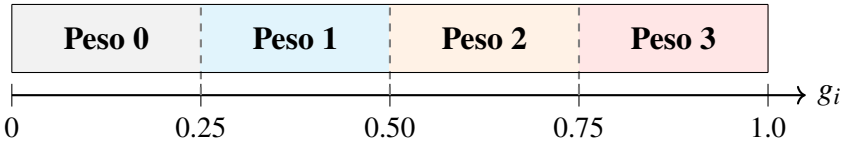


Figura 17 – Representação dos intervalos de decodificação. O valor do gene (g_i) determina o peso atribuído ao vértice correspondente através de quartis.
Fonte: Elaborado pelo autor.

O Algoritmo 11 descreve o processo de decodificação, transformando as chaves reais em pesos discretos conforme os intervalos definidos. Em contrapartida, o Algoritmo 12 apresenta a codificação, que realiza o caminho inverso para converter soluções inteiras em cromossomos válidos.

Algorithm 11 Decodificador: Transforma chaves aleatórias em pesos

Function *Decodificar(Cromossomo)*:
- **Input:** Vetor chamado *Cromossomo* com valores em $[0,1]$
- **Output:** Vetor chamado *Pesos* com valores em $\{0,1,2,3\}$
- $N \leftarrow$ Tamanho(*Cromossomo*) *Pesos* $\leftarrow$ vetor de inteiros de tamanho N
- **for** $i \leftarrow 0$ **to** $N-1$ **do**
  - $val \leftarrow Cromossomo[i]$
  - **if** $val < 0.25$ **then**
    - $Pesos[i] \leftarrow 0$
  - **else if** $val < 0.50$ **then**
    - $Pesos[i] \leftarrow 1$
  - **else if** $val < 0.75$ **then**
    - $Pesos[i] \leftarrow 2$
  - **else**
    - $Pesos[i] \leftarrow 3$
- **end**
- **return** *Pesos*

Algorithm 12 Codificador: Transforma pesos discretos em chaves aleatórias

Function *Codificar(Pesos)*:
  Input: Vetor de *Pesos* com valores em $\{0,1,2,3\}$
  Output: Vetor *Cromossomo* (x) com valores em $[0,1]$
  $N \leftarrow$ Tamanho(*Pesos*)
  Cromossomo $\leftarrow$ vetor de reais de tamanho N
  for $i \leftarrow 0$ **to** $N-1$ **do**
    if $Pesos[i] == 0$ **then**
      $Cromossomo[i] \leftarrow 0.10$
    else if $Pesos[i] == 1$ **then**
      $Cromossomo[i] \leftarrow 0.30$
    else if $Pesos[i] == 2$ **then**
      $Cromossomo[i] \leftarrow 0.60$
    else if $Pesos[i] == 3$ **then**
      $Cromossomo[i] \leftarrow 0.90$
  end
  return *Cromossomo*

5.2 Avaliação e Tratamento de Viabilidade

Uma característica intrínseca do decodificador proposto é que ele opera gene a gene, de forma independente e sem considerar a topologia do grafo. Consequentemente, o vetor de pesos resultante da decodificação direta frequentemente viola a condição de Dominação Romana $2-$forte, gerando soluções estruturalmente inviáveis.

Para contornar essa limitação sem restringir a liberdade de exploração do algoritmo genético, adota-se uma estratégia de reparo. Imediatamente após a etapa de decodificação, a solução candidata é submetida à heurística de reparo (`fix_instance`), cuja lógica foi previamente detalhada na Seção 4.5.

O fluxo de avaliação consolida-se, portanto, em uma sequência integrada: o cromossomo é decodificado; em seguida, os pesos são ajustados para garantir a cobertura de todos os vértices; e, por fim, calcula-se a aptidão somando-se os pesos da solução já reparada. Essa abordagem híbrida permite que o BRKGA explore o espaço contínuo de chaves aleatórias.

O Algoritmo 13 formaliza este procedimento, que atua como a função objetivo invocada pelo framework a cada geração.

Algorithm 13 Fluxo de Avaliação e Reparo no BRKGA

Function *AvaliarIndividuo(*$Cromossomo, Grafo$*)*:

Input: Vetor de chaves aleatórias *Cromossomo*, estrutura do $Grafo$

Output: Valor de aptidão (Fitness)

`// Transforma genótipo em fenótipo bruto`

$Pesos_{brutos} \leftarrow$ `Decodificar(`$Cromossomo$`)`

`// Ajusta pesos para garantir viabilidade (Heurística)`

$Solucao_{viavel} \leftarrow$ `fix_instance(`$Pesos_{brutos}, Grafo$`)`

`// Calcula o custo total da solução válida`

$Fitness \leftarrow \sum_{v \in V(Grafo)} Solucao_{viavel}[v]$

return $Fitness$

6 NOVA PROPOSTA DE PROGRAMA INTEIRO

Neste capítulo, apresentamos uma nova formulação matemática para o Problema da Dominação Romana 2-Forte (PDR2F). Diferentemente de abordagens presentes na literatura para problemas correlatos, como no trabalho de (Liu *et al.*, 2020), que frequentemente dependem da enumeração de subconjuntos de vértices (conjuntos de ataque ou defesa) ou do uso de constantes Big-M associadas a cortes combinatórios, a proposta a seguir modela as condições específicas da dominação romana 2-forte através de um conjunto polinomial (linear no tamanho do grafo de entrada) de variáveis auxiliares e restrições locais. A formulação aqui apresentada é baseada na definição apresentada na Seção 2.2.1.

6.1 Variáveis de Decisão

O modelo proposto utiliza três classes de variáveis binárias para capturar o estado dos vértices e as relações de vizinhança necessárias para satisfazer as condições da definição de função de dominação romana 2-forte.

- Variáveis de Estado ($z_{v,k}$): Para todo $v \in V(G)$ e $k \in \{0,1,2,3\}$, a variável binária $z_{v,k}$ indica o rótulo atribuído ao vértice:

$$z_{v,k} = 1 \iff f(v) = k$$

- Variáveis de Contagem de Protetores ($q_{u,k}$): Para todo $u \in V(G)$ e $k \in \{0,1,2\}$, estas variáveis binárias categorizam a quantidade de vizinhos fortes (rótulo ≥ 2) que um vértice u possui:
  - $q_{u,0} = 1 \iff u$ não possui vizinhos protetores.
  - $q_{u,1} = 1 \iff u$ possui exatamente 1 vizinho protetor.
  - $q_{u,2} = 1 \iff u$ possui 2 ou mais vizinhos protetores.
- Variáveis de Proteção Exclusiva ($a_{u,v}$): Para todo $u \in V(G)$ e vizinho $v \in N(u)$, a variável binária $a_{u,v}$ identifica se v é o único responsável por defender u. Formalmente, $a_{u,v} = 1$ se, e somente se, $f(u) = 0$, $f(v) = 2$ e v é o único vizinho de u com rótulo ≥ 2 (ou seja, $PN(u) = \{v\}$).

6.2 Modelo Matemático

O problema pode ser formulado pelo seguinte Programa Linear Inteiro:

$$\min \quad \sum_{u \in V} (z_{u,1} + 2z_{u,2} + 3z_{u,3}) \tag{6.1}$$

$$\text{s.t.} \quad z_{u,0} + z_{u,1} + z_{u,2} + z_{u,3} = 1 \qquad \forall u \in V \tag{R1}$$

$$z_{u,0} = 0 \qquad \forall u \in V, deg(u) = 0 \tag{R1.1}$$

$$\sum_{w \in N(u)} (z_{w,2} + z_{w,3}) \geq z_{u,0} \qquad \forall u \in V, deg(u) > 0 \tag{R2}$$

$$\sum_{w \in N(u)} a_{w,u} \leq z_{u,2} \qquad \forall u \in V, deg(u) > 0 \tag{R3}$$

$$q_{u,0} + q_{u,1} + q_{u,2} = 1 \qquad \forall u \in V, deg(u) > 0 \tag{R4.1}$$

$$\sum_{w \in N(u)} (z_{w,2} + z_{w,3}) \leq 0 + deg(u)(1 - q_{u,0}) \qquad \forall u \in V, deg(u) > 0 \tag{R4.2}$$

$$\sum_{w \in N(u)} (z_{w,2} + z_{w,3}) \leq 1 + deg(u)(1 - q_{u,1}) \qquad \forall u \in V, deg(u) > 0 \tag{R4.3}$$

$$\sum_{w \in N(u)} (z_{w,2} + z_{w,3}) \geq 1 - deg(u)(1 - q_{u,1}) \qquad \forall u \in V, deg(u) > 0 \tag{R4.4}$$

$$\sum_{w \in N(u)} (z_{w,2} + z_{w,3}) \geq 2q_{u,2} \qquad \forall u \in V, deg(u) > 0 \tag{R4.5}$$

$$\sum_{w \in N(u)} (z_{w,2} + z_{w,3}) \leq 1 + (deg(u) - 1) \cdot q_{u,2} \qquad \forall u \in V, deg(u) > 0 \tag{R4.6}$$

$$a_{u,w} \leq z_{u,0} \qquad \forall u \in V, w \in N(u) \tag{R5}$$

$$a_{u,w} \leq z_{w,2} \qquad \forall u \in V, w \in N(u) \tag{R6}$$

$$a_{u,w} \leq q_{u,1} \qquad \forall u \in V, w \in N(u) \tag{R7}$$

$$a_{u,w} \geq z_{u,0} + z_{w,2} + q_{u,1} - 2 \qquad \forall u \in V, w \in N(u) \tag{R8}$$

$$z_{u,k} \in \{0,1\} \qquad \forall u \in V, k \in \{0,\ldots,3\} \tag{R9}$$

$$q_{u,k} \in \{0,1\} \qquad \forall u \in V, k \in \{0,1,2\} \tag{R10}$$

$$a_{u,w} \in \{0,1\} \qquad \forall u \in V, w \in N(u) \tag{R11}$$

A Função Objetivo (6.1) minimiza o somatório dos pesos atribuídos aos vértices. O conjunto de restrições modela as regras do problema da seguinte maneira:

– Unicidade e Rótulos (Restrições R1, R1.1): A restrição (R1) assegura que cada vértice receba exatamente um rótulo do conjunto $\{0,1,2,3\}$. A restrição (R1.1) trata o caso particular de vértices isolados: como eles não possuem vizinhos para defendê-los, não podem receber rótulo 0.

- Condição de Dominação Romana (R2): Implementa a condição (a) da definição dada na Seção 2.2.1. Se um vértice u possui rótulo 0 ($z_{u,0} = 1$), a soma das variáveis que indicam vizinhos com peso ≥ 2 deve ser pelo menos 1.
- Contagem de Vizinhos Fortes (R4.1 – R4.6): Este bloco de restrições lineariza a contagem de vizinhos protetores para definir as variáveis $q_{u,k}$. As equações utilizam o grau do vértice, $deg(u)$, para ativar ou desativar as desigualdades. Em suma, elas garantem que $q_{u,1} = 1$ se e somente se o vértice u tem *exatamente* um vizinho com rótulo ≥ 2.
- Definição de Proteção Exclusiva (R5 – R8): Estas restrições definem a variável $a_{u,w}$ através da lógica booleana $a_{u,w} \iff (z_{u,0} \wedge z_{w,2} \wedge q_{u,1})$. Ou seja, $a_{u,w}$ será 1 apenas se u tiver peso 0, w tiver peso 2, e u tiver exatamente um protetor (que, por consequência, deve ser w).
- Condição 2-Forte (R3): Implementa a condição (b) da definição. Ela limita que, se um vértice u tem rótulo 2 ($z_{u,2} = 1$), ele pode ser o protetor exclusivo de no máximo um vizinho. Isso é feito somando as variáveis $a_{w,u}$ para todos os vizinhos w de u.

O número de variáveis e restrições do modelo acima é linear em função do número de vértices e arestas:

- Variáveis: $4|V|$ variáveis z, $3|V|$ variáveis q e $2|E|$ variáveis a. Total: $7|V| + 2|E|$.
- O número de restrições é, no máximo, $9|V| + 8|E|$ (desconsiderando-se as restrições de integralidade).

7 EXPERIMENTOS E RESULTADOS OBTIDOS

Neste capítulo, são apresentados e discutidos os resultados computacionais obtidos pelas meta-heurísticas desenvolvidas (GA e BRKGA) em comparação com o método exato. A análise é conduzida de forma segmentada, onde cada família de instâncias é avaliada em uma subseção específica, permitindo observar o comportamento dos algoritmos frente a diferentes características topológicas. Vale ressaltar que algumas das classes de grafos utilizadas contêm instâncias comuns aos experimentos da literatura, especificamente os do algoritmo VNS (Djukanović *et al.*, 2025), o que possibilita estabelecer comparativos com trabalhos externos.

7.1 Ambiente Computacional e Ferramentas

Todos os experimentos foram conduzidos em um computador pessoal equipado com sistema operacional Linux (Ubuntu), processador Intel Core i7-1185G7 (11ª Geração) operando a 3.00GHz e 16 GB de memória RAM. Embora o equipamento disponha de múltiplos núcleos, a execução dos testes de desempenho (tempo de CPU) foi realizada considerando o tempo de processamento efetivo.

A implementação dos algoritmos foi realizada integralmente na linguagem de programação Python em sua versão 3.12, utilizando o gerenciador de projetos e dependências *UV* para garantir a reprodutibilidade do ambiente. As principais bibliotecas utilizadas foram:

- Manipulação de Dados e Grafos: A biblioteca *NetworkX* foi utilizada para a representação das estruturas de grafos e cálculo de métricas topológicas, enquanto a biblioteca *NumPy* foi empregada para operações vetoriais de alto desempenho, essenciais para a manipulação dos cromossomos no algoritmo genético.
- Algoritmo Genético (BRKGA): A meta-heurística foi construída sobre o framework *pymoo* (Multi-objective Optimization in Python). O código foi adaptado para explorar o paralelismo da máquina através do módulo *multiprocessing*, permitindo a avaliação simultânea de múltiplos indivíduos da população.
- Método Exato (PI): O modelo de Programação Inteira foi implementado utilizando a biblioteca *docplex* (Decision Optimization CPLEX Modeling for Python). Esta biblioteca atua como interface para o solver comercial IBM ILOG CPLEX Optimization Studio, reconhecido por sua eficiência na resolução de problemas combinatórios complexos.
- Calibragem de Parâmetros: Para o ajuste fino dos hiperparâmetros do algoritmo genético,

utilizou-se a ferramenta *irace* (Iterated Racing for Automatic Algorithm Configuration), que implementa métodos estatísticos de corrida (racing methods) para encontrar configurações robustas de forma automatizada.

7.2 Base de Dados

Para avaliar a robustez e a eficiência dos métodos propostos, compilou-se um conjunto de testes composto por 244 grafos com diferentes características topológicas, variando desde estruturas de pequeno porte até redes com milhares de vértices. A seleção destas instâncias teve como objetivo cobrir diferentes classes de complexidade, permitindo analisar o comportamento do algoritmo genético e do modelo exato sob diversas condições de esparsidade e conectividade. As instâncias utilizadas foram categorizadas em três grupos principais, detalhados a seguir.

7.2.1 Instâncias da Literatura de Dominação Romana k-Forte

O primeiro grupo de instâncias provém dos experimentos realizados por. (Liu *et al.*, 2020), que definiram o problema da Dominação Romana k-Forte. A utilização destes grafos permite uma comparação direta com os resultados obtidos. Este conjunto inclui:

- Geoinstances (137 grafos): Grafos geométricos ($G(n,r)$), construídos a partir da distribuição de pontos em um espaço métrico bi-dimensional. Neste modelo, dois vértices são conectados por uma aresta se a distância euclidiana entre eles for menor ou igual a um limiar r. Devido à sua natureza espacial, estes grafos exibem propriedades topológicas similares a mapas geográficos e redes físicas, sendo amplamente utilizados para modelar problemas onde a localização e a vizinhança física são determinantes, como em redes de sensores sem fio e logística.
- Wireless Networks (16 grafos): Instâncias que modelam redes sem fio ad-hoc, caracterizadas por topologias dinâmicas e conectividade baseada em alcance de sinal.
- Random Graphs (35 grafos): Grafos gerados aleatoriamente utilizados pelos autores para validar a performance média dos algoritmos em cenários com distribuição uniforme de arestas, sem tendências topológicas predefinidas.

7.2.2 Grafos Cúbicos Aleatórios

O segundo grupo é composto por 26 grafos cúbicos aleatórios (*Random Cubic Graphs*) gerados especificamente neste trabalho, cujas definições e propriedades foram apresentadas na Seção 2.1.

A inclusão desta classe justifica-se pelo desafio específico que ela impõe às meta-heurísticas de dominação romana. Como cada vértice possui um número fixo e baixo de vizinhos, a capacidade de um vértice com rótulo 2 (ou 3) defender seus vizinhos é estritamente limitada. A baixa densidade de arestas restringe a propagação da defesa, criando mínimos locais que dificultam a convergência para ótimos globais e forçam o algoritmo a encontrar configurações muito precisas para cobrir todo o grafo sem desperdício de recursos.

7.2.3 Coleção Harwell-Boeing

Para avaliar a escalabilidade dos algoritmos em instâncias de maior porte e provenientes de aplicações reais, selecionaram-se 30 grafos da coleção *Harwell-Boeing*.

Originalmente criada para testar algoritmos de matrizes esparsas em computação científica (como dinâmica dos fluidos e engenharia estrutural), esta coleção tornou-se um padrão ouro (*benchmark*) na área de otimização combinatória e teoria dos grafos (Duff *et al.*, 1989). Neste trabalho, foram utilizadas instâncias variando de aproximadamente 100 até mais de 5000 vértices. A diversidade estrutural destes grafos serve como um teste de estresse para verificar se o GA, o BRKGA e o modelo exato conseguem lidar com o aumento exponencial do espaço de busca.

Tabela 2 – Resumo das instâncias utilizadas nos experimentos.

Classe	**Qtd.**	**Origem/Referência**	**Característica Principal**
Geoinstances	137	Liu et al. (Liu *et al.*, 2020)	Geométricos / Sensores
Random (Filtered)	35	Liu et al. (Liu *et al.*, 2020)	Estrutura Aleatória
Wireless Networks	16	Liu et al. (Liu *et al.*, 2020)	Redes Ad-hoc
Random Cubic	26	Gerados	Regularidade ($d = 3$) / Esparso
Harwell-Boeing	30	NIST / Matrix Market	Aplicações Reais
Total	**244**	-	-

7.3 Calibração dos Parâmetros das Meta-Heurísticas

A comparação entre as abordagens propostas fundamenta-se na análise conjunta de três indicadores de desempenho: a qualidade da solução (fitness), mensurada pelo peso total

da função de dominação; a eficiência computacional, avaliada pelo tempo de processamento; e o gap relativo, métrica que quantifica o desvio percentual da solução heurística em relação ao resultado obtido pelo modelo matemático exato (considerado como baseline).

7.4 Calibragem Automatizada dos Parâmetros

A definição manual dos valores de hiperparâmetros do GA e BRKGA é sensível. Se realizada por meio de tentativa e erro, pode levar a configurações subótimas e conclusões enviesadas. Para mitigar esse problema e explorar o potencial máximo dos algoritmos propostos, utilizou-se o software **irace** (*Iterated Racing for Automatic Algorithm Configuration*) (Lopez-Ibanez *et al.*, 2016). O *irace* implementa um procedimento de corrida iterada (iterated racing) que testa automaticamente diversas configurações, descartando estatisticamente aquelas que apresentam desempenho inferior e concentrando o orçamento computacional nas configurações mais promissoras.

7.4.1 Configuração do Experimento de Ajuste

O processo de calibragem foi realizado utilizando um subconjunto representativo de 47 instâncias selecionadas da base de dados descrita na Seção 7.2. A escolha dessas instâncias buscou cobrir a diversidade estrutural dos grafos (diferentes densidades e tamanhos) para evitar o sobreajuste (*overfitting*) a uma classe específica de problema.

O orçamento computacional para o ajuste foi definido em um máximo de 200 experimentos (`maxExperiments`) por instância. Cada configuração candidata gerada pelo *irace* foi executada através de um script (*target-runner*) com um tempo limite de 900 segundos. O processo foi paralelizado em 5 núcleos de processamento para acelerar a convergência.

7.4.2 Parâmetros do Algoritmo Genético (GA)

Para o Algoritmo Genético padrão, o espaço de busca de parâmetros envolveu o tamanho da população, taxas de operação genética e elitismo. Ressalta-se que o número máximo de gerações não foi submetido ao processo de calibração automática, sendo fixado em 100 gerações para todos os experimentos, atuando como o único critério de parada do algoritmo. A Tabela 3 apresenta os intervalos investigados pelo *irace* e a melhor configuração encontrada, que foi posteriormente utilizada em todos os experimentos finais.

Observa-se que o algoritmo convergiu para uma população grande (300 indivíduos) e uma taxa de cruzamento elevada (≈ 0.89), indicando que a recombinação de material genético é um componente crucial para a busca neste problema.

Tabela 3 – Espaço de busca e melhores parâmetros encontrados para o Algoritmo Genético (GA).

Parâmetro	Descrição	Intervalo de Busca	Melhor Valor
`population_size`	Tamanho da População	$\{100, 150, 200, 250, 300\}$	**300**
`crossover_rate`	Taxa de Cruzamento	$[0.5, 0.9]$	**0.8921**
`elitism_rate`	Taxa de Elitismo	$[0.1, 0.5]$	**0.4803**
`n_mutants`	Taxa de Indivíduos Mutantes	$[0.1, 0.5]$	**0.4188**
`gene_mutation_rate`	Taxa de Mutação (Gene)	$[0.01, 0.5]$	**0.2825**

7.4.3 Parâmetros do BRKGA

Para o BRKGA, a calibragem focou nas definições estruturais da população (tamanho das partições de elite e mutantes) e no viés do cruzamento. Assim como no GA, o critério de parada foi fixado deterministicamente em 100 gerações, sem a utilização de mecanismos de interrupção por estagnação, garantindo a isonomia nas condições de teste entre as abordagens. Diferentemente do GA clássico, o BRKGA possui menos parâmetros, mas sua eficiência depende estritamente do equilíbrio entre a exploração (fornecida pelos mutantes) e a intensificação (preservação da elite).

A Tabela 4 detalha os resultados obtidos pelo *irace*. A configuração vencedora também optou pelo limite superior do tamanho da população (300). Nota-se que a taxa de mutantes resultante ($\approx 6\%$) ficou próxima ao limite inferior do intervalo de busca, sugerindo que, para o PDR2F, um fluxo excessivo de aleatoriedade pode prejudicar a convergência.

Tabela 4 – Espaço de busca e melhores parâmetros encontrados para o BRKGA.

Parâmetro	Descrição	Intervalo de Busca	Melhor Valor
`population_size`	Tamanho da População (p)	$\{100, 150, 200, 250, 300\}$	**300**
`bias`	Viés do Cruzamento ($Prob > 0.5$)	$[0.6, 0.9]$	**0.7106**
`n_elites`	Taxa de Elite (p_e)	$[0.1, 0.4]$	**0.13** (39 ind.)
`n_mutants_brkga`	Taxa de Mutantes (p_m)	$[0.05, 0.4]$	**0.063** (19 ind.)

*O número de descendentes (p_o) é derivado automaticamente: $p - p_e - p_m = 242$.

7.5 Análise dos Experimentos em Grafos Cúbicos Aleatórios

A Tabela 5 detalha os experimentos realizados na classe *Random Cubic Graphs*. Conforme antecipado pela análise teórica, a estrutura regular destes grafos impôs dificuldades

significativas ao método exato (PI). O solver foi capaz de comprovar a otimalidade apenas para as instâncias pequenas (até 100 vértices); a partir de $|V| \geq 150$, o limite de tempo de 900 segundos foi sistematicamente atingido em todas as execuções, fazendo com que o modelo retornasse apenas a melhor solução viável encontrada até a interrupção.

O maior gap relativo observado em todos os experimentos foi de 19,60%, atingido pelo algoritmo BRKGA. Esse resultado ocorreu na instância *cubic_850*, que possui dimensão de 850 vértices. Em comparação, o maior gap do GA foi significativamente menor, com 15,42% na instância de 176 vértices, mostrando a dificuldade crescente do BRKGA nas maiores instâncias desta classe.

Tabela 5 – Resultados detalhados para as instâncias *Random Cubic*. O asterisco (*) na coluna de Solução do PI indica que a otimalidade foi comprovada pelo método exato.

Instância	$\|V\|$	$\|E\|$	**PI (Exato)**		**BRKGA**				**Algoritmo Genético (GA)**			
			Sol.	**T(s)**	**Best**	**Worst**	**Gap(%)**	**T(s)**	**Best**	**Worst**	**Gap(%)**	**T(s)**
cubic_50	50	75	34*	14.12	35	39	8.82	11.11	38	39	12.79	5.98
cubic_76	76	114	52*	36.57	55	60	9.42	14.41	58	59	12.88	7.61
cubic_100	100	150	68*	83.37	73	79	12.13	21.99	77	79	14.85	8.98
cubic_150	150	225	102	900.01	113	120	13.25	31.25	117	118	14.17	12.68
cubic_176	176	264	120	900.07	133	143	14.79	35.73	137	139	15.42	19.88
cubic_200	200	300	139	900.02	155	166	14.17	38.61	156	158	13.38	20.28
cubic_250	250	375	178	900.02	197	204	12.87	44.68	196	199	10.87	24.40
cubic_276	276	414	194	900.02	219	229	15.10	46.71	217	220	12.68	26.81
cubic_300	300	450	209	900.02	238	252	17.03	48.67	234	239	13.78	29.37
cubic_350	350	525	249	900.02	281	295	15.32	62.27	276	279	11.55	34.33
cubic_376	376	564	271	900.02	305	317	14.65	85.40	296	300	10.17	36.87
cubic_400	400	600	297	900.02	322	339	10.84	96.98	316	319	7.02	39.26
cubic_450	450	675	336	900.05	365	382	11.13	96.26	355	360	6.50	43.59
cubic_476	476	714	358	900.03	386	413	11.34	85.54	378	381	6.02	44.32
cubic_500	500	750	374	900.03	412	431	12.58	81.59	397	400	6.59	48.23
cubic_550	550	825	422	900.04	459	475	10.53	97.70	436	441	3.99	50.63
cubic_576	576	864	434	900.02	476	497	12.42	99.18	458	461	5.99	52.09
cubic_600	600	900	455	900.03	499	521	12.15	92.32	476	480	5.25	53.57
cubic_650	650	975	491	900.04	549	567	13.44	102.16	516	522	5.75	61.45
cubic_676	676	1014	511	900.49	565	594	13.73	105.32	537	542	5.71	63.74
cubic_700	700	1050	525	900.05	593	611	14.61	108.95	557	562	6.65	66.94
cubic_750	750	1125	565	900.19	637	664	14.87	114.02	599	602	6.35	71.78
cubic_776	776	1164	581	900.11	662	688	15.40	122.22	618	623	6.92	76.84
cubic_800	800	1200	593	900.07	678	705	16.91	125.41	636	643	7.94	78.12
cubic_850	850	1275	619	900.06	725	755	19.60	129.89	679	683	10.07	83.08
cubic_876	876	1314	644	900.06	751	778	18.69	133.97	699	703	8.96	84.97

7.5.1 Análise Comparativa: Wireless Networks

A Tabela 6 apresenta os resultados obtidos para a classe *Wireless Networks*, confrontando o desempenho do modelo exato (PI) e das meta-heurísticas propostas (BRKGA e GA) com os resultados do algoritmo VNS reportados no artigo de (Djukanović *et al.*, 2025). Para as

instâncias de pequeno porte (20 e 30 vértices), observa-se uma boa convergência, onde todas as abordagens foram capazes de alcançar as soluções ótimas em tempos computacionais reduzidos.

À medida que o tamanho e a densidade dos grafos aumentam, destaca-se a eficiência do modelo matemático proposto, que superou o VNS em tempo de execução na maioria dos cenários comparáveis, especialmente nas instâncias de ordem maior ou igual a 100. Um resultado notável ocorre na instância de 100 vértices com densidade 0.6: enquanto o VNS reportou uma solução de valor 7, tanto o modelo exato quanto as heurísticas propostas (BRKGA e GA) foram capazes de encontrar uma solução de melhor qualidade (valor 6). É importante ressaltar, ainda, que os testes do VNS foram limitados apenas para essas bases calculadas, com $k = 2$.

É importante notar que o BRKGA encontrou o ótimo em quase todas as instâncias dessa base, com exceção apenas do grafo `ad-hoc-net_150_0.3`. Apesar de tanto o PI quanto o BRKGA encontrarem a solução ótima nessas instâncias, o BRKGA teve uma pior performance de tempo, quando comparado ao PI. Possivelmente, isso se deu pelo fato do BRKGA trabalhar com uma quantidade fixa de gerações máximas. É provável que o BRKGA encontre a solução ótima logo nas primeiras gerações, mas a quantidade fixa de gerações máximas o obrigue a ir até o final da execução. Comportamento semelhante pode ser observado com relação ao GA. É possível que a ordem pequena dessas instâncias dê uma maior vantagem ao PI.

Tabela 6 – Comparação de abordagens na classe *Wireless Networks*. Os dados do algoritmo VNS foram extraídos da literatura de referência para as instâncias compatíveis ($k = 2$).

Instância	$\|V\|$	$\|E\|$	**PI**		**VNS (Lit.)**		**BRKGA**			**GA**		
			Sol.	**T(s)**	**Sol.**	**T(s)**	**Best**	**Gap(%)**	**T(s)**	**Best**	**Gap(%)**	**T(s)**
ad-hoc-net_20_0.3	20	35	**10***	0.15	12	1.1	**10**	0.00	5.85	**10**	0.00	4.92
ad-hoc-net_20_0.4	20	56	**9***	0.17	9	0.3	**9**	0.00	7.00	**9**	0.00	5.54
ad-hoc-net_20_0.5	20	124	**5***	0.10	5	0.2	**5**	0.00	9.64	**5**	0.00	7.32
ad-hoc-net_30_0.3	30	95	**13***	0.17	-	-	**13**	0.00	9.75	**13**	0.00	7.07
ad-hoc-net_30_0.4	30	185	**7***	0.35	-	-	**7**	0.00	13.38	**7**	0.00	9.61
ad-hoc-net_30_0.5	30	179	**8***	0.16	-	-	**8**	0.00	13.32	9	12.50	9.47
ad-hoc-net_50_0.3	50	304	**13***	1.08	13	3.8	**13**	0.00	21.84	14	7.69	13.88
ad-hoc-net_50_0.6	50	746	**5***	0.33	5	2.3	**5**	0.00	38.74	**5**	0.00	25.35
ad-hoc-net_100_0.3	100	1114	**13***	6.86	13	237.8	**13**	0.00	66.61	17	30.77	38.14
ad-hoc-net_100_0.4	100	1885	**10***	14.29	10	25.8	**10**	0.00	109.50	**10**	0.00	56.50
ad-hoc-net_100_0.5	100	2505	**9***	20.53	9	0.0	**9**	0.00	144.61	**9**	0.00	73.21
ad-hoc-net_100_0.6	100	3129	**6***	2.53	7	21.5	**6**	0.00	148.86	**6**	0.00	91.88
ad-hoc-net_150_0.3	150	2313	**16***	58.24	-	-	18	12.50	121.79	20	25.00	71.43
ad-hoc-net_150_0.4	150	3960	**12***	410.30	-	-	**12**	0.00	209.68	**12**	0.00	114.75
ad-hoc-net_150_0.5	150	5062	**9***	135.36	-	-	**9**	0.00	252.40	**9**	0.00	147.07
ad-hoc-net_150_0.6	150	7302	**6***	7.53	-	-	**6**	0.00	365.06	**6**	0.00	208.91

*Solução ótima provada pelo método exato (PI).

Valores em negrito indicam que a heurística alcançou a solução ótima.

7.5.2 Análise Comparativa: Random Graphs

A Tabela 7 resume os experimentos realizados na classe *Random Graphs* comparando o desempenho do modelo exato proposto e das meta-heurísticas com os resultados de referência do algoritmo VNS. Para as instâncias de pequeno e médio porte (até 50 vértices), todos os métodos demonstraram alta eficácia, convergindo rapidamente para as soluções ótimas.

A superioridade do modelo matemático exato torna-se evidente nas instâncias de maior dimensão ($|V| \geq 100$). Nestes cenários, o modelo proposto não apenas encontrou soluções de qualidade igual ou superior às reportadas pelo VNS, como o fez em uma fração do tempo computacional. Por exemplo, nas instâncias de 100 vértices, o tempo médio do modelo exato variou entre 30 e 45 segundos, enquanto o VNS da literatura necessitou de tempos superiores a 400 segundos em diversos casos, chegando a entregar soluções subótimas (e.g., instância 100_1 e 100_5) comparadas ao ótimo comprovado pelo nosso método.

Tabela 7 – Comparação de abordagens na classe *Random Graphs*. As colunas do algoritmo VNS estão prenchidas com os testes que estão no artigo

Instância ($\|V\|$_ID e densidade)	$\|E\|$	**PI**		**VNS (Lit.)**		**BRKGA**			**GA**		
		Sol.	**T(s)**	**Sol.**	**T(s)**	**Best**	**Gap(%)**	**T(s)**	**Best**	**Gap(%)**	**T(s)**
10_1 (0.44)	20	**6***	0.03	-	-	**6**	0.00	4.63	**6**	0.00	4.07
10_2 (0.44)	20	**6***	0.06	-	-	**6**	0.00	4.32	**6**	0.00	4.07
10_3 (0.44)	20	**6***	0.03	-	-	**6**	0.00	4.40	**6**	0.00	4.05
10_4 (0.44)	20	**5***	0.01	-	-	**5**	0.00	4.34	**5**	0.00	4.09
10_5 (0.44)	20	**6***	0.03	-	-	**6**	0.00	4.39	**6**	0.00	4.09
15_1 (0.29)	30	**9***	0.18	-	-	**9**	0.56	5.59	**9**	2.22	4.59
15_2 (0.29)	30	**8***	0.15	-	-	**8**	1.25	5.45	**8**	0.00	4.58
15_3 (0.29)	30	**8***	0.03	-	-	**8**	0.00	5.45	**8**	0.00	4.66
15_4 (0.29)	30	**9***	0.14	-	-	**9**	2.78	5.58	**9**	10.56	4.54
15_5 (0.29)	30	**9***	0.15	-	-	**9**	0.00	5.58	**9**	0.00	4.54
20_1 (0.21)	40	**11***	0.15	-	-	**11**	0.00	6.39	**11**	0.00	5.09
20_2 (0.21)	40	**12***	0.16	-	-	**12**	0.00	6.41	**12**	0.00	5.06
20_3 (0.21)	40	**11***	0.14	-	-	**11**	2.27	6.49	**11**	0.00	5.07
20_4 (0.21)	40	**11***	0.07	-	-	**11**	0.45	6.51	**11**	0.00	5.18
20_5 (0.21)	40	**12***	0.17	-	-	**12**	0.00	6.46	**12**	0.00	5.04
30_1 (0.14)	60	**17***	0.42	**17**	2.0	**17**	4.71	8.35	**17**	0.00	6.03
30_2 (0.14)	60	**17***	0.68	**17**	16.1	**17**	3.53	8.35	**17**	0.00	5.99
30_3 (0.14)	60	**17***	1.12	**17**	2.2	**17**	7.94	8.43	18	9.71	6.03
30_4 (0.14)	60	**16***	0.22	**16**	0.1	**16**	1.88	8.31	**16**	0.00	6.07
30_5 (0.14)	60	**17***	0.94	**17**	12.63	**17**	6.18	8.26	18	5.88	6.04
45_1 (0.09)	90	**26***	1.03	-	-	27	5.19	10.93	27	3.85	7.42
45_2 (0.09)	90	**26***	1.97	-	-	27	6.35	10.99	28	7.69	7.43
45_3 (0.09)	90	**26***	1.41	-	-	**26**	6.73	12.20	27	3.85	7.50
45_4 (0.09)	90	**27***	2.49	-	-	**27**	3.15	11.93	**27**	0.93	7.45
45_5 (0.09)	90	**26***	1.68	-	-	**26**	4.04	13.11	27	3.85	7.54
50_1 (0.08)	100	**29***	2.89	**29**	15.0	30	6.55	13.58	30	3.45	7.88
50_2 (0.08)	100	**28***	1.51	**28**	7.0	**28**	1.96	13.58	29	3.57	7.95
50_3 (0.08)	100	**28***	2.12	**28**	116.1	**28**	7.68	13.67	30	7.14	7.96
50_4 (0.08)	100	**28***	2.23	**28**	5.6	**28**	6.43	12.81	**28**	5.54	7.89
50_5 (0.08)	100	**29***	2.79	**29**	137.8	30	8.45	12.23	30	3.45	7.89
100_1 (0.04)	200	**57***	36.43	58	505.1	60	12.37	20.23	61	7.11	12.69
100_2 (0.04)	200	**58***	42.92	61	461.2	62	11.90	20.21	61	6.81	12.60
100_3 (0.04)	200	**60***	37.70	62	460.0	63	10.92	20.24	63	6.50	12.67
100_4 (0.04)	200	**58***	35.19	60	492.1	60	8.28	20.28	61	5.17	12.67
100_5 (0.04)	200	**57***	33.23	59	189.6	60	9.91	20.26	60	5.26	12.74

7.5.3 Análise dos Resultados na Coleção Harwell-Boeing

A Tabela 8 apresenta o desempenho das abordagens na coleção *Harwell-Boeing*. Observa-se que, para as instâncias de menor porte (até aproximadamente 100 vértices), o modelo exato (PI) é capaz de provar a otimalidade em tempos curtos, com as heurísticas BRKGA e GA

acompanhando esses resultados na maioria dos casos, encontrando o ótimo ou soluções muito próximas.

À medida que o número de vértices e arestas aumenta, o método exato atinge consistentemente o tempo limite de 900 segundos, retornando apenas a melhor solução viável encontrada até o momento, sem garantia de otimalidade. É neste cenário que o algoritmo GA se destaca notavelmente. Em diversas instâncias de médio e grande porte, como *dwt_310*, *bcsstm07* e *bcspwr10*, o GA obteve *gaps* negativos significativos (chegando a -16,32%), o que indica que a heurística foi capaz de encontrar soluções de qualidade superior às encontradas pelo solver comercial dentro do tempo estipulado.

Em contrapartida, o BRKGA apresentou uma dificuldade maior em convergir para soluções competitivas nesta classe de instâncias, exibindo *gaps* positivos elevados em casos onde o GA e o PI obtiveram melhor desempenho. Isso sugere que, para as estruturas específicas presentes na coleção *Harwell-Boeing*, os operadores genéticos clássicos do GA mostraram-se mais robustos.

Tabela 8 – Resultados detalhados para as instâncias da coleção *Harwell-Boeing*. O Gap é relativo à melhor solução encontrada pela heurística em comparação ao método exato (PI).

Instância	$\|V\|$	$\|E\|$	**PI (Viável)**		**BRKGA**			**GA**		
			Sol.	**T(s)**	**Best**	**Gap(%)**	**T(s)**	**Best**	**Gap(%)**	**T(s)**
bcspwr02	49	59	**35***	0.10	**35**	0.00	9.86	**35**	0.00	5.11
can_61	61	248	**17***	0.40	**17**	0.00	19.35	20	17.65	13.15
dwt_66	66	127	**38***	33.39	42	7.69	14.77	39	0.00	11.36
bcsstk02	66	2145	**3***	0.62	**3**	0.00	101.48	**3**	0.00	70.57
ash85	85	219	**41***	351.98	44	7.32	19.56	43	4.88	9.78
can_96	96	336	**35***	95.22	36	2.86	26.74	**35**	0.00	17.56
bcspwr03	118	179	**74***	1.97	76	2.70	20.24	77	4.05	10.23
can_161	161	608	**57**	900.17*	64	12.28	44.27	**57**	0.00	30.51
dwt_193	193	1650	**36**	900.02*	41	13.89	93.39	40	11.11	80.37
dwt_198	198	597	**79***	13.02	98	24.05	46.23	85	7.59	41.74
dwt_310	310	1069	144	900.03*	151	4.86	74.84	**129**	-10.42	64.15
dwt_361	361	1296	170	900.06*	178	4.71	88.83	**150**	-11.76	74.79
plat362	362	2712	95	900.05*	113	18.95	157.19	**85**	-10.53	107.03
bfw398b	398	1256	**209**	900.02*	234	11.96	92.48	224	7.18	78.31
bcsstm07	420	3416	131	900.05*	141	7.63	184.41	**112**	-14.50	146.39
bcsstk20	485	1325	**249**	900.03*	300	20.48	100.91	263	5.62	82.49
494_bus	494	586	**347***	11.14	401	15.56	68.06	365	5.19	40.51
dwt_512	511	1495	**264**	900.03*	323	21.89	112.24	277	4.53	90.46
can_634	634	3297	204	900.07*	276	35.29	209.20	**174**	-14.71	164.71
nos6	675	1290	470	900.05*	521	10.85	113.93	**467**	-0.64	86.26
bcsstk19	817	3018	387	900.05*	469	21.19	203.60	**351**	-9.30	156.16
young1c	841	1624	584	900.07*	661	13.18	139.44	**580**	-0.68	98.97
bcsstk08	1071	5943	**244**	900.14*	475	94.67	368.39	335	37.30	307.38
bcsstk09	1083	8677	**275**	900.07*	498	81.09	483.78	296	7.64	367.01
1138_bus	1138	1458	**762***	72.15	970	27.30	155.06	796	4.46	98.34
bcspwr06	1454	1923	**961**	900.05*	1273	32.47	247.02	1016	5.72	133.08
bcspwr08	1624	2213	**1057**	900.06*	1427	35.00	248.08	1117	5.68	153.46
bcspwr09	1723	2394	**1115**	900.07*	1527	36.95	285.87	1166	4.57	244.26
add20	2395	7462	**872***	9.53	1899	117.78	536.20	1673	91.86	354.92
bcspwr10	5300	8271	4234	900.15*	5545	30.96	1080.41	**3543**	-16.32	1105.82

*O tempo de 900s indica que o solver atingiu o limite de tempo (Time Limit).

Gaps negativos indicam que a heurística encontrou solução melhor que a viável do solver.

* na solução do PI indica resultado ótimo

7.5.4 Análise dos Resultados na Coleção Geoinstances

A Tabela 9 detalha os resultados obtidos para a classe *Geoinstances*, composta por grafos derivados de redes reais (cidades e regiões geográficas). Esta coleção apresenta uma topologia distinta das classes aleatórias, refletindo estruturas encontradas em aplicações práticas de redes.

O método exato (PI) demonstrou bom desempenho nesta classe, sendo capaz de provar a otimalidade em 81 das 96 instâncias testadas (aproximadamente 84% do conjunto). No entanto, para as instâncias de maior porte (com mais de 500 vértices), o solver frequentemente atingiu o limite de tempo de 900 segundos, retornando a melhor solução viável encontrada.

Comparando as meta-heurísticas, o Algoritmo Genético (GA) mostrou-se consis-

tentemente superior ao BRKGA. O GA obteve um *gap* médio de 2,36% em relação ao método exato, enquanto o BRKGA apresentou um *gap* médio de 3,78%. Além disso, o GA encontrou a solução ótima (ou a melhor conhecida) em um número maior de casos, confirmando sua robustez também em grafos baseados em redes reais.

É válido mencionar os experimentos realizados por (Djukanović *et al.*, 2025) utilizando a meta-heurística VNS (*Variable Neighborhood Search*) no conjunto de instâncias "Real", que possui características similares a esta coleção. Naquele estudo, para o caso $k = 2$, o algoritmo VNS obteve uma média de solução de 20,18 contra uma média ótima de 20,11 (proveniente do método exato), resultando em um *gap* extremamente reduzido de aproximadamente 0,35%. Comparativamente, a abordagem gulosa reportada pelos mesmos autores obteve uma média de 20,98 (gap de $\approx$ 4,3%).

Isso posiciona o GA desenvolvido neste trabalho (gap de 2,36%) como uma abordagem competitiva, superando a estratégia construtiva gulosa da literatura e aproximando-se dos resultados de métodos de busca local intensiva como o VNS.

Tabela 9 – Resultados detalhados para as instâncias da coleção *Geoinstances*. O símbolo * indica otimalidade provada pelo PI.

Instância	$\|V\|$	$\|E\|$	**PI**		**BRKGA**			**GA**		
			Sol.	**T(s)**	**Best**	**Gap(%)**	**T(s)**	**Best**	**Gap(%)**	**T(s)**
new-york-city-boroughs	4	5	**3***	0.01	**3**	0.00	3.34	**3**	0.00	2.35
oman	7	11	**4***	0.01	**4**	0.00	3.53	**4**	0.00	2.62
surrey	7	11	**4***	0.02	**4**	0.00	3.75	**4**	0.00	2.96
winterthur	7	10	**4***	0.01	**4**	0.00	4.70	**4**	0.00	2.47
atlanta	8	4	**8***	0.01	**8**	0.00	3.18	**8**	0.00	2.47
honolulu	8	9	**5***	0.01	**5**	0.00	3.38	**5**	0.00	3.13
chesapeake	9	16	**5***	0.01	**5**	0.00	3.98	**5**	0.00	3.09
utrecht	10	19	**6***	0.07	**6**	0.00	7.36	**6**	0.00	3.08
kansas-city	10	22	**5***	0.06	**5**	0.00	4.90	**5**	0.00	2.67
st-petersburg	10	19	**5***	0.06	**5**	0.00	3.79	**5**	0.00	2.50
williamsburg	10	16	**6***	0.02	**6**	0.00	4.10	**6**	0.00	2.58
pittsburgh	11	25	**5***	0.03	**5**	0.00	3.84	**5**	0.00	3.12

Continua na próxima página...

Tabela 9 – Resultados detalhados para as instâncias da coleção *Geoinstances* (Continuação).

Instância	$\|V\|$	$\|E\|$	**PI**		**BRKGA**			**GA**		
			Sol.	**T(s)**	**Best**	**Gap(%)**	**T(s)**	**Best**	**Gap(%)**	**T(s)**
canberra	12	30	**6***	0.03	**6**	0.00	3.86	**6**	0.00	2.72
philadelphia	12	24	**6***	0.05	**6**	0.00	3.99	**6**	0.00	2.51
orlando	13	36	**6***	0.05	**6**	0.00	5.10	**6**	0.00	3.19
des-moines	13	35	**6***	0.04	**6**	0.00	4.17	**6**	0.00	2.74
venice	13	18	**8***	0.02	**8**	0.00	3.96	**8**	0.00	2.61
richmond	14	37	**7***	0.07	**7**	0.00	4.50	**7**	0.00	2.82
manhattan	14	36	**7***	0.06	**7**	0.00	4.88	**7**	0.00	2.68
madrid	14	40	**7***	0.05	**7**	0.00	5.12	**7**	0.00	2.85
red-deer	16	37	**7***	0.06	**7**	0.00	4.13	**7**	0.00	3.15
san-francisco	16	48	**6***	0.07	**6**	0.00	4.16	**6**	0.00	3.14
hamburg	16	55	**7***	0.07	**7**	0.00	4.35	**7**	0.00	3.20
saskatoon	16	43	**7***	0.09	**7**	0.00	5.17	**7**	0.00	3.13
geneva	16	31	**9***	0.04	**9**	0.00	4.49	**9**	0.00	2.58
milwaukee	17	48	**7***	0.10	**7**	0.00	5.06	**7**	0.00	2.76
chapel-hill	18	46	**9***	0.09	**9**	0.00	6.20	**9**	0.00	3.17
st-louis	18	58	**8***	0.09	**8**	0.00	4.54	**8**	0.00	2.94
birmingham	19	56	**9***	0.11	**9**	0.00	5.46	**9**	0.00	2.72
liberia-east	19	30	**9***	0.04	**9**	0.00	4.29	**9**	0.00	2.76
minneapolis	19	56	**8***	0.11	**8**	0.00	4.70	**8**	0.00	3.13
london-underground	20	42	**11***	0.05	**11**	0.00	4.54	**11**	0.00	2.58
baltimore	20	60	**9***	0.13	**9**	0.00	4.88	**9**	0.00	3.03
brooklyn	20	59	**9***	0.12	**9**	0.00	4.96	**9**	0.00	3.26
cologne	20	68	**9***	0.20	**9**	0.00	5.76	**9**	0.00	3.01
italy-regions	20	33	**11***	0.05	**11**	0.00	4.41	**11**	0.00	2.89
austin	21	74	**9***	0.18	**9**	0.00	5.37	**9**	0.00	3.52
ulm	21	56	**9***	0.18	**9**	0.00	5.23	**9**	0.00	3.06
dresden	21	70	**9***	0.16	**9**	0.00	5.09	**9**	0.00	3.30

Continua na próxima página...

Tabela 9 – Resultados detalhados para as instâncias da coleção *Geoinstances* (Continuação).

Instância	$\|V\|$	$\|E\|$	**PI**		**BRKGA**			**GA**		
			Sol.	**T(s)**	**Best**	**Gap(%)**	**T(s)**	**Best**	**Gap(%)**	**T(s)**
rockville	22	64	**10***	0.19	**10**	0.00	6.00	**10**	0.00	3.48
seattle	22	71	**9***	0.24	**9**	0.00	6.22	**9**	0.00	3.65
charlottesville	22	66	**10***	0.23	**10**	0.00	5.17	**10**	0.00	3.19
eindhoven	22	66	**10***	0.22	**10**	0.00	5.01	**10**	0.00	3.09
bronx	23	73	**10***	0.23	**10**	0.00	6.09	**10**	0.00	3.44
calgary	23	74	**10***	0.38	**10**	0.00	7.72	**10**	0.00	3.36
los-angeles	24	84	**10***	0.23	**10**	0.00	6.27	**10**	0.00	3.23
berlin	25	82	**11***	0.40	**11**	0.00	7.23	**11**	0.00	3.57
edmonton	25	79	**11***	0.30	**11**	0.00	5.75	**11**	0.00	3.25
moscow	25	72	**11***	0.25	**11**	0.00	6.13	**11**	0.00	3.48
san-diego	25	84	**10***	0.26	**10**	0.00	5.91	**10**	0.00	3.49
queens	26	78	**11***	0.46	**11**	0.00	6.18	**11**	0.00	3.41
houston	26	86	**10***	0.41	**10**	0.00	6.64	**10**	0.00	3.29
denver	26	94	**11***	0.45	**11**	0.00	6.55	**11**	0.00	3.84
enschede	27	100	**11***	0.46	**11**	0.00	6.81	**11**	0.00	4.14
macon	28	80	**14***	0.36	**14**	0.00	7.14	**14**	0.00	3.48
mississauga	29	97	**12***	0.69	**12**	0.00	7.42	**12**	0.00	4.09
west-linn	32	90	**15***	0.61	**15**	0.00	6.88	**15**	0.00	3.84
milan	32	107	**14***	0.81	**14**	0.00	8.92	**14**	0.00	3.84
san-antonio	33	106	**14***	1.09	**14**	0.00	8.44	**14**	0.00	4.29
toronto	33	110	**14***	0.96	**14**	0.00	8.41	**14**	0.00	4.28
norfolk	39	116	**18***	2.15	**18**	0.00	10.99	**18**	0.00	5.17
hartford	46	130	**22***	3.10	24	9.09	14.15	**22**	0.00	5.86
oakland	47	142	**20***	4.28	22	10.00	15.65	**20**	0.00	6.29
le-mans-cantons	47	107	**24***	0.82	**24**	0.00	13.91	**24**	0.00	5.92
portland	47	154	**19***	4.54	21	10.53	15.83	20	5.26	6.91
providence	47	138	**21***	4.88	22	4.76	14.39	**21**	0.00	5.67

Continua na próxima página...

Tabela 9 – Resultados detalhados para as instâncias da coleção *Geoinstances* (Continuação).

Instância	$\|V\|$	$\|E\|$	**PI**		**BRKGA**			**GA**		
			Sol.	**T(s)**	**Best**	**Gap(%)**	**T(s)**	**Best**	**Gap(%)**	**T(s)**
raleigh	49	150	**22***	4.67	23	4.55	14.73	**22**	0.00	6.84
tampa	50	150	**22***	6.27	23	4.55	15.67	**22**	0.00	6.66
henderson	51	156	**22***	8.78	23	4.55	16.79	**22**	0.00	7.15
united-states	51	105	**26***	0.94	**26**	0.00	13.06	**26**	0.00	5.88
minneapolis-cities	53	145	**25***	5.41	**25**	0.00	15.35	**25**	0.00	5.98
louisville	54	166	**23***	16.19	26	13.04	18.23	**23**	0.00	7.82
windsor	54	164	**24***	6.89	26	8.33	17.65	**24**	0.00	7.13
antwerp	61	155	**30***	7.24	**30**	0.00	20.25	**30**	0.00	9.49
liberia-central	65	133	**31***	6.13	34	9.68	20.89	32	3.23	10.32
olympia	67	206	**29***	30.64	33	13.79	22.01	30	3.45	11.23
apulia	75	183	**36***	32.53	39	8.33	29.35	37	2.78	14.39
mexico	82	234	**41***	512.91	42	2.44	30.13	**41**	0.00	14.28
united-states-international	82	200	**41***	63.81	43	4.88	31.75	**41**	0.00	14.73
los-angeles-county	88	216	**42***	61.27	45	7.14	36.31	43	2.38	16.14
nordrhein-westfalen	95	232	**47***	731.83	51	8.51	42.14	**47**	0.00	21.03
new-orleans	101	284	**45***	506.77	49	8.89	44.75	46	2.22	22.25
quebec	103	246	**51**	900.02*	58	13.73	51.35	53	3.92	22.75
james-city-county	106	260	**52***	412.33	58	11.54	54.49	**52**	0.00	24.36
italy-provinces	110	260	**55**	900.02*	59	7.27	51.37	56	1.82	23.32
rio-de-janeiro	123	325	**57**	900.01*	65	14.04	68.61	59	3.51	33.15
taiwan	155	439	**69**	900.02*	76	10.14	92.51	72	4.35	43.62
united-kingdom	169	470	**76**	900.02*	84	10.53	108.99	81	6.58	52.02
japan	176	503	**78**	900.01*	88	12.82	118.82	83	6.41	56.40
russia	183	556	**79**	900.02*	94	18.99	134.40	84	6.33	67.57
new-york-city	233	676	**99**	900.02*	113	14.14	195.42	105	6.06	94.61
turkey	241	740	**95**	900.03*	115	21.05	201.20	104	9.47	92.93
china	552	1765	196	900.03*	257	31.12	665.34	**193**	-1.53	345.54

Continua na próxima página...

Tabela 9 – Resultados detalhados para as instâncias da coleção *Geoinstances* (Continuação).

Instância	$\|V\|$	$\|E\|$	**PI**		**BRKGA**			**GA**		
			Sol.	**T(s)**	**Best**	**Gap(%)**	**T(s)**	**Best**	**Gap(%)**	**T(s)**
brazil	593	1918	210	900.04*	280	33.33	805.81	**203**	-3.33	453.64
india	657	2176	230	900.04*	317	37.83	1026.06	**219**	-4.78	548.81

7.6 Análise Geral dos Resultados Experimentais

Nesta seção, apresentamos um resumo do desempenho das abordagens propostas: Modelo Exato (PI), BRKGA e Algoritmo Genético (GA), considerando todas as classes de instâncias avaliadas. A Tabela 10 sumariza as médias de tempo de execução e os *gaps* percentuais médios obtidos por cada método em relação à melhor solução conhecida.

Observa-se que o modelo exato apresenta excelente desempenho em instâncias de pequeno porte (como em *Random Graphs* e instâncias iniciais de *Wireless*), com tempos médios reduzidos. Contudo, em classes mais complexas ou densas, como *Random Cubic* e *Harwell-Boeing*, o tempo médio do PI eleva-se drasticamente, evidenciando a escalabilidade limitada.

Entre as meta-heurísticas, nota-se um comportamento distinto que depende da forma do grafo. O BRKGA obteve gaps médios muito baixos para *Wireless Networks* (0,78%) e *Random Graphs* (1,16%), superando o GA nestes cenários específicos. Por outro lado, o GA demonstrou maior robustez e generalização em instâncias estruturadas e maiores, como *Harwell-Boeing* e *Geoinstances*, onde obteve gaps médios significativamente menores que o BRKGA (4,54% vs 23,79% em Harwell-Boeing), além de manter tempos de execução consistentemente inferiores.

Esses resultados sugerem que o BRKGA desenvolvido tende a ser melhor em instâncias de menor porte (ordem e densidade), enquanto o GA tende a ser melhor em instâncias tanto de pqeueno quanto de maior porte. No geral, nenhum dos métodos conseguiu superar o outro em todas as bases de grafos. Porém, pelo cálculo dos GAPs médios, sugere-se uma maior superioridade do GA em conseguir obter soluções de melhor qualidade em menor tempo que o BRKGA na maioria das bases testadas.

Tabela 10 – Resumo geral dos resultados médios por classe de instância. Os tempos estão em segundos e os gaps em porcentagem relativos à solução do PI.

Classe de Instância	**Qtd.**	**PI**	**BRKGA**		**GA**	
		Tempo Médio (s)	**Tempo Médio (s)**	**Gap Médio (%)**	**Tempo Médio (s)**	**Gap Médio (%)**
Random Graphs	35	6.01	10.00	1.16	6.83	2.18
Wireless Networks	16	41.13	96.13	0.78	55.32	4.75
Geoinstances	137	196.17	23.09	3.97	11.33	2.39
Harwell-Boeing	30	589.69	178.19	23.79	141.50	4.54
Random Cubic	26	801.37	78.17	10.93	44.07	8.54

8 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho abordou o Problema de Dominação Romana 2-Forte, um problema de otimização combinatória classificado como NP-difícil, por meio da implementação e análise comparativa de duas abordagens evolutivas: um Algoritmo Genético (GA) e um Algoritmo Genético de Chaves Aleatórias Enviesadas (BRKGA). A eficácia das heurísticas propostas foi avaliada frente a um método exato (Programação Linear Inteira - PLI) em conjuntos de instâncias diversificados, abrangendo grafos aleatórios e a coleção *Harwell-Boeing*.

A análise experimental revelou uma diferença clara no desempenho das abordagens condicionada às características das instâncias. Para grafos de menor porte, o modelo exato (PI) demonstrou superioridade, garantindo a otimalidade das soluções com tempos computacionais reduzidos, superando frequentemente ambas as heurísticas tanto em qualidade quanto em velocidade. Entretanto, observou-se uma degradação acentuada na performance do método exato à medida que o número de vértices aumenta e a densidade diminui, cenário que se tornou evidente nos experimentos com grafos cúbicos e esparsos. Nestas situações, o Algoritmo Genético (GA) apresentou um comportamento mais robusto e satisfatório, sendo capaz de fornecer soluções de melhor qualidade em tempos inferiores aos exigidos pelo solver comercial para provar a otimalidade ou mesmo encontrar uma solução viável. Na coleção *Harwell-Boeing*, por exemplo, o GA destacou-se ao encontrar soluções melhores que as viáveis fornecidas pelo solver em instâncias de grande porte, onde o método exato frequentemente atingiu o tempo limite.

Como trabalhos futuros, recomenda-se a expansão dos conjuntos de teste para incluir grafos de grande porte e com variações de densidade mais acentuadas, como as instâncias da coleção DIMACS, a fim de validar a escalabilidade das heurísticas em cenários onde o método exato se torna inviável. Sugere-se também o aprimoramento contínuo dos algoritmos, investigando novos operadores genéticos e ajustes finos que possam elevar a qualidade das soluções. Por fim, propõe-se a extensão direta deste estudo adaptando as abordagens desenvolvidas (GA e BRKGA) para lidar com a generalização do problema, a Dominação Romana k-Forte para um k genérico ($k > 2$), avaliando se o desempenho comparativo observado neste trabalho se mantém para valores superiores de k.

REFERÊNCIAS

Abdollahzadeh Ahangar, H.; ÁLVAREZ, M.; CHELLALI, M.; SHEIKHOLESLAMI, S.; VALENZUELA-TRIPODORO, J. Triple roman domination in graphs. **Applied Mathematics and Computation**, v. 391, p. 125444, 2021. ISSN 0096-3003. Disponível em: https://www.sciencedirect.com/science/article/pii/S0096300320304057.

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. **ORSA Journal on Computing**, INFORMS, v. 6, n. 2, p. 154–160, 1994.

BEELER, R. A.; HAYNES, T. W.; HEDETNIEMI, S. T. Double roman domination. **Discrete Applied Mathematics**, v. 211, p. 23–29, 2016. ISSN 0166-218X. Disponível em: https://www.sciencedirect.com/science/article/pii/S0166218X1630155X.

COCKAYNE, E. J.; JR, P. A. D.; HEDETNIEMI, S. M.; HEDETNIEMI, S. T. Roman domination in graphs. **Discrete Mathematics**, Elsevier, v. 278, n. 1-3, p. 11–22, 2004.

DARWIN, C. **On the Origin of Species**. London: John Murray, 1859.

DIESTEL, R. **Graph Theory**. 5. ed. Springer, 2017. v. 173. (Graduate Texts in Mathematics, v. 173). ISBN 978-3-662-53621-6. Disponível em: https://diestel-graph-theory.com.

DJUKANOVIć, M.; KAPUNAC, S.; KARTELJ, A.; MATIć, D. Graph protection under multiple simultaneous attacks: A heuristic approach. **Knowledge-Based Systems**, v. 309, p. 112791, 2025. ISSN 0950-7051. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950705124014254.

DUFF, I. S.; GRIMES, R. G.; LEWIS, J. G. Sparse matrix test problems. **ACM Transactions on Mathematical Software (TOMS)**, ACM New York, NY, USA, v. 15, n. 1, p. 1–14, 1989.

Genially. **Mapa do Império Romano**. 2023. https://view.genially.com/63ecbf2101aa5e0012b054e2/interactive-image-mapa-imperio-romano. Adaptado pelo autor.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Boston: Addison-Wesley, 1989. ISBN 9780201157673.

GONCALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. **Journal of Heuristics**, v. 17, n. 5, p. 487–525, 2011. ISSN 1572-9397. Disponível em: https://doi.org/10.1007/s10732-010-9143-1.

HAYNES, T. W.; HEDETNIEMI, S. T.; SLATER, P. J. **Fundamentals of Domination in Graphs**. [*S. l.*]: Marcel Dekker, 1998. v. 208. (Monographs and Textbooks in Pure and Applied Mathematics, v. 208). ISBN 978-0824700331.

HEATHER, P. **The Fall of the Roman Empire: A New History of Rome and the Barbarians**. [*S. l.*]: Oxford University Press, 2006.

HENNING, M.; KLOSTERMEYER, W. Perfect roman domination in regular graphs. **Applicable Analysis and Discrete Mathematics**, v. 12, p. 143–152, 04 2018.

KATOCH, S.; CHAUHAN, S. S.; KUMAR, V. A review on genetic algorithm: past, present, and future. **Multimedia Tools and Applications**, v. 80, n. 5, p. 8091–8126, fev. 2021.

LIU, C.-A.; WANG, Y.-C.; LIN, C.-H. The k-strong roman domination problem. **Applied Mathematics and Computation**, Elsevier, v. 380, p. 125267, 2020.

LOPEZ-IBANEZ, M.; DUBOIS-LACOSTE, J.; STÜTZLE, T.; BIRATTARI, M. irace package: Iterated racing for automatic algorithm configuration. **Operations Research Perspectives**, Elsevier, v. 3, p. 43–58, 2016.

NIKOLIć, B.; DJUKANOVIć, M.; GRBIć, M.; MATIć, D. Theoretical studies of the k-strong roman domination problem. **Kuwait Journal of Science**, v. 51, n. 4, p. 100283, 2024. ISSN 2307-4108. Disponível em: https://www.sciencedirect.com/science/article/pii/S2307410824001081.

SILVA, F.; MACIEL, P.; MAIA, C. **A biased random-key genetic algorithm for scheduling divisible loads - Scientific Figure on ResearchGate**. 2015. Available from: https://www.researchgate.net/figure/Population-evolution-between-consecutive-generations-of-a-BRKGA_fig1_281276030 [accessed 6 Jul 2025].

Stewart, I. Defend the Roman Empire! **Scientific American**, v. 281, n. 6, p. 136–138, dez. 1999.

WOLSEY, L. A. **Integer Programming**. New York: Wiley-Interscience, 1998. ISBN 9780471283669.