UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANTÔNIO ERICK FREITAS FERREIRA

METAHEURÍSTICAS PARA O PROBLEMA DA DOMINAÇÃO ROMANA PERFEITA EM GRAFOS

QUIXADÁ

2026

ANTÔNIO ERICK FREITAS FERREIRA

METAHEURÍSTICAS PARA O PROBLEMA DA DOMINAÇÃO ROMANA PERFEITA 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)

F439m Ferreira, Antônio Erick Freitas.
Metaheurísticas para o problema de dominação romana perfeita em grafos / Antônio Erick Freitas Ferreira. – 2026.
77 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. Algoritmo genético. 2. Otimização combinatória. 3. Teoria dos grafos. I. Título.
CDD 004

ANTÔNIO ERICK FREITAS FERREIRA

METAHEURÍSTICAS PARA O PROBLEMA DA DOMINAÇÃO ROMANA PERFEITA 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: 21/01/2026

BANCA EXAMINADORA

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

Prof. Dr. Lucas Ismaily Bezerra Freitas
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo Henrique Macedo de Araujo
Universidade Federal do Ceará (UFC)

Dr. Natanael Ramos
Mercado Livre, Departamento de IT Shipping
Optimization FBM & Logistics

À minha mãe, Laura Adriana, e ao meu pai, Damião Carneiro, que sempre acreditaram em mim quando nem eu mesmo acreditava. Pelo amor, pela dedicação e pelos sacrifícios feitos para que eu tivesse acesso à melhor educação possível. Tudo o que sou e conquisto carrega um pouco de vocês.

AGRADECIMENTOS

Gostaria de iniciar agradecendo aos meus pais, que sempre lutaram para que eu tivesse acesso à melhor educação possível. Durante toda a graduação, conciliei os estudos com o trabalho como Técnico em Enfermagem, em regime de escala 12/36. Em muitos momentos, assisti às aulas virado ou após poucas horas de sono e, ao longo do dia, precisava me manter firme entre provas, atividades, estudos e estágios. O que sempre me manteve determinado nessa trajetória foi o reconhecimento de todo o esforço que meus pais fizeram por mim. Pensar neles foi, muitas vezes, o combustível para seguir em frente e buscar ir cada vez mais longe por meio do estudo.

Agradeço, com muito carinho, à minha namorada Kartylla Araújo, pelo amor incondicional e por estar ao meu lado mesmo nos momentos mais difíceis dessa caminhada. Estendo também meus agradecimentos aos meus sogros, que sempre demonstraram apoio total à minha jornada acadêmica e profissional.

Deixo aqui um agradecimento especial ao meu fiel companheiro de quatro patas, Go'el, que, embora não entendesse nada do que estudei nos últimos quatro anos, nunca deixou de estar presente nas tantas madrugadas de estudo, oferecendo companhia silenciosa e conforto nos momentos de cansaço.

Agradeço profundamente ao meu orientador, Prof. Dr. Atílio Gomes Luiz, pela amizade, pelo zelo e pela paciência ao longo deste trabalho. Tudo o que você fez por mim ecoará pela minha eternidade. Agradeço também aos membros da banca examinadora, que sempre se mostraram dispostos a contribuir com esta pesquisa, em especial ao Dr. Natanael Ramos, que, mesmo não atuando como professor, sempre trouxe ideias valiosas e enriquecedoras.

Registro ainda um agradecimento especial aos Profs. Drs. Lucas Ismaily e Bruno Góes, que me ajudaram de forma ativa a conquistar minhas primeiras oportunidades de estágio, contribuindo não apenas para a minha formação acadêmica, mas também para a minha formação profissional.

Agradeço a todo o corpo docente da Universidade Federal do Ceará, com quem tive a oportunidade de criar vínculos de amizade ao longo da graduação. Todos os conselhos, ensinamentos e momentos de aprendizado certamente serão levados comigo. Agradeço também à equipe de trabalhadores que mantém o campus funcionando diariamente, sempre prestativos com os alunos e zelosos com o nosso querido espaço acadêmico.

De forma muito especial, agradeço aos meus amigos do grupo "Vamo trancar o

curso", que, apesar do nome, permaneceu firme e unido durante os quatro anos de graduação. Os momentos de resenha e estudos tornaram essa caminhada muito mais leve e significativa.

Por fim, deixo meu sincero agradecimento à Universidade Federal do Ceará, por ter transformado a minha vida por meio da educação. Que este campus continue mudando a trajetória de muitos outros jovens, assim como mudou a minha.

"A cruz não era leve mas foi carregada, pois quem tem propósito não negocia com a dor."
(Autor desconhecido)

RESUMO

Dado um grafo G, uma função $f : V(G) \to \{0,1,2\}$ é dita uma *função de dominação romana perfeita* (FDRP) de G se, para todo vértice v em G com $f(v) = 0$, existe exatamente um vértice u adjacente a v de modo que $f(u) = 2$. O peso da função f é definido como $\omega(f) = \sum_{v \in V(G)} f(v)$. O número de dominação romana perfeita, denotado por $\gamma_R^p(G)$, corresponde ao menor valor de $\omega(f)$ dentre todas as funções de dominação romana perfeitas f de G. Uma FDRP com o menor peso é dita *ótima*. O *problema de dominação romana perfeita* (PDRP) tem sido objeto de estudo nos últimos anos, revelando-se NP-completo para diversas classes de grafos. Neste trabalho de conclusão de curso, elaboramos duas metaheurísticas baseadas em *algoritmo genético* (GA): a primeira segue uma abordagem clássica, enquanto a segunda é baseada no modelo de *algoritmo genético de chaves aleatórias enviesadas* (BRKGA). Adicionalmente, foi implementado um modelo de *programação inteira* (PI) com o objetivo de fornecer uma abordagem exata para comparação com os resultados das metaheurísticas. As soluções foram obtidas para diferentes classes de grafos e comparadas através do tempo de execução e valor da função de aptidão.

Palavras-chave: algoritmo genético; otimização combinatória; teoria dos grafos.

ABSTRACT

Given a graph G, a function $f : V(G) \to \{0,1,2\}$ is called a Perfect Roman Dominating Function (PRDF) of G if, for every vertex v in G with $f(v) = 0$, there exists exactly one adjacent vertex u such that $f(u) = 2$. The weight of the function f is defined as $\omega(f) = \sum_{v \in V(G)} f(v)$. The perfect Roman domination number, denoted by $\gamma_R^p(G)$, corresponds to the minimum value of $\omega(f)$ among all perfect Roman dominating functions f of G. A PRDF with minimum weight is called optimal. The perfect Roman domination problem has been the subject of study in recent years and has been shown to be NP-complete for various graph classes. In this undergraduate thesis, were developed two metaheuristics based on genetic algorithms: the first follows a classical approach, while the second is based on the biased random-key genetic algorithm model. Additionally, a integer programming model was implemented in order to provide an exact approach for comparison with the metaheuristic results. Solutions were obtained for different classes of graphs and compared through execution time and fitness value.

Keywords: genetic algorithm; combinatorial optimization; graph theory.

LISTA DE FIGURAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

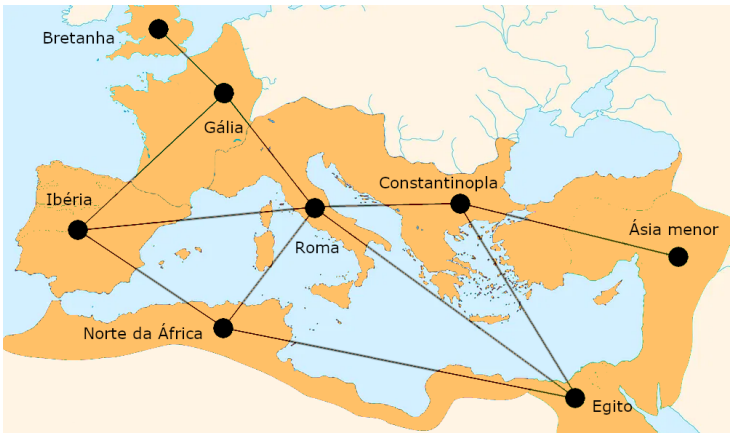
A segurança nacional é um tema central das políticas públicas de países da sociedade moderna, especialmente em países de dimensão continental como o Brasil. Com fronteiras extensas, regiões de difícil acesso e infraestrutura militar desigual, o desafio de proteger eficazmente todo o território nacional torna-se ainda mais complexo diante de restrições orçamentárias.

Recentemente, o Exército Brasileiro emitiu um alerta público sobre os impactos de cortes no orçamento militar, destacando que a redução dos recursos pode comprometer projetos estratégicos e a própria capacidade operacional das Forças Armadas, num momento em que o cenário internacional se mostra cada vez mais instável. Segundo a publicação da Sociedade Militar, "há risco de comprometimento das ações de manutenção da soberania nacional, uma vez que os recursos disponíveis não garantem a operacionalidade plena dos meios militares". (Sociedade Militar, 2025)

Diante dessa realidade, surge a necessidade de ferramentas matemáticas e computacionais que possam auxiliar os departamentos de segurança nacional no planejamento estratégico da defesa, especialmente na alocação eficiente de tropas e recursos militares em regiões prioritárias. A Pesquisa Operacional, a Matemática Aplicada e a Ciência da Computação são disciplinas que historicamente têm auxiliado no planejamento estratégico de países, especialmente em contextos bélicos. Curiosamente, esse tipo de desafio — proteger grandes áreas com recursos limitados — não é exclusivo do mundo moderno. Ao longo da história, situações semelhantes exigiram estratégias criativas e eficientes para garantir a segurança de territórios extensos.

Um exemplo clássico desse tipo de problema foi enfrentado pelo imperador Constantino, o Grande, enquanto o império romano esteve sob o seu comando, no final do século III d.C. e início do século IV d.C (Arquilla, 1995). Com um território vasto (como apresentado na Figura 1) e poucas legiões disponíveis, Constantino precisava tomar decisões estratégicas sobre onde posicionar suas forças para proteger as regiões mais vulneráveis do império.

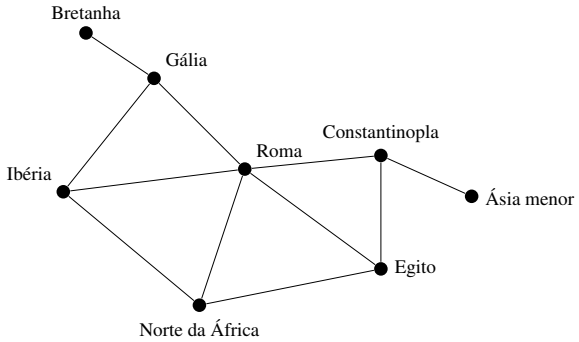
Figura 1 – Império romano de Constantino.



Fonte: autoria própria.

Diante de um império territorialmente extenso e com o número de legiões reduzido, surgia o desafio: como alocar tropas de forma eficiente para garantir a proteção de todas as regiões? Esse cenário histórico serviu de inspiração para o problema matemático conhecido como *dominação romana*. Um dos primeiros estudos formais relacionados a esse problema foi proposto por Ian Stewart (Stewart, 1999), e posteriormente discutido por Charles S. ReVelle e Kenneth E. Rosing (ReVelle; Rosing, 2000), que exploraram esse cenário sob a ótica da Teoria dos Grafos e da Pesquisa Operacional, respectivamente. A Figura 2 apresenta o império romano de Constantino como um grafo.

Figura 2 – Império romano de Constantino como um grafo.



Fonte: autoria própria.

Com apenas quatro legiões disponíveis, o imperador Constantino e seus estrategistas estabeleceram duas regras fundamentais para definir uma estratégia eficiente de defesa:

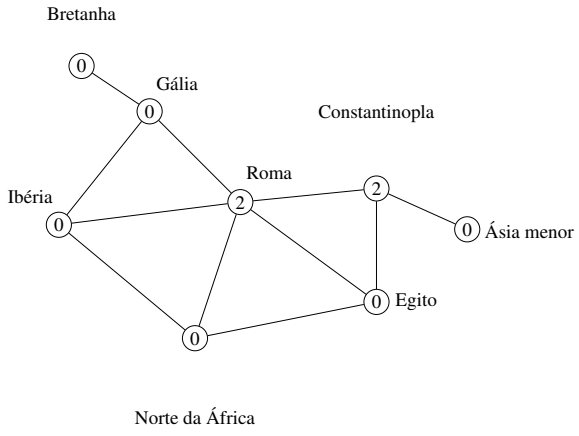
– Uma província é considerada segura se ela já tiver uma legião alocada nela, ou se uma legião puder se locomover até ela em um único passo.

– Uma província poderia enviar tropas a outra província somente se nela houvesse no mínimo duas legiões.

O principal objetivo dessas regras era permitir que províncias sem tropas pudessem receber reforços de províncias vizinhas, sem que a província doadora ficasse desprotegida. Adotando tais diretrizes, Constantino decidiu alocar suas legiões nas suas duas principais províncias: Roma e Constantinopla. Essa configuração deixou a região da Bretanha vulnerável, tornando necessária uma resposta rápida diante de uma ameaça iminente. A solução mais viável consistia em realizar uma série de movimentações estratégicas: uma legião seria transferida de Roma para a Gália, enquanto outra partiria de Constantinopla em direção a Roma. Em seguida, uma tropa de Roma reforçaria a Gália, permitindo, por fim, que uma legião da Gália fosse enviada à Bretanha. No entanto, como o deslocamento das legiões exigia um tempo considerável, é provável que essa lentidão tenha sido um dos principais fatores que contribuíram para a conquista da Bretanha por forças inimigas (Stewart, 1999).

Segundo Stewart (1999), este cenário pode ser modelado matematicamente como um problema de rotulação em grafos, no qual cada vértice representa uma província, cada aresta representa uma conexão entre duas províncias e o rótulo atribuído ao vértice representa a quantidade de tropas alocadas. O conjunto de rótulos possíveis é dado por $\{0,1,2\}$, e a regra principal é que todo vértice com rótulo 0 (isto é, sem tropas) deve ser adjacente a pelo menos um vértice com rótulo 2 (capaz de fornecer reforços), garantindo assim que todas as regiões estejam protegidas. A Figura 3 representa o império romano de Constantino como um grafo rotulado após a decisão de posicionar suas tropas em Roma e Constantinopla.

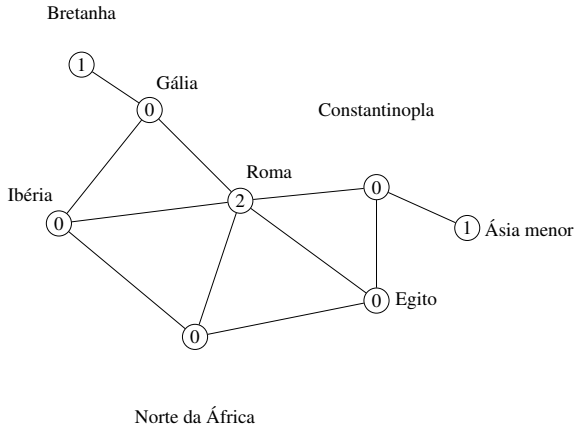
Figura 3 – Império romano de Constantino após alocação das legiões.



Fonte: autoria própria.

Inspirados no trabalho de Stewart, os autores Cockayne et al. formalizaram o problema romano de alocação de tropas, introduzindo assim o conceito formal de dominação romana em grafos, como definido a seguir (Cockayne *et al.*, 2004). Seja $G = (V(G), E(G))$ um grafo simples e não direcionado. Uma função $f\colon V(G) \to \{0,1,2\}$ é uma *função de dominação romana* (FDR) se todo vértice $v \in V(G)$ com $f(v) = 0$ possui pelo menos um vizinho $w \in V(G)$ com $f(w) = 2$. O *peso* de uma função de dominação romana f é dado por $\omega(f) = \sum_{v \in V(G)} f(v)$. O *número de dominação romana* de G, denotado por $\gamma_R(G)$, é o menor peso que uma função de dominação romana de G pode ter, ou seja, $\gamma_R(G) = \min\{\omega(f) : f \text{ é uma FDR de } G\}$. Uma função de dominação romana que tem o menor peso possível é chamada de *ótima*. Nesse contexto, a Figura 4 exemplifica uma solução ótima aplicada ao grafo que modela o império de Constantino. Essa configuração estratégica garante a segurança da região da Bretanha ao mesmo tempo em que assegura o controle de todas as demais províncias.

Figura 4 – Solução ótima para o problema de alocação de legiões no império de Constantino.



Fonte: autoria própria.

O desafio, portanto, é determinar a menor quantidade total de tropas (ou seja, a menor soma de rótulos) necessária para garantir a segurança de todo o grafo. A versão de decisão do *Problema da Dominação Romana* (PDR) consiste em, dado um grafo arbitrário G e um inteiro k, decidir se $\gamma_R(G) \leq k$. Esse problema foi demonstrado como sendo *NP*-completo até mesmo quando restrito a grafos cordais, bipartidos, split e planares (Dreyer, 2000; Cockayne *et al.*, 2004).

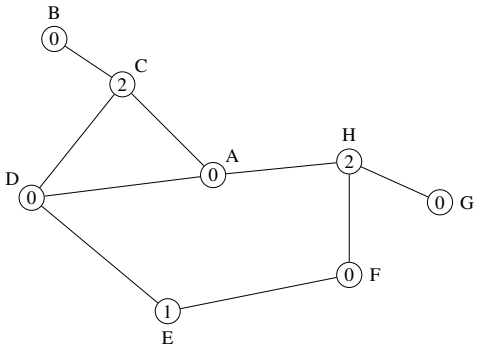
Diversas variações do conceito clássico de dominação romana foram propostas na literatura, com o objetivo de modelar cenários estratégicos mais específicos ou com restrições

adicionais, alterando as regras de alocação ou intensificando os critérios de defesa (Chellali *et al.*, 2020; Chellali *et al.*, 2021). Em algumas dessas variantes, por exemplo, o conjunto de rótulos permitidos para os vértices pode ser expandido para $\{0,1,2,\ldots,k\}$, permitindo maior flexibilidade na distribuição dos recursos (legiões). Em outras variantes, o critério para considerar um vértice protegido pode ser relaxado ou reforçado. Um exemplo disso é a dominação romana fraca, em que a exigência de um vizinho com rótulo 2 é substituída por um vizinho com rótulo positivo que possa realocar sua defesa de forma que nenhum vértice fique desprotegido. Já na dominação romana independente, impõe-se que os vértices com rótulo 1 ou 2 formem um conjunto independente, refletindo cenários em que não é desejável posicionar tropas em locais adjacentes. Em variações como a dominação romana total ou mista, busca-se proteger não apenas os vértices, mas também as arestas do grafo, o que simula situações em que tanto os locais quanto as rotas de comunicação precisam de cobertura. Essas e outras diversas variações contribuem para enriquecer a modelagem do problema, permitindo uma representação mais precisa de diferentes cenários estratégicos. Além disso, ampliam significativamente sua aplicabilidade, tanto em contextos teóricos quanto em situações práticas mais complexas.

Como uma nova abordagem do problema de dominação romana, surgiu o problema de *dominação romana perfeita*, proposto por Henning, Klostermeyer e MacGillivray em 2017 (Henning *et al.*, 2017). Essa variante é definida a seguir. Dado um grafo simples G, dizemos que uma função $f\colon V(G) \to \{0,1,2\}$ é uma *função de dominação romana perfeita* (FDRP) de G se todo vértice com rótulo 0 possui exatamente um vizinho com rótulo 2. O *peso* de uma função de dominação romana perfeita f é dado por $\omega(f) = \sum_{v \in V(G)} f(v)$. O *número de dominação romana perfeita* de G, denotado por $\gamma_R^p(G)$, é o menor peso que uma função de dominação romana perfeita de G pode ter. Uma função de dominação romana perfeita que tem o menor peso possível é chamada de *ótima*.

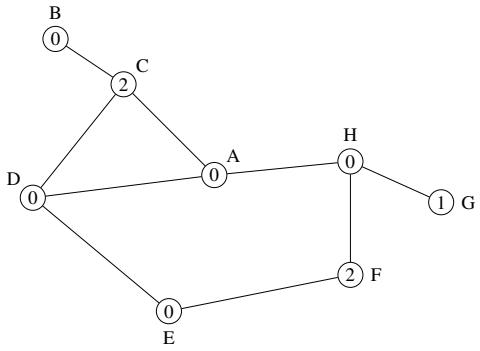
A Figura 5 mostra um grafo com os vértices rotulados de forma a representar uma solução para o problema de *dominação romana*. Ao seu lado, a Figura 6 apresenta o mesmo grafo, mas com uma rotulação que satisfaz as condições do problema de *dominação romana perfeita*.

Figura 5 – Rotulação de um grafo segundo o problema de dominação romana.



Fonte: autoria própria.

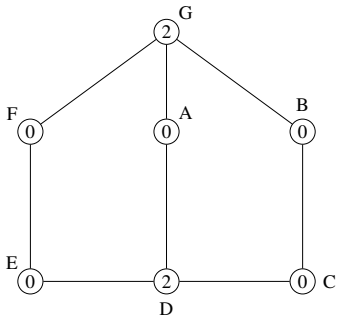
Figura 6 – Rotulação do mesmo grafo segundo o problema de dominação romana perfeita.



Fonte: autoria própria.

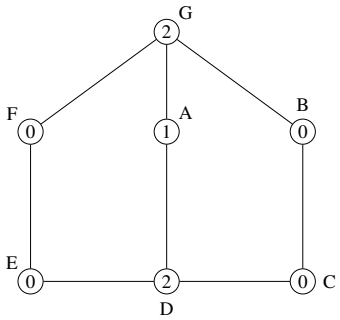
Note que, embora a dominação romana perfeita seja uma variante do problema de dominação romana, ambas as rotulações apresentadas nas Figuras 5 e 6 resultaram na mesma soma total de rótulos atribuídos aos vértices. Isso indica que, para esse grafo específico, a solução obtida pela variação perfeita teve o mesmo custo da solução do problema base, ainda que os valores tenham sido distribuídos de forma diferente entre os vértices. Entretanto, nem sempre é possível obter os mesmos valores, como é apresentado nas Figuras 7 e 8. Decorre das definições que toda função de dominação romana perfeita é uma função de dominação romana, isso imediatamente implica que $\gamma_R(G) \leq \gamma_R^p(G)$ para todo grafo G, mas a recíproca não é verdadeira.

Figura 7 – Grafo com uma função de dominação romana de peso 4.



Fonte: autoria própria.

Figura 8 – Grafo com uma função de dominação romana perfeita de peso 5.



Fonte: autoria própria.

O problema de dominação romana perfeita é NP-completo, mesmo quando restrito a diversas classes de grafos (Banerjee *et al.*, 2019; Darkooti *et al.*, 2019; Chakradhar; Reddy, 2021; Mann; Fernau, 2024), o que evidencia sua elevada complexidade computacional. Ademais, embora um modelo exato baseado em programação inteira tenha sido proposto na literatura

por Mirhoseini e Rad (2023), até o momento não há registros de sua aplicação prática a instâncias de grafos de bases estudadas na literatura.

Nesse contexto, as limitações inerentes aos métodos exatos para o tratamento de instâncias de grande porte motivam a investigação de abordagens heurísticas e aproximadas. Assim, este trabalho propõe o uso de algoritmos genéticos como uma estratégia promissora para a resolução do problema de dominação romana perfeita em grafos, com foco na obtenção de soluções de boa qualidade para instâncias de grafos extraídas de bases de dados comumente usadas na literatura.

1.1 Contribuições deste trabalho

Este trabalho de conclusão de curso apresenta dois algoritmos Algoritmos Genéticos (do inglês, *Genetic Algorithm (GA)*) aplicados ao Problema de Dominação Romana Perfeita, contemplando tanto um modelo clássico de GA quanto uma variante denominada Algoritmo Genético de Chaves Aleatórias Enviesadas (do inglês, *Biased Random-Key Genetic Algorithm (BRKGA)*). Ademais, o presente trabalho corrige e melhora um modelo de programação inteira (PI), originalmente proposto por Mirhoseini e Rad (2023). Disponibilizamos em repositório público, a base de grafos utilizada nos experimentos computacionais e fornecemos o código-fonte completo de todas as implementações desenvolvidas ao longo da pesquisa, assim como os documentos empregados na obtenção dos resultados.

No que se refere aos resultados, foram calculados os valores exatos de γ_R^p para diversos grafos da base considerada, os quais podem ser utilizados como referência em trabalhos futuros. As análises realizadas demonstram que as metaheurísticas propostas são capazes de explorar eficientemente o espaço de busca, alcançando, na maioria dos casos, soluções ótimas ou de alta qualidade, o que evidencia a viabilidade e o potencial do uso de abordagens evolutivas para a resolução do PDRP.

1.2 Organização do texto

A organização dos próximos capítulos está estruturada como segue. O Capítulo 2 apresenta a fundamentação teórica necessária para o desenvolvimento da pesquisa, abordando os principais conceitos relacionados à dominação em grafos, dominação romana, dominação romana perfeita, programação linear (PL), algoritmos genéticos e técnicas metaheurísticas. O

Capítulo 3 reúne os trabalhos correlatos, destacando as abordagens existentes e suas contribuições para a resolução do problema de dominação romana perfeita, servindo como base comparativa para este estudo. O Capítulo 4 propõe uma correção e um novo modelo de programação inteira para o PDRP. O Capítulo 5 descreve detalhadamente a variante de Algoritmo Genético clássico desenvolvida neste trabalho. O Capítulo 6 descreve em detalhes o Algoritmo genético de chaves aleatórias enviesadas. O Capítulo 7 discute os experimentos realizados e os resultados obtidos. Por fim, o Capítulo 8 aborda as conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

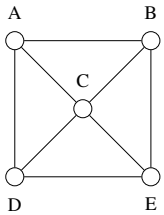
Este capítulo tem como objetivo apresentar os conceitos básicos que serão utilizados durante o restante do texto.

2.1 Conceitos básicos de teoria dos grafos

As definições e conceitos apresentados nesta seção foram baseados no livro de Chartrand e Zhang (Chartrand; Zhang, 2012). Todos os grafos considerados neste trabalho são grafos simples e finitos. Para fins de simplificação, utilizaremos apenas o termo grafo ao nos referirmos aos grafos simples.

Um *grafo* $G = (V(G), E(G))$ é uma estrutura matemática abstrata formada por dois conjuntos denotados por $V(G)$ e $E(G)$. O conjunto $V(G)$ é um conjunto finito e não vazio de elementos denominados *vértices*, enquanto o conjunto $E(G)$ é um conjunto disjunto de $V(G)$ formado por pares não ordenados de elementos distintos de $V(G)$, pares estes denominados *arestas*. Uma aresta $\{u, v\} \in E(G)$ poder ser denotada por uv ou vu, indicando que os vértices u e v são *adjacentes* no grafo. Dizemos também que a aresta uv *incide* nos vértices u e v e vice-versa. Se duas arestas compartilham o mesmo vértice, então dizemos que são *arestas adjacentes*. Também é comum os vértices serem chamados de *nós* e as arestas de *linhas* e dizer que dois vértices adjacentes são *vizinhos*. Um grafo é comumente representado por um diagrama desenhado em um plano, no qual os vértices são representados por pontos no plano e as arestas são representadas como linhas retas ou curvas entre dois vértices. A Figura 9 demonstra um grafo H, onde o conjunto $V(H) = \{A, B, C, D, E\}$ e o conjunto $E(H) = \{AB, AD, AC, BC, BE, DC, DE, EC\}$.

Figura 9 – Exemplo de grafo H.



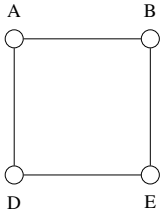
Fonte: autoria própria.

A *ordem* de um grafo G, denotado por $|V(G)|$, corresponde a quantidade de vértices presentes no grafo. Já o número de arestas, representado por $|E(G)|$, é denominado *tamanho* do grafo. Assim, para o grafo H anteriormente descrito, temos $|V(H)| = 5$ e $|E(G)| = 8$. Portanto,

a ordem mínima de qualquer grafo é, no mínimo, igual a 1, uma vez que não há grafos sem vértices. Convencionalmente utilizam-se as letras n e m para representar, respectivamente, a ordem e o tamanho de um grafo. Aplicando essa notação para o grafo H, temos $n = 5$ e $m = 8$.

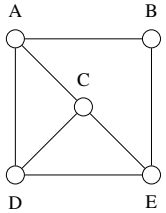
Um grafo T é chamado de *subgrafo* de G se $V(T) \subseteq V(G)$ e $E(T) \subseteq E(G)$. Nessas condições, dizemos que $T \subseteq G$. A Figura 10 ilustra um subgrafo T do grafo H ilustrado na Figura 9. Quando um subgrafo T possui o mesmo conjunto vértices de G, ou seja, $V(T) = V(G)$, ele é denominado *subgrafo gerador* do grafo G. É possível remover um vértice de um grafo G, essa operação acarreta na remoção das arestas que incidem no vértice. Formalmente, dado um vértice $v_1 \in V(G)$, denotamos por $G - \{v_1\}$ o grafo obtido a partir de G removendo o vértice v_1 e todas as arestas que incidem nele. De forma análoga, também é possível remover arestas específicas de um grafo. A notação $G - \{v_1v_2\}$, com $\{v_1v_2\} \in E(G)$, representa o grafo obtido a partir de G pela remoção da aresta v_1v_2, mantendo os demais vértices e arestas. A Figura 11 apresenta um grafo D tal que $V(D) = V(H)$ e $E(D) = E(H) - \{BC\}$, ou seja, o grafo D resulta da remoção da aresta BC do grafo H ilustrado na Figura 9.

Figura 10 – Subgrafo T do grafo H.



Fonte: autoria própria.

Figura 11 – Grafo D.

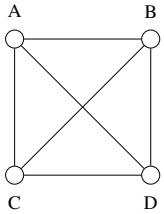


Fonte: autoria própria.

A *vizinhança aberta* de um vértice $v \in V(G)$ é o conjunto formado por todos os vértices que são vizinhos a v, e é denotada por $N_G(v)$. Já a *vizinhança fechada* de um vértice $v \in V(G)$, denotado por $N_G[v]$, inclui o próprio vértice v além de seus vizinhos, ou seja, $N_G[v] = v \cup N_G(v)$.

O *grau* de um vértice $v \in V(G)$, denotado por $d_G(v)$, é a quantidade de arestas que incidem em v. Um vértice de grau 0 é um *vértice isolado*. O *grau máximo* de um grafo G, denotado por $\Delta(G)$, é definido como $\Delta(G) = \max\{d(v) \mid v \in V(G)\}$. De forma análoga, o *grau mínimo* de um grafo G, denotado por $\delta(G)$, é definido como $\delta(G) = \min\{d(v) \mid v \in V(G)\}$. Um grafo k-regular é um grafo em que todos os vértices possuem grau igual a k. Um grafo G é dito *cúbico* se G é 3-regular. A Figura 12 apresenta um exemplo de um grafo cúbico.

Figura 12 – Exemplo de grafo cúbico.

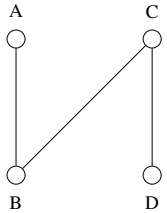


Fonte: autoria própria.

Dado um grafo G, um *caminho* em G é uma sequência de vértices distintos de G dado por $v_1, v_2, \ldots, v_k$ tal que $v_i v_{i+1} \in E(G)$ para todo $i = 1, 2, \ldots, k-1$. O *comprimento* de um caminho é o número de arestas do caminho. É comum dizer que os vértices v_1 e v_k são os *extremos* do caminho. A *distância* entre dois vértices u e v em um grafo G é o comprimento do menor caminho em G que tem u e v como extremos e é denotada por $d(u,v)$. Caso haja um caminho entre quaisquer dois vértices u e v de um grafo G, então dizemos que G é *conexo*, caso contrário, dizemos que G é *desconexo*. Uma *componente conexa* de G é um subgrafo conexo e maximal de G.

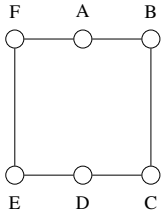
Um grafo de ordem k, cujo os vértices podem ser distruibuídos em uma sequência linear $v_0, v_1, v_2, ..., v_{k-1}$, de forma que dois vértices são adjacentes se, e somente se, eles são consecutivos na sequência é chamado de *caminho* e é denotado por P_k. Já um grafo de ordem k, com $k \geq 3$, cujos vértices estão dispostos em uma sequência cíclica $v_0, v_1, v_2, ..., v_{k-1}, v_0$, de modo que dois vértices são adjacentes se, e somente se, eles são consecutivos na sequência cíclica, é chamado de *ciclo* e é denotado por C_k. A Figura 13 apresenta um caminho que vai do vértice A até o D ou vice-versa. Já a Figura 14 exibe um ciclo que tem início e fim no vértice A.

Figura 13 – Exemplo de caminho P_4.



Fonte: autoria própria.

Figura 14 – Exemplo de um ciclo C_6.



Fonte: autoria própria.

Um grafo de ordem k é chamado de *grafo completo*, e é denotado por K_k, se todo par distinto de vértices é adjacente. A Figura 12 apresenta o grafo completo K_4.

2.2 Dominação romana perfeita em grafos

Com a definição formal de dominação romana perfeita, a contextualização do problema e a apresentação do número de dominação romana perfeita, $\gamma_R^p(G)$, estabelecidas no Capítulo 1, a presente seção apresenta propriedades adicionais deste conceito. São apresentados alguns limitantes inferiores e superiores para $\gamma_R^p(G)$, bem como alguns teoremas que fundamentam a pesquisa e as contribuições deste trabalho.

Para uma função de dominação romana perfeita $f : V(G) \to \{0,1,2\}$, define-se, para cada $i \in \{0,1,2\}$, o conjunto $V_i = \{v \in V(G) \mid f(v) = i\}$. A tripla ordenada (V_0, V_1, V_2) constitui, portanto, uma partição do conjunto de vértices $V(G)$. Por conveniência, representamos a função f por essa tripla, a qual também chamamos de função de dominação romana. Note que $\omega(f) = |V_1| + 2|V_2|$.

Para efeito das análises e resultados apresentados ao longo deste trabalho, adota-se a seguinte noção de dominância: um vértice $v \in V(G)$ é dito *dominado* se satisfaz pelo menos uma das seguintes condições: (i) v possui rótulo 2; (ii) v possui rótulo 1; ou (iii) v possui exatamente um vizinho $u \in N_G(v)$ tal que u possui rótulo 2.

Proposição 2.2.1. *(Yue; Song, 2020) Seja G um grafo, então* $\gamma_R^p(G) \geq \gamma_R(G)$.

Demonstração. Como toda FDRP é uma FDR, então $\gamma_R^p(G) \geq \gamma_R(G)$. □

A Proposição 2.2.2 foi originalmente apresentada sem demonstração no trabalho de Banerjee et al. (Banerjee *et al.*, 2019). Apresentamos a seguir uma demonstração para este resultado.

Proposição 2.2.2. *(Banerjee et al., 2019) Se G for um grafo conexo com mais de um vértice, então existe uma FDRP de peso mínimo f em G tal que* $f(v) = 2$ *para pelo menos um vértice* $v \in V(G)$.

Demonstração. Para $|V(G)| = 2$, seja $V(G) = \{u, v\}$. A rotulação $f(u) = 2$ e $f(v) = 0$ é uma FDRP de peso mínimo com $f(v) = 2$ para o vértice v. Agora, seja $|V(G)| \geq 3$ e suponha, por absurdo, que exista uma FDRP f de peso mínimo tal que $f(v) \in \{0,1\}$ para todo $v \in V(G)$, ou seja, $V_2 = \emptyset$. Pela definição de FDRP, todo vértice com valor 0 deve ser adjacente a exatamente um vértice com valor 2, o que é impossível se $V_2 = \emptyset$. Logo, $f(v) = 1$ para todo $v \in V(G)$ e o peso total é $\omega(f) = |V(G)|$. Como G é conexo e $|V(G)| \geq 3$, existe um vértice u adjacente a outros dois vértices v e w. Desta forma, podemos definir uma nova função f' com $f'(u) = 2$,

$f'(v) = 0$, $f'(w) = 0$ e $f'(x) = f(x)$ para $x \in V(G) - \{u, v, w\}$. Essa nova função é uma FDRP válida e tem peso $\omega(f') = \omega(f) - 1$, o que contradiz o fato de $\omega(f)$ ser mínimo. Portanto, toda FDRP de peso mínimo em G deve ter pelo menos um vértice com $f(v) = 2$. □

Proposição 2.2.3. *(Yue; Song, 2020) Se G é um grafo de ordem pelo menos 2, então,*

$2 \leq \gamma_R^P(G) \leq n$.

Demonstração. O fato de que $2 \leq \gamma_R^P(G)$ segue imediatamente da Proposição 2.2.2. Agora, seja $f = (\varnothing, V(G), \varnothing)$ uma FDRP de G, então, $\gamma_R^p(G) \leq \omega(f) = n$. □

Pela Proposição 2.2.3, temos que $\gamma_R^p(G) \leq n$. Porém, como mostra o lema a seguir, os únicos grafos que atingem esse limite superior são os grafos cujas componentes conexas possuem no máximo dois vértices.

Lema 2.2.1. *(Yue; Song, 2020) Seja G um grafo de ordem $n \geq 2$. Então $\gamma_R^p(G) = n$ se, e somente se, $G = tK_2 \cup (n-2t)K_1$, onde $0 \leq t \leq \lfloor \frac{n}{2} \rfloor$.*

Demonstração. Se $G = tK_2 \cup (n-2t)K_1$, então todo vértice isolado deve ser atribuído a 1, e os vértices em K_2 de G podem ser atribuídos como (0,2) ou (1,1). Portanto, $\gamma_R^p(G) = n$. Seja G o grafo com $\gamma_R^p(G) = n$. Então cada componente conexo de G tem no máximo 2 vértices. Caso contrário, existem 3 vértices x, y, z em algum componente conexo de G, tal que xyz é um caminho. Agora, definimos uma FDRP como $\{(x,z), V(G) \setminus \{x,y,z\}, \{y\}\}$. Assim, $\gamma_R^p \leq n-1$, uma contradição. Então, cada componente de G é K_2 ou K_1. Isso quer dizer que, $G = tK_2 \cup (n-2t)K_1$, para algum t, onde $0 \leq t \leq \lfloor \frac{n}{2} \rfloor$. □

Pela Proposição 2.2.1, limitantes inferiores para a dominação romana também são limitantes inferiores para a dominação romana perfeita. Logo a seguir, apresentamos um deles.

Teorema 2.2.1. *(Cockayne et al., 2004) Se G é um grafo conexo com grau máximo Δ e ordem $n \geq 2$, então $\gamma_R(G) \geq \frac{2n}{\Delta+1}$.*

Demonstração. Seja $f = (V_0, V_1, V_2)$ uma FDR de G com peso $\gamma_R(G)$. Como cada vértice $v \in V_0$ é adjacente a pelo menos um vértice em V_2, podemos observar que:

$$|V_0| \leq \Delta |V_2| \tag{2.1}$$

Como $\gamma_R(G) = |V_1| + 2|V_2|$, temos que:

$$\begin{aligned}(\Delta+1)\gamma_R(G) &= (\Delta+1)|V_1| + (\Delta+1)2|V_2| \\ &= (\Delta+1)|V_1| + 2|V_2| + 2\Delta|V_2| \\ &\geq (\Delta+1)|V_1| + 2|V_2| + 2|V_0| \quad \text{pela desigualdade (2.1)} \\ &\geq 2|V_1| + 2|V_2| + 2|V_0| \\ &= 2n.\end{aligned}$$

Assim, $(\Delta+1)\gamma_R(G) \geq 2n$. □

Corolário 2.2.1. *(Yue; Song, 2020) Se G é um grafo conexo com grau máximo* Δ *e ordem* $n \geq 2$*, então* $\gamma_R^p(G) \geq \frac{2}{\Delta+1}$.

Demonstração. Pela Proposição 2.2.1, temos que $\gamma_R^p(G) \geq \gamma_R(G)$. Já pelo Teorema 2.2.1, sabe-se que $\gamma_R(G) \geq \frac{2n}{\Delta+1}$. Logo concluímos que $\gamma_R^p(G) \geq \frac{2}{\Delta+1}$. □

A base de grafos utilizada neste trabalho é composta, dentre outros, por grafos cúbicos. A seguir, apresentamos um limite superior conhecido para o número de dominação romana perfeita desta classe.

Teorema 2.2.2. *(Henning; Klostermeyer, 2018) Se G é um grafo 3-regular de ordem n, então* $\gamma_R^p(G) \leq \frac{3}{4}n$*, e esse limitante é o melhor possível.*

2.3 Programação linear

A *programação linear* (PL) é uma técnica matemática clássica da Pesquisa Operacional cujo objetivo é otimizar — por meio da maximização ou minimização — uma função linear, sujeita a um conjunto de restrições também lineares. Esse tipo de modelagem é amplamente utilizada em problemas de alocação de recursos escassos, planejamento da produção, logística e em diversas situações nas quais há linearidade entre variáveis e restrições.

A programação linear usa um modelo matemático para descrever o problema em questão. O adjetivo linear significa que todas as funções matemáticas nesse modelo são necessariamente funções lineares. A palavra programação, nesse caso, não se refere à programação de computador; ela é, essencialmente, um sinônimo para planejamento. Portanto, a programação linear envolve o planejamento de atividades para obter um resultado ótimo, isto é, um resultado

que atinja o melhor objetivo especificado (de acordo com o modelo matemático) entre todas as alternativas viáveis (Hillier; Lieberman, 2013).

Um modelo de PL busca representar tanto a função objetivo quanto as restrições do problema por meio de expressões lineares. Além disso, assume-se que todas as variáveis de decisão são contínuas. Em um modelo de programação linear, as restrições devem refletir fielmente as condições do problema real. Dessa forma, uma solução é considerada viável se, e somente se, satisfaz todas as restrições impostas. Por fim, o valor máximo (ou mínimo) da função objetivo, obtido entre todas as soluções viáveis, corresponde à uma solução ótima do problema. Quando expresso em sua forma canônica, o modelo apresenta todas as restrições como desigualdades lineares.

Um modelo de programação linear na *forma canônica* é expresso como:

$$\begin{aligned} \max \quad & z = c^T x \\ \text{sujeito a} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

Em que:

- $x \in \mathbb{R}^n$ representa o vetor das variáveis de decisão;
- $c \in \mathbb{R}^n$ é o vetor de coeficientes da função objetivo;
- $A \in \mathbb{R}^{m \times n}$ é a matriz de coeficientes das restrições;
- $b \in \mathbb{R}^m$ é o vetor de constantes do lado direito das restrições;
- $z \in \mathbb{R}$ representa o valor da função objetivo.

Em 1947, George Dantzig desenvolveu um algoritmo eficiente para a resolução de problemas de programação linear, denominado *simplex* (Nash, 2000). Trata-se de um método iterativo e exato que visa encontrar a solução ótima de um problema linear. A lógica do algoritmo baseia-se na navegação entre soluções viáveis, que são pontos extremos no politopo definido pelas restrições do problema, sempre em busca de uma alternativa que proporcione uma melhora no valor da função objetivo. O processo é encerrado quando não há mais soluções viáveis que ofereçam ganho em relação à solução atual, indicando que a otimização foi alcançada. (Goldbarg; Luna, 2005)

2.3.1 Programação linear inteira

Além da programação linear clássica, existe uma importante variação chamada *programação linear inteira* (PLI). Nessa modalidade, impõe-se que uma ou mais variáveis de

decisão assumam apenas valores inteiros, restringindo o conjunto de soluções possíveis a um subconjunto discreto (Nemhauser; Wolsey, 1988). De acordo com Goldbarg e Luna (2005), um problema de otimização é classificado como de *programação inteira* (PI) quando pelo menos uma das variáveis não puder assumir valores contínuos, ficando condicionada a valores discretos. Formalmente, em um problema de PLI, as variáveis de decisão x_i devem pertencer ao conjunto dos inteiros $\mathbb{Z}$. Essa restrição, apesar de simples em sua formulação, introduz um aumento significativo na complexidade computacional do problema (Schrijver, 1986).

A PLI é especialmente relevante em situações onde decisões devem ser tomadas em quantidades indivisíveis — como alocação de unidades inteiras, escalas de trabalho ou escolha de caminhos binários (0 ou 1). Para resolver problemas de PLI, são necessárias técnicas mais sofisticadas do que o método simplex, como ramificação e delimitação (branch-and-bound) (Wolsey, 1998) e métodos de enumeração, ou combinações como branch-and-cut (Bertsimas; Weismantel, 2005).

Um conceito fundamental para a resolução de problemas de PLI é o *relaxamento linear*. Este consiste em remover as restrições de integralidade das variáveis, convertendo o problema em uma programação linear contínua. A solução ótima do relaxamento linear fornece um limitante dual: um valor superior no caso de problemas de maximização e inferior no caso de minimização. Se a solução do relaxamento linear resultar em valores inteiros para todas as variáveis que deveriam ser inteiras, então essa solução é, de fato, a solução ótima para o problema de PLI (Nemhauser; Wolsey, 1988).

2.4 Problema de Programação Inteira de Mirhoseini e Rad (2023)

Um modelo de PI para o PDRP foi proposto em 2023 por Mirhoseini e Rad (2023) e é apresentado a seguir. Seja G um grafo com $n = |V(G)|$ e $m = |E(G)|$. Ademais, seja $f : V(G) \to \{0,1,2\}$ uma FDRP de G. O problema de dominação romana perfeita pode ser modelado como um programa linear inteiro. Este modelo usa o conjunto de 3 variáveis binárias a_v, b_v, c_v. Para cada vértice $v \in V(G)$, defina:

$$a_v = \begin{cases} 1, & \text{se } f(v) = 0; \\ 0, & \text{caso contrário.} \end{cases} \quad b_v = \begin{cases} 1, & \text{se } f(v) = 1; \\ 0, & \text{caso contrário.} \end{cases} \quad c_v = \begin{cases} 1, & \text{se } f(v) = 2; \\ 0, & \text{caso contrário.} \end{cases}$$

De acordo com esse modelo, o peso de uma FDRP é dado por $\sum_{v \in V(G)} (b_v + 2c_v)$.

Portanto, o programa inteiro para o problema de dominação romana perfeita foi formulado como:

$$\text{minimize} \quad \sum_{v \in V(G)} (b_v + 2c_v) \tag{2.2a}$$

$$\text{subject to} \quad 2b_v + c_v + (1 - b_v)(1 - c_v) \sum_{u \in N(v)} 2c_u = 2 \quad v \in V(G), \tag{2.2b}$$

$$a_v + b_v + c_v = 1 \qquad v \in V(G), \tag{2.2c}$$

$$a_v, b_v, c_v \in \{0, 1\} \qquad v \in V(G) \tag{2.2d}$$

Segundo os autores, a função objetivo (2.2a) minimiza o peso de f. As restrições (2.2b) garantem que todo vértice rotulado com 0 seja adjacente a exatamente um vértice v com $f(v) = 2$. As restrições (2.2c) garantem que exatamente um rótulo seja atribuído a todo vértice e as restrições (2.2d) garantem que as variáveis sejam binárias.

Foram identificados erros nesta modelagem, bem como oportunidades de aprimoramento, que são apresentados no Capítulo 4.

2.5 Algoritmos genéticos

As definições e conceitos apresentados nesta seção foram baseados majoritariamente na obra de Sivanandam e Deepa (Sivanandam; Deepa, 2007), a qual fornece uma introdução clara e sistemática à algoritmos genéticos.

Charles Darwin apresentou a teoria da evolução natural em sua obra *A Origem das Espécies*. De acordo com essa teoria, os organismos biológicos passam por um processo de evolução ao longo de várias gerações, guiados pelo princípio da seleção natural, ou seja, a sobrevivência dos mais aptos. Isso significa que características vantajosas para a adaptação ao ambiente tendem a ser preservadas e aprimoradas com o tempo. Exemplos notáveis dessa eficiência evolutiva podem ser observados nas formas aerodinâmicas do albatroz, na habilidade de natação dos tubarões e na semelhança funcional entre tubarões e golfinhos — apesar de pertencerem a grupos distintos.

A notável eficácia com que a evolução natural é capaz de gerar soluções complexas e adaptativas na natureza tem inspirado o desenvolvimento de algoritmos computacionais que simulam esse processo com o intuito de resolver problemas práticos. Nesse sentido, observa-se um crescente interesse na formulação de métodos baseados em princípios evolutivos para enfrentar desafios de otimização e busca em diversas áreas do conhecimento. Inserem-se nesse contexto as

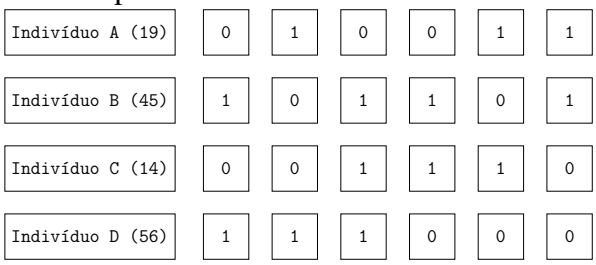
metaheurísticas, concebidas como estruturas algorítmicas de alto nível, independentes do problema, que estabelecem diretrizes gerais para a construção de algoritmos de otimização heurística. Diferentemente dos métodos exatos — que asseguram a obtenção da solução ótima em tempo finito —, as metaheurísticas visam encontrar soluções suficientemente boas dentro de limites computacionais viáveis, evitando, assim, a explosão combinatória associada à complexidade de problemas classificados como NP-difíceis (Sörensen; Glover, 2013).

Classificado como uma metaheurística, um *algoritmo genético* (GA) é uma técnica de busca estocástica e iterativa inspirada nos princípios da seleção natural e genética biológica. Ele opera sobre uma *população* de indivíduos, onde cada indivíduo representa uma possível solução codificada do problema. A evolução das soluções ocorre por meio da aplicação sistemática de operadores genéticos como *seleção*, *cruzamento (crossover)* e *mutação*, que podem ser implementados de diversas formas, com variações adaptadas ao problema em questão, com o objetivo de maximizar (ou minimizar) uma *função de aptidão*, também chamada de *fitness*, que avalia a qualidade de cada solução. Um aspecto fundamental dessa abordagem é a preservação *elitista*, na qual os melhores indivíduos de uma geração são diretamente transferidos para a próxima, garantindo que as soluções mais promissoras não se percam no processo evolutivo.

Algoritmos genéticos vêm sendo amplamente investigados na literatura e demonstram potencial na resolução de problemas complexos de otimização combinatória, entre os quais insere-se o problema de dominação romana perfeita (Khandelwal; Saran, 2021; Raju; Reddy, 2024; Aggarwal; Reddy, 2024). Essa categorização decorre de sua capacidade de realizar buscas robustas e eficazes em espaços de solução grandes e complexos, mesmo sem o uso de derivadas, conhecimento prévio aprofundado ou estrutura específica da função objetivo.

Para ilustrar o funcionamento de um algoritmo genético padrão, considere o problema de maximizar um número binário de 6 bits, em que cada solução candidata é representada por uma string de comprimento fixo composta apenas pelos valores $\{0, 1\}$. A função de aptidão interpreta cada string como um número binário e atribui à solução um valor correspondente em decimal. Por exemplo, a string 101101 representa o número decimal 45. O objetivo, portanto, é encontrar a string que representa o maior valor possível — ou seja, 111111, que corresponde a 63. A Figura 15 ilustra a população inicial, composta por indivíduos gerados de forma aleatória, utilizada na resolução do problema mencionado.

Figura 15 – População inicial do problema de maximizar um número binário de 6 bits.



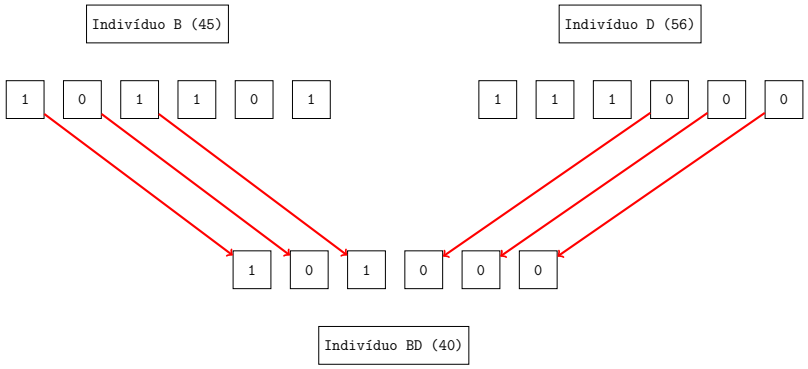
Fonte: autoria própria.

A partir dessa população, o algoritmo avalia cada indivíduo com base em sua aptidão, neste caso definida pelo valor decimal correspondente à string binária. Em seguida, os indivíduos passam por uma etapa de *seleção*, na qual são aplicadas estratégias para identificar os mais aptos a participarem do processo de reprodução (Miller; Goldberg, 1995; Goldberg; Deb, 1991) — ou seja, para combinar suas informações genéticas e gerar novos indivíduos, esse processo é denominado *cruzamento* (do inglês, *crossover*). Essa combinação pode ser feita trocando partes de duas strings, o que simula o cruzamento de características entre os pais. Por exemplo, ao combinar parte da string do indivíduo $B = 101101$ com parte da string do indivíduo $D = 111000$, podemos obter dois novos indivíduos:

- Novo indivíduo BD: 101000 (valor 40)
- Novo indivíduo DB: 111101 (valor 61)

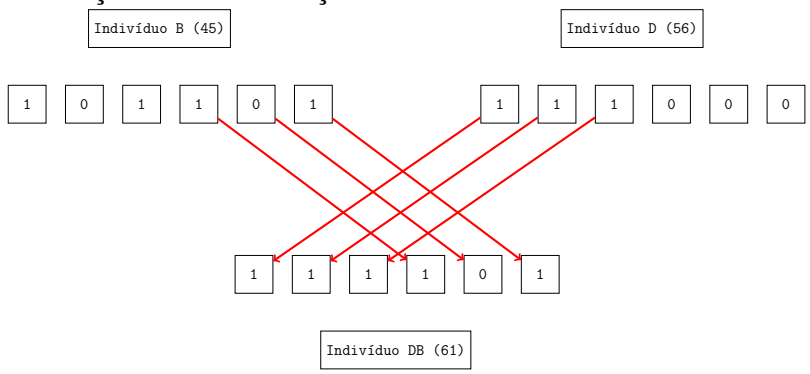
A Figura 16 ilustra visualmente o processo de *crossover* utilizado para gerar o indivíduo BD. De maneira análoga, a Figura 17 apresenta o mesmo procedimento aplicado na criação do indivíduo DB.

Figura 16 – Demonstração visual da criação do indivíduo BD.



Fonte: autoria própria.

Figura 17 – Demonstração visual da criação do indivíduo DB.



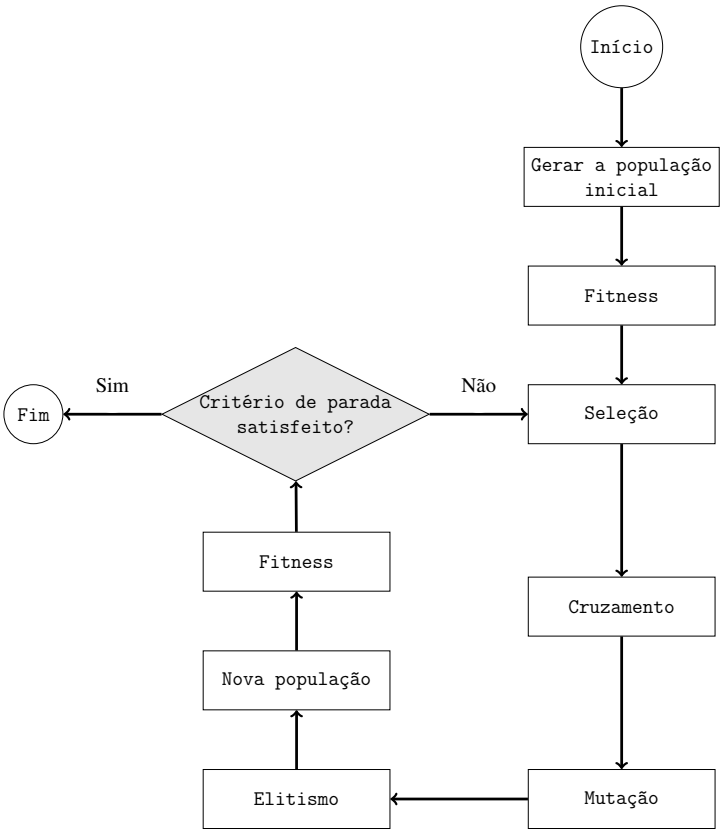
Fonte: autoria própria.

Um outro operador genético que frequentemente é aplicado aos cromossomos da população é a *mutação* — uma pequena alteração aleatória em algum dos dígitos binários da string. Esse processo é essencial para manter a diversidade genética na população e evitar que o algoritmo convirja prematuramente para soluções intermediárias. Por exemplo, o indivíduo $BD = 101000$ pode sofrer uma mutação no último dígito, transformando-se em 101001 (valor 41).

Outra estratégia importante adotada neste processo é o *elitismo*, que consiste em preservar automaticamente os melhores indivíduos da geração anterior. Assim, mesmo que o processo de cruzamento e mutação gere soluções menos eficazes, as boas soluções não são descartadas. No exemplo, o indivíduo $D = 111000$ (valor 56) poderia ser mantido na nova geração por ser um dos melhores da população anterior.

A nova população pode então ser formada pelos dois novos indivíduos $BD = 101001$ e $DB = 111101$, o melhor da geração anterior $D = 111000$ e um novo indivíduo gerado aleatoriamente, como $E = 011111$ (valor 31). O ciclo se repete: avaliação, seleção, crossover, possível mutação e elitismo — formando novas gerações até que uma condição de parada seja atingida, como um número máximo de iterações ou a estagnação da melhora das soluções. A Figura 18 apresenta de forma mais clara o fluxo passo-a-passo de um GA.

Figura 18 – Fluxograma do GA.



Fonte: autoria própria.

2.5.1 Algoritmos genéticos com chaves aleatórias enviesadas

O Algoritmo Genético de Chaves Aleatórias (do inglês, *Random-Key Genetic Algorithm (RKGA)*) é uma variação dos algoritmos genéticos tradicionais abordado na sessão anterior, proposto por Bean (Bean, 1994) com o objetivo de lidar com problemas de otimização combinatória onde as soluções podem ser representadas como permutações. Em vez de trabalhar diretamente com estruturas discretas, o *algoritmo genético de chaves aleatórias* (RKGA) utiliza vetores de números reais no intervalo [0,1), chamados de *chaves aleatórias*, para codificar as soluções. Um *decodificador* determinístico interpreta esses vetores, gerando uma solução para o problema, que pode ser viável ou não, que é então avaliada de acordo com a função objetivo (Resende, 2013). A implementação deste decodificador é específica do problema e deve ser feita por quem está modelando a solução.

O RKGA evolui uma população de tamanho fixo ao longo de diversas gerações. Em cada iteração, a população é dividida em dois subconjuntos: o conjunto *elite*, que contém os melhores indivíduos (com base na avaliação do decodificador), e o conjunto *não-elite*, com os demais. Todos os indivíduos do conjunto elite são copiados diretamente para a próxima geração,

caracterizando o elitismo do algoritmo. Além disso, um subconjunto da próxima população é preenchido com *mutantes* — vetores totalmente novos gerados aleatoriamente, com o objetivo de preservar a diversidade. O restante da população é preenchido por descendentes gerados a partir do cruzamento de pares de indivíduos da população atual, utilizando o cruzamento uniforme parametrizado definido a seguir. Para cada posição i do vetor de chaves, o filho $c[i]$ recebe a chave $a[i]$ do pai a com probabilidade p_a ou a chave $b[i]$ do pai b com probabilidade $p_b = 1 - p_a$. Ambos os pais (a e b) são escolhidos aleatoriamente da população atual (Spears; Jong, 1991).

O *algoritmo genético de chaves aleatórias enviesadas* (BRKGA) é uma extensão do RKGA que introduz um viés sistemático no processo de seleção dos pais e no cruzamento, com o objetivo de acelerar a convergência sem comprometer a diversidade da população (Goncalves; Resende, 2011). No BRKGA, a população é dividida da mesma forma que no RKGA, em três grupos:

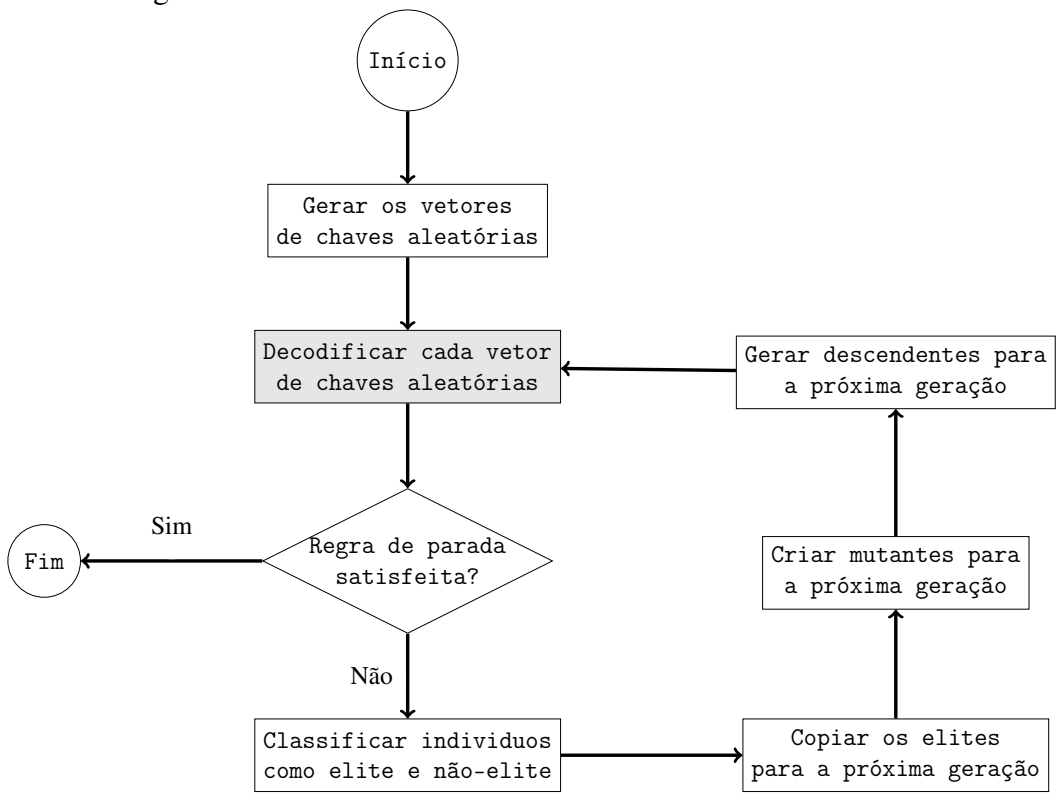
- $Elite$: as melhores soluções atuais, que são copiadas diretamente para a próxima geração;
- $Mutantes$: soluções aleatórias criadas a cada geração, para manter diversidade;
- $Filhos$: gerados por cruzamento entre um pai do conjunto elite e outro do conjunto não-elite.

A diferença do BRKGA em relação ao RKGA está na seleção dos pais e no cruzamento. No BRKGA, um dos pais é sempre selecionado do conjunto elite, enquanto o outro vem do conjunto não-elite. Isso garante que a influência das melhores soluções seja constante e significativa na geração de novos indivíduos. Para cada posição i do vetor de chaves, é lançada uma "moeda viciada" com probabilidade $p > 0.5$ de selecionar o valor da chave do pai elite. Caso contrário, a chave é herdada do pai não-elite. Essa estratégia aumenta a chance de herdar bons genes do pai elite, enquanto a probabilidade de herdar do pai não-elite ainda permite variabilidade, balanceando exploração e explotação da população (Resende, 2013).

Além disso, o BRKGA pode empregar estratégias de *reinicialização*. Caso a melhor solução da população não melhore após um número predefinido de gerações, a população pode ser reinicializada parcial ou totalmente (com a exceção dos indivíduos elite que são preservados), evitando a estagnação em ótimos locais. Outra extensão importante é o uso de *múltiplas populações* evoluindo em paralelo. Essas populações podem, periodicamente, trocar informações, substituindo as piores soluções de uma população pelas melhores de outra, o que ajuda a intensificar a exploração e diversificação da busca (Resende, 2013). Essa estrutura torna o BRKGA uma metaheurística robusta, eficiente e de fácil adaptação a uma grande variedade

de problemas de otimização combinatória e contínua. Corroborando essa visão, um *survey* publicado em 2025 sobre o BRKGA destaca sua versatilidade e eficácia em diferentes domínios, com desempenho consistentemente igual ou superior ao de outros métodos, especialmente em problemas envolvendo grafos (Londe *et al.*, 2025). Em muitos contextos, o BRKGA tem apresentado desempenho superior ao do RKGA, o que se deve à forma como os pais são selecionados para cruzamento e como o cruzamento é implementado (Resende, 2013). A Figura 19 apresenta de forma mais clara o fluxo passo-a-passo de um BRKGA.

Figura 19 – Fluxograma do BRKGA.



Fonte: autoria própria.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os principais estudos relacionados ao tema em questão, destacando as contribuições de cada um e analisando suas convergências e divergências em relação à proposta deste trabalho.

3.1 Artigo: Perfect Roman domination in trees

O artigo *Perfect Roman domination in trees* (Henning *et al.*, 2017) é o primeiro trabalho que propõe formalmente o conceito de dominação romana perfeita em grafos. Essa definição é uma variação mais restritiva da dominação romana clássica.

O trabalho de Henning et al. define formalmente uma função de dominação romana perfeita e concentra-se no estudo dessa função para a classe das árvores, provando que, para qualquer árvore T com $n \geq 3$ vértices, vale a desigualdade $\gamma_R^p(T) \leq \frac{4}{5}n$. Os autores também caracterizam exatamente as árvores que atingem a igualdade, as quais pertencem a uma classe especial denominada τ, formada por árvores compostas por caminhos de cinco vértices interligados por seus vértices centrais.

As contribuições do trabalho de Henning et al. (Henning *et al.*, 2017) são de grande relevância para este trabalho, uma vez que fornecem os conceitos formais e os principais resultados teóricos sobre a dominação romana perfeita – parâmetro central do problema aqui investigado. A relação entre os trabalhos está na busca por soluções para o problema de minimizar $\gamma_R^p(G)$, ainda que por caminhos distintos: enquanto Henning et al. (Henning *et al.*, 2017) utilizam técnicas analíticas e combinatórias para caracterizar famílias específicas de árvores e obter limites teóricos exatos, este trabalho propõe o uso de metaheurísticas para abordar o problema de forma geral, inclusive para instâncias nas quais os algoritmos de programação inteira/mista não são computacionalmente viáveis. Dessa forma, o presente estudo se apoia nas definições e teoremas estabelecidos no artigo como base teórica e utiliza suas estruturas como casos de teste para a avaliação empírica das soluções geradas.

3.2 Artigo: On the Computational Complexity Aspects of Perfect Roman Domination

O artigo *On the Computational Complexity Aspects of Perfect Roman Domination* (Mirhoseini; Rad, 2023) trata de aspectos teóricos e computacionais do problema de dominação romana perfeita. Nele, os autores discutem a complexidade do problema, provando

que ele é NP-completo em determinadas classes de grafos, como os bipartidos estrela-convexos, e APX-hard mesmo em grafos com grau máximo igual a 4. Além disso, apresentam uma formulação exata do problema por meio de Programação Inteira (PI), utilizando variáveis binárias que definem o valor atribuído a cada vértice (0, 1 ou 2), com o objetivo de minimizar o peso total da função objetivo, respeitando a restrição de que todo vértice com rótulo 0 tenha exatamente um vizinho com rótulo 2.

Este artigo se relaciona com o presente trabalho por tratar diretamente do mesmo problema e por propor um algoritmo exato para o problema, no caso, um modelo de programação linear inteira. No âmbito deste trabalho, a formulação inteira será implementada com o objetivo de servir como comparação de desempenho frente às soluções obtidas por meio de metaheurísticas, funcionando como um benchmark exato em instâncias de menor porte.

A principal diferença entre os trabalhos está no foco metodológico: enquanto Mirhoseini e Jafari Rad concentram-se na análise da complexidade teórica do problema e em sua modelagem formal, o presente trabalho tem como objetivo principal o desenvolvimento de soluções aproximadas com base em metaheurísticas, utilizando a formulação PI apenas como referência para avaliação de desempenho.

3.3 Artigo: On Roman Domination of Graphs Using a Genetic Algorithm

O artigo *On Roman Domination of Graphs Using a Genetic Algorithm* (Khandelwal; Saran, 2021) apresenta uma abordagem baseada em GA para resolver o problema de dominação romana em grafos. A proposta do artigo se destaca por ser uma das primeiras a aplicar uma metaheurística ao problema, considerando instâncias de grafos gerais.

O algoritmo genético desenvolvido pelos autores é composto por duas heurísticas de construção para a geração da população inicial, operadores de *crossover* específicos ao problema e um procedimento de correção para garantir a viabilidade das soluções. Além disso, os autores adotam uma estratégia de seleção por torneio binário e um critério de parada baseado na estagnação do custo por 100 gerações consecutivas ou no limite de 1000 gerações. Os experimentos foram conduzidos sobre diferentes classes de grafos com soluções conhecidas — como caminhos, ciclos, grafos estrela e grades $2 \times n$ — e também sobre 120 instâncias da coleção Harwell–Boeing. Os resultados mostram que o algoritmo atinge os valores ótimos em todas as instâncias com solução conhecida e apresenta bom desempenho nas demais, com valores dentro dos limites superiores e inferiores publicados.

Este trabalho de conclusão de curso propõe o uso de algoritmos genéticos para o problema de dominação romana perfeita. Embora o artigo não trate dessa versão específica, sua contribuição é relevante para o presente trabalho, pois fornece estratégias para construção de população inicial, operadores genéticos adaptados à estrutura do problema e mecanismos de correção viável, que podem ser modificados para atender às exigências da dominação romana perfeita. Além disso, este trabalho de conclusão de curso também estende a abordagem original ao implementar uma variação do BRKGA, em conjunto com uma formulação de PI, com o objetivo de aumentar a eficiência e a qualidade das soluções em instâncias de maior porte, além de fornecer uma referência exata para comparação dos resultados obtidos.

3.4 Artigo: Note on the Perfect Roman Domination Number of Graphs

O artigo *Note on the Perfect Roman Domination Number of Graphs* (Yue; Song, 2020) apresenta contribuições teóricas relacionadas ao número de dominação romana perfeita (PRD) em grafos. Os autores discutem diversas propriedades do parâmetro, estabelecendo limites superiores e inferiores, além de caracterizar classes de grafos com valores específicos para o número de dominação romana perfeita. Um dos principais resultados é um algoritmo de tempo linear para calcular esse número em *cografos* — grafos que não contêm o caminho com quatro vértices (P_4) como subgrafo induzido. A abordagem se baseia na estrutura recursiva dos cografos e utiliza operações de união e junção com base na árvore de decomposição (cotree) do grafo.

O presente trabalho se relaciona com esse artigo ao tratar diretamente do mesmo parâmetro e ao empregar esse conceito em contexto algorítmico. Além disso, a caracterização teórica oferecida pelos autores serve de base para o entendimento estrutural do problema, o que pode ser útil para a construção de instâncias de teste ou para análises de comportamento das soluções em grafos específicos. A principal diferença em relação a este trabalho está no enfoque metodológico: enquanto Yue e Song desenvolvem resultados puramente teóricos e um algoritmo exato restrito a uma classe específica de grafos, o presente trabalho busca propor metaheurísticas aplicadas ao problema de dominação romana perfeita em grafos gerais. Além disso, este trabalho também prevê a implementação de uma formulação de PI para o problema, com o objetivo de servir como benchmark para avaliação das soluções aproximadas, abordagem não considerada no artigo em questão.

4 NOVA PROPOSTA DE PROBLEMA DE PROGRAMAÇÃO INTEIRA

Ao analisarmos o modelo de PI de Mirhoseini e Rad (2023), apresentado na Seção 2.4, vimos que as restrições (2.2b) tinham um pequeno erro, onde a variável c_v deveria ser multiplicada por 2, o que pode ter sido um erro de digitação na hora da composição do artigo deles. Porém, também identificamos que a variável binária a_v é desnecessária, podendo ser descartada do modelo. Essa simplificação reduz o número total de variáveis binárias no modelo, contribuindo para uma formulação mais enxuta e, possivelmente, mais eficiente em termos de resolução. Logo, podemos reescrever o modelo como segue.

Dado um grafo $G = (V(G), E(G))$, nós criamos duas variáveis binárias x_v e y_v para todo vértice $v \in V(G)$ cujos valores são definidos abaixo.

$$x_v = \begin{cases} 1, & \text{se } f(v) = 1; \\ 0, & \text{caso contrário.} \end{cases} \qquad y_v = \begin{cases} 1, & \text{se } f(v) = 2; \\ 0, & \text{caso contrário.} \end{cases}$$

Uma nova formulação matemática é dada a seguir.

$$\text{minimize} \quad \sum_{v \in V(G)} (x_v + 2y_v) \tag{4.1a}$$

$$\text{subject to} \quad 2x_v + 2y_v + (1 - x_v)(1 - y_v) \sum_{u \in N(v)} 2y_u = 2 \quad \forall v \in V(G), \tag{4.1b}$$

$$x_v + y_v \leq 1 \qquad \forall v \in V(G), \tag{4.1c}$$

$$x_v, y_v \in \{0, 1\} \qquad \forall v \in V(G) \tag{4.1d}$$

De forma análoga ao modelo de Mirhoseini e Rad (2023), a função objetivo (4.1a) minimiza o peso de uma FDRP f de G. As restrições (4.1b) garantem que todo vértice v desprotegido, isto é, com $f(v) = 0$ seja adjacente a exatamente um vértice u de modo que $f(u) = 2$. As restrições (4.1c) garantem que todo vértice tenha exatamente 1 rótulo pertencente ao conjunto $\{0, 1, 2\}$. Observe que, pela definição das variáveis, se $x_v = 0$ e $y_v = 0$, isso significa que, para esse vértice v, seu rótulo é 0, ou seja, $f(v) = 0$. Por fim, as restrições (4.1d) garantem que as variáveis sejam binárias. Esse modelo tem $2|V(G)|$ variáveis e $2|V(G)|$ restrições (desconsiderando as restrições de integralidade).

5 ALGORITMO GENÉTICO

Este capítulo descreve o GA implementado neste trabalho, especificando os parâmetros de entrada do algoritmo, a codificação da solução (representação como um cromossomo), as estratégias de geração da população inicial e as sub-rotinas essenciais para o funcionamento do algoritmo.

5.1 Parâmetros do Algoritmo Genético

O algoritmo genético possui sete parâmetros de entrada que controlam seu comportamento e que são listados abaixo:

- `G`: grafo que representa a instância do problema a ser resolvida.
- `popFactor`: parâmetro inteiro positivo responsável por determinar o tamanho da população, isto é, o número de cromossomos (soluções) em cada geração. Neste trabalho, o GA utiliza um tamanho de população constante ao longo das gerações, conforme a prática mais comum na literatura. O tamanho da população é definido em função da ordem do grafo, sendo calculado como $|V(G)| \div popFactor$.
- `tournSize`: parâmetro inteiro positivo que representa o número de cromossomos que competem em cada torneio durante a seleção.
- `stagnant`: parâmetro inteiro positivo que indica o número máximo de iterações consecutivas sem melhoria na qualidade das soluções encontradas.
- `mutRate`: parâmetro com domínio no intervalo real [0, 1] que indica a probabilidade de um gene sofrer mutação.
- `eliSize`: parâmetro com domínio no intervalo real [0, 1] que indica quantidade dos melhores cromossomos que são diretamente preservados para a próxima geração.
- `maxGenerations`: parâmetro inteiro positivo que indica o número máximo de iterações que o algoritmo pode executar.

Como saída, o algoritmo produz o resultado do experimento, contemplando o nome do grafo, sua ordem, tamanho e densidade, bem como o melhor valor de aptidão (*fitness*) alcançado e o tempo total de execução, medido em segundos.

5.2 Representação da Solução

Dado um grafo G com conjunto de vértices $V(G) = \{v_0, \ldots, v_{n-1}\}$, uma solução (*solution*) para o problema de dominação romana perfeita é representada por uma estrutura composta por dois componentes principais: um vetor de inteiros chamado `solution` e um valor inteiro chamada `fitness`.

O vetor `solution[0,...,n-1]` armazena os rótulos atribuídos aos vértices do grafo, de forma que a posição i deste vetor corresponde ao rótulo do vértice v_i do grafo de entrada G. O valor armazenado nessa posição, que pode assumir os valores 0, 1 ou 2, indica o rótulo atribuído ao vértice v_i, respeitando as restrições impostas por uma função de dominação romana perfeita.

O atributo `fitness` armazena o valor de aptidão da solução, o qual expressa sua qualidade e é calculado como a soma dos rótulos atribuídos aos vértices do grafo, ou seja, $\texttt{fitness} = \sum_{i=0}^{n-1} \texttt{solution}[i]$. A Figura 20 demonstra um grafo G rotulado com uma FDRP e o vetor `solution` correspondente a essa solução.

Figura 20 – Representação visual do grafo e do cromossomo contendo uma rotulação deste grafo.

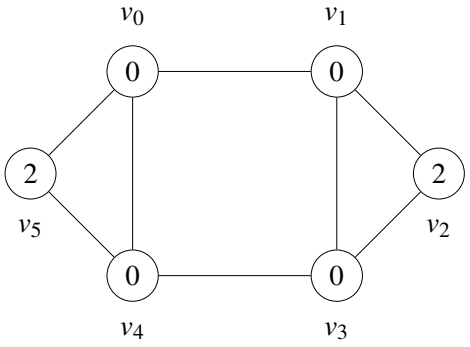


Figura 21 – Exemplo de grafo rotulado.

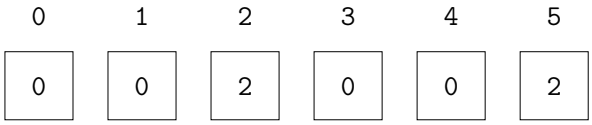


Figura 22 – Representação do cromossomo: um vetor de tamanho 6, no qual cada posição armazena o rótulo atribuído ao respectivo vértice do grafo. Observa-se que, neste caso, o atributo `fitness` assume valor igual a 4, correspondente à soma dos rótulos atribuídos aos vértices na solução representada.
Fonte: Elaborado pelo autor.

5.3 Visão geral do algoritmo genético

O GA inicia com a geração da população inicial, a qual é construída a partir da combinação de uma heurística gulosa e de um gerador aleatório de cromossomos, que são abordados na Seção 5.5. Cada cromossomo representa uma solução candidata para o problema de dominação romana perfeita. Destaca-se que todos os indivíduos, inclusive aqueles gerados de forma aleatória, são soluções viáveis para o grafo dado como entrada, uma vez que, independentemente do método de geração empregado, todos os cromossomos são submetidos a um procedimento de correção, o qual assegura que todas as restrições do problema são satisfeitas.

A cada geração, o algoritmo executa as seguintes etapas, nesta ordem: inicialmente, aplica-se um processo de seleção, no qual pares de cromossomos são escolhidos a partir da população corrente para participação no cruzamento. Em seguida, entra em ação o operador de cruzamento, responsável pela geração de uma nova população. Sobre a prole obtida através do processo de cruzamento, aplica-se o operador de mutação, com o objetivo de promover diversidade genética. Posteriormente, adota-se a estratégia de elitismo, na qual os melhores indivíduos da população atual são preservados e incorporados à nova população.

Ressalta-se que o valor de *fitness* de cada solução é calculado no momento de sua criação, logo após o procedimento de correção, não sendo, portanto, necessária uma etapa adicional específica para a avaliação de aptidão.

Ao término de cada iteração, é garantido que a população seja composta exclusivamente por soluções viáveis. O algoritmo é finalizado quando o número máximo de gerações é atingido; ou quando é alcançado o limite máximo de gerações consecutivas sem melhoria, caracterizando a ausência de progresso significativo na qualidade das soluções obtidas; ou quando o tempo limite de 900 segundos (15 minutos) é atingindo.

O Algoritmo 1 apresenta o pseudocódigo do laço principal do GA proposto. Ao término de sua execução, o algoritmo retorna uma estrutura contendo as principais informações do grafo, bem como o melhor valor de *fitness* obtido e o tempo total de execução, medido em segundos.

Algoritmo 1: gaFlow

Entrada: G, popFactor, tournSize, stagnant, mutRate, eliSize, maxGenerations

Resultado do experimento com o nome do grafo, ordem, tamanho, densidade, melhor fitness e tempo gasto

início

- $currentPopulation \leftarrow initializePopulation(popFactor)$
- $gen \leftarrow 0$
- $i \leftarrow 0$
- $result \leftarrow (\text{nome do grafo}, \text{ordem}, \text{tamanho}, \text{densidade}, +\infty, 0)$
- **while** $i < maxGenerations$ **and** $gen < stagnant$ **do**
  - **if** *tempo de processamento maior ou igual que 15 minutos* **then**
    - $result.elapsed\_time \leftarrow 900$
    - **return** result
  - **end**
  - $select \leftarrow selection(currentPop, tournSize)$
  - $newPop \leftarrow crossover(select)$
  - $mutation(newPop, mutRate)$
  - $currentPop \leftarrow elitism(currentPop, newPop, eliSize)$
  - **if** $currentPop.bestFitness < result.fitness$ **then**
    - $result.fitness \leftarrow currentPop.bestFitness$
    - $gen \leftarrow 0$
  - **end**
  - **else**
    - $gen \leftarrow gen + 1$
  - **end**
  - $i \leftarrow i + 1$
- **end**
- $result.elapsed\_time \leftarrow tempo\ total$
- **return** result

fim

5.4 Procedimentos auxiliares

Esta seção apresenta os procedimentos auxiliares empregados nas estratégias de geração de cromossomos. Tais procedimentos têm como principal objetivo assegurar que todas as soluções geradas ao longo do algoritmo, ainda que eventualmente inviáveis em sua forma

original, sejam submetidas a um mecanismo de correção. Dessa forma, garante-se que as soluções consideradas nas etapas subsequentes do processo evolutivo sejam sempre corretas e viáveis em relação às restrições do problema.

5.4.1 Checagem e correção de viabilidade

Como o gerador aleatório utilizado na inicialização da população não garante, por si só, a viabilidade das soluções geradas — o mesmo ocorrendo com os operadores de mutação e cruzamento —, torna-se necessário avaliar cada novo indivíduo após sua criação e submetê-lo a um procedimento de correção. O procedimento `fixSolution` é responsável por verificar se os valores dos genes de um cromossomo, isto é, os valores atribuídos a cada posição do vetor de rótulos (cujos possíveis valores são 0, 1 ou 2), estão em conformidade com as restrições impostas pelo PDRP.

Caso algum gene viole essas restrições, seu valor é ajustado de modo a restabelecer a viabilidade da solução. Esse processo consiste em inspecionar a vizinhança de cada vértice do grafo e, com base nos rótulos atribuídos a seus vizinhos, redefinir os valores dos genes para 0, 1 ou 2 sempre que necessário, garantindo que a solução final satisfaça todas as restrições do problema. A Figura 23 ilustra de forma visual o processo, enquanto o Algoritmo 2 detalha seu passo a passo.

5.4.1.1 Correções para genes com valor 0

Quando um gene assume o valor 0, o algoritmo verifica se ele satisfaz as condições do PDRP com base na rotulação de seus vértices vizinhos. Para isso, contabiliza-se o número de vizinhos com rótulo 2. Caso exista exatamente um vizinho com rótulo 2, o gene é considerado válido e nenhuma modificação é necessária. Caso contrário, o valor do gene é ajustado conforme os seguintes critérios:

- Se houver mais de um vizinho com rótulo 2 em sua vizinhança, o gene é ajustado para o rótulo 1, uma vez que um vértice rotulado com 1 satisfaz as restrições do PDRP sem comprometer a viabilidade global da solução.
- Se não houver nenhum vizinho com rótulo 2 em sua vizinhança, a vizinhança é analisada em busca de um vértice com rótulo 0 que esteja devidamente dominado. Caso tal vértice exista, o gene é ajustado para o rótulo 1; caso contrário, o gene recebe o rótulo 2, passando a atuar como um dominador local.

Figura 23 – Aplicando a checagem e correção de viabilidade no cromossomo para corrigir rótulos que não satisfazem o PDRP.

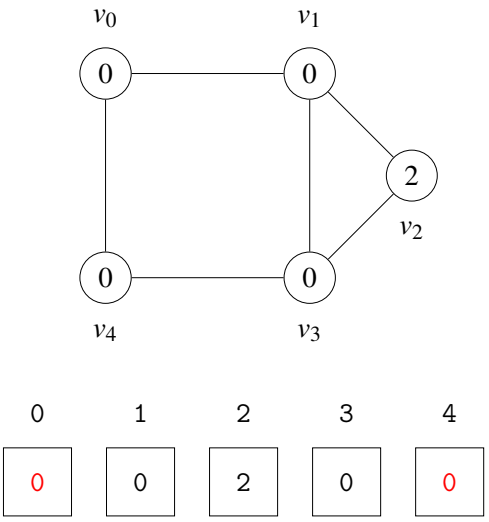


Figura 24 – Grafo antes da verificação de viabilidade. Observa-se que os vértices v_0 e v_4 não satisfazem as restrições da FDRP, uma vez que, para atender às condições do problema, cada vértice com rótulo 0 deve possuir exatamente um vértice adjacente rotulado com 2 ou assumir o rótulo 1.

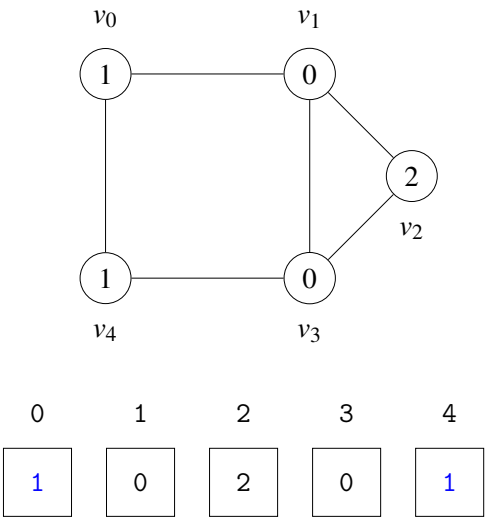


Figura 25 – Grafo após a checagem e correção de viabilidade. Observe que os vértices v_0 e v_4 agora cumprem às restrições de uma FDRP.
Fonte: autoria própria.

Algoritmo 2: fixSolution

Entrada: grafo G, cromossomo $f[0,\ldots n-1]$

Cromossomo válido segundo o PDRP

início

- **for** *todo vértice* $v \in V(G)$ *tal que* $f[v] = 0$ **do**
  - **if** *v tem exatamente um vizinho com rótulo* 2 **then**
    - **continue**
  - **end**
  - **if** *v tem dois ou mais vizinhos com rótulo* 2 **then**
    - $f[v] \leftarrow 1$
  - **end**
  - **else**
    - **if** *existe* $u \in N(v)$ *tal que* $f[u] = 0$ *e u é dominado* **then**
      - $f[v] \leftarrow 1$
    - **end**
    - **else**
      - $f[v] \leftarrow 2$
    - **end**
  - **end**
- **end**

fim

5.4.2 Minimizador de peso (reduceWeight)

A função `reduceWeight` é aplicada após a rotina de correção de viabilidade e tem como objetivo reduzir o peso total de uma solução viável, minimizando os rótulos atribuídos aos vértices sempre que possível, sem violar as restrições do problema. Inicialmente, o procedimento percorre todos os vértices rotulados com valor 2 e verifica se cada um deles é estritamente necessário para a dominância da solução. Um vértice com rótulo 2 é considerado dispensável caso não seja o único responsável por dominar todos os vértices vizinhos rotulados com 0. Nessas situações, seu rótulo é reduzido para 1.

Em seguida, a função analisa os vértices rotulados com valor 1 e verifica se eles são dominados por exatamente um vértice com rótulo 2. Quando essa condição é satisfeita, o rótulo do vértice pode ser reduzido para 0, uma vez que sua dominância permanece garantida. Dessa forma, o procedimento tenta realizar uma redução progressiva dos rótulos dos vértices,

assegurando a viabilidade da solução final. O Algoritmo 3 detalha o funcionamento da função.

Algoritmo 3: reduceWeight

Entrada: grafo G, cromossomo $f[0,\ldots n-1]$

Cromossomo com um possível peso menor.

início

  for *todo vértice* $v \in V(G)$ *tal que* $f[v] = 2$ **do**

    if *não existe* $u \in N(v)$ *tal que* $f[u] = 0$ *e u seja dominado exclusivamente por v* **then**

      $f[v] \leftarrow 1$

    end

  end

  for *todo vértice* $v \in V(G)$ *tal que* $f[v] = 1$ **do**

    if *existe exatamente um vértice* $u \in N(v)$ *tal que* $f[u] = 2$ **then**

      $f[v] \leftarrow 0$

    end

  end

fim

5.5 Heurística e gerador aleatório

Esta seção apresenta a heurística gulosa e o gerador aleatório empregados na geração da população inicial do GA.

Heurística de geração de cromossomo (solução viável): A heurística inicia inserindo todos os vértices do grafo em uma fila de prioridade ordenada de forma decrescente pelo grau, de modo que vértices com maior número de vizinhos sejam processados prioritariamente.

Enquanto a fila de prioridade não estiver vazia, o vértice de maior prioridade é removido e analisado. Caso esse vértice já possua rótulo definido, ele é desconsiderado. Se o vértice for isolado, isto é, possuir grau zero, ele é rotulado com valor 1.

Quando o vértice não é isolado, verifica-se a possibilidade de atribuição do rótulo 2. Para isso, sua vizinhança é analisada a fim de garantir que nenhum de seus vértices adjacentes esteja rotulado com 1 ou 2, nem que esteja dominado mantendo rótulo 0. Caso essa condição seja satisfeita e o vértice ainda não esteja dominado, ele recebe o rótulo 2. Em seguida, todos os seus vizinhos que ainda não possuam rótulo definido recebem o rótulo 0.

Se o vértice não puder receber o rótulo 2 e ainda não estiver dominado, ele é rotulado com valor 1. Por fim, caso o vértice já esteja dominado e ainda não possua rótulo definido, ele recebe o rótulo 0, assegurando a consistência da rotulação.

Ao término do processamento de todos os vértices, os rótulos atribuídos são utilizados para construir e retornar uma única solução viável, correspondente a uma configuração obtida por meio de um procedimento guloso. Para garantir a correção de inconsistências a sub-rotina `fixSolution` é chamada, como mostra o Algoritmo 4.

Algoritmo 4: greedyInitialization

Entrada: grafo G

Solução viável inicial **início**
  Seja Q uma fila de prioridade de máximo contendo todos os vértices de G, em que a chave é o grau do vértice.
  Defina um vetor de rótulos $f = [-1, -1, \ldots, -1]$ de tamanho $|V(G)|$
  while *Q não estiver vazia* **do**
    Remover o vértice v de Q com maior prioridade (maior grau)
    if $f[v] \neq -1$ **then**
      continue
    end
    if *v é um vértice isolado* **then**
      $f[v] \leftarrow 1$
      continue
    end
    if *v não está dominado* $\wedge \nexists u \in N(v)$ *tal que* $\big[(f[u] = 0 \wedge u \text{ está dominado}) \vee f[u] = 1 \vee f[u] = 2\big]$ **then**
      $f[v] \leftarrow 2$
      for *cada* $u \in N(v)$ **do**
        if $f[u] = -1$ **then**
          $f[u] \leftarrow 0$
        end
      end
    end
    else if *v não está dominado* **then**
      $f[v] \leftarrow 1$
    end
    else
      $f[v] \leftarrow 0$
    end
  end
  fixSolution(f)
  return f
fim

Gerador aleatório: O gerador atribui, de forma aleatória, rótulos 0, 1 ou 2 a cada vértice do grafo. Em seguida, a sub-rotina `fixSolution` é aplicada com o objetivo de corrigir eventuais inconsistências decorrentes dessa atribuição aleatória, assegurando que a solução resultante seja viável em relação às restrições do problema. O Algoritmo 5 detalha de forma clara o funcionamento da função.

Algoritmo 5: randomSolution

Entrada: grafo G

Solução viável inicial.
início
    Defina um vetor de rótulos $f = [-1, -1, \ldots, -1]$ de tamanho $|V(G)|$
    for *todo vértice* $v \in V(G)$ **do**
        sortear $n \in \{0, 1, 2\}$ uniformemente ao acaso
        $f[v] \leftarrow n$
    end
    fixSolution(f)
    return f
fim

5.6 Operação de Seleção

O operador de seleção implementado utiliza a técnica clássica de seleção por torneio (Goldberg; Deb, 1991). A função `tournamentSelection` adota esse mecanismo com o objetivo de selecionar pares de soluções que serão utilizados na etapa de *crossover* do algoritmo genético. O procedimento executa um número de iterações proporcional ao tamanho da população, selecionando aproximadamente metade desse conjunto na forma de pares.

Em cada iteração, uma cópia auxiliar da população é criada e embaralhada aleatoriamente, garantindo diversidade no processo de seleção. A partir dessa população embaralhada, é definido um subconjunto de tamanho fixo de indivíduos, definido pelo parâmetro `tournSize`, que participam de cada torneio. Em seguida, duas soluções são selecionadas por meio de torneios independentes, nos quais a solução com menor valor de *fitness* é escolhida entre as candidatas. Essas soluções são então agrupadas para formar um par de indivíduos selecionados para reprodução.

Ao final do processo, a função retorna um conjunto de pares de soluções, representando os indivíduos escolhidos para participar das operações de cruzamento do algoritmo

genético.

5.7 Operação de Cruzamento

O operador de cruzamento é responsável por combinar duas soluções S_1 e S_2 pertencentes à população atual de cromossomos, com o objetivo de gerar novas soluções descendentes. Utilizamos o clássico operador de cruzamento de um ponto (do inglês, *one-point crossover* (Goldberg, 1989)). O procedimento é realizado conforme descrito a seguir:

1. **Determinação do ponto de cruzamento:** Um índice aleatório $I \in \{0, \ldots, n-1\}$ é selecionado ao longo do comprimento do cromossomo, definindo o ponto em que ocorrerá o cruzamento.
2. **Troca de genes:** A partir do ponto I, são gerados dois descendentes da seguinte forma: o primeiro descendente é construído concatenando a subsequência $S_1[0, \ldots, I-1]$ com a subsequência $S_2[I, \ldots, n-1]$, enquanto o segundo descendente é formado concatenando a subsequência $S_2[0, \ldots, I-1]$ com a subsequência $S_1[I, \ldots, n-1]$.
3. **Garantia de viabilidade:** As soluções geradas são submetidas à sub-rotina `fixSolution`, responsável por corrigir eventuais inconsistências e assegurar que as soluções resultantes sejam viáveis.

5.8 Operação de Mutação

O operador de mutação implementado realiza alterações aleatórias nos genes das soluções da população, com o objetivo de introduzir diversidade genética e evitar a convergência prematura do algoritmo genético.

Para cada solução da população, o algoritmo percorre todos os seus genes individualmente. Em cada posição do cromossomo, uma mutação pode ocorrer com probabilidade igual à taxa de mutação previamente definida. Quando uma mutação é acionada, o gene correspondente tem seu valor substituído por um novo rótulo escolhido aleatoriamente entre os valores possíveis do conjunto $\{0, 2\}$

Caso ao menos um gene da solução seja alterado durante esse processo, a solução resultante é submetida à sub-rotina `fixSolution`, garantindo sua viabilidade, e tem seu valor de *fitness* recalculado. O Algoritmo 6 aborda de forma mais clara o procedimento.

Algoritmo 6: randomMutation

Entrada: População P, taxa de mutação `mutRate`

```
1  População possivelmente mutada
2  início
3      for cada solução S ∈ P do
4          alterado ← falso
5          for cada gene g_i ∈ S do
6              Seja n um número real sorteado uniformemente no intervalo [0,1]
7              if n < mutRate then
8                  Seja r um número inteiro sorteado uniformemente no conjunto {0,2}
                   g_i ← r
9                  alterado ← verdadeiro
10             end
11         end
12         if alterado then
13             fixSolution(S)
14             Recalcular o valor de fitness de S
15         end
16     end
17     return P
18 fim
```

5.9 Operação de Elitismo

O operador de elitismo implementado tem como objetivo preservar as melhores soluções da população atual ao longo das gerações, garantindo que indivíduos de alta qualidade não sejam perdidos durante o processo evolutivo. Para isso, a função recebe como entrada a população corrente, a nova população gerada após as operações genéticas e a taxa de elitismo.

Inicialmente, ambas as populações são ordenadas de forma crescente de acordo com o valor de *fitness*. Em seguida, os melhores indivíduos da população atual — em quantidade definida pelo valor do parâmetro `eliSize` — são diretamente copiados para a população resultante. As demais soluções da população atual são descartadas.

Na sequência, a população resultante é completada com as melhores soluções provenientes da nova população. As soluções excedentes da nova população também são removidas.

Por fim, a população resultante é novamente ordenada pelo valor de *fitness*, assegu-

rando que a melhor solução esteja posicionada na primeira posição, e então retornada como a população da próxima geração. O Algoritmo 7 detalha o funcionamento do procedimento.

Algoritmo 7: defaultElitism

Entrada: População atual P, nova população P', $eliSize$

População da próxima geração

início

    Ordenar P em ordem crescente de *fitness*

    Ordenar P' em ordem crescente de *fitness*

    Seja $R \leftarrow \emptyset$

    for $i \leftarrow 1$ **to** $eliSize$ **do**

        Inserir $P[i]$ em R

    end

    for $i \leftarrow 1$ **to** $|P'| - eliSize$ **do**

        Inserir $P'[i]$ em R

    end

    Ordenar R em ordem crescente de *fitness*

    return R

fim

6 ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS ENVIESADAS

O algoritmo BRKGA foi apresentado e detalhadamente discutido no Capítulo 2, no qual foram descritos seus principais componentes, bem como as etapas fundamentais que compõem seu funcionamento, tais como inicialização da população, seleção, cruzamento enviesado e atualização geracional. Pela própria definição do algoritmo, os operadores utilizados nestas etapas são fixos e pré-definidos.

Dessa forma, neste capítulo, assume-se que a estrutura geral do BRKGA já é conhecida, assim como os seus operados pré-fixados, o foco passa a ser seus elementos variáveis: os parâmetros de entrada que o algoritmo possui e a sua função decodificadora (*decoder*). Neste trabalho, utilizamos como ponto de partida a API do BRKGA proposta por Toso e Resende (2014), que fornece uma interface em `C++` para a implementação eficiente de algoritmos BRKGA. Portanto, a fim de completar a proposta de algoritmo BRKGA para o PDRP, neste capítulo, revisamos os parâmetros de entrada do algoritmo e descrevemos como as soluções codificadas em vetores de chaves aleatórias são transformadas em soluções viáveis para o PDRP.

O *decoder* desempenha um papel fundamental no BRKGA, pois é responsável por mapear cada cromossomo — representado por um vetor de valores reais no intervalo $[0, 1)$ — para uma solução factível do PDRP, além de avaliar sua qualidade por meio da função objetivo. Assim, a correta definição e implementação do *decoder* é determinante para o desempenho do algoritmo, uma vez que ele encapsula todo o conhecimento específico do problema dentro da estrutura genérica do BRKGA.

Nos tópicos a seguir, são apresentados os parâmetros do BRKGA; além disso, o funcionamento do *decoder* é descrito em detalhes, bem como sua integração com a API, destacando-se os principais aspectos de sua implementação e seu impacto na qualidade das soluções obtidas.

6.1 Parâmetros do BRKGA

O algoritmo possui onze parâmetros de entrada, descritos a seguir.

- `G`: Grafo que representa a instância do problema a ser resolvida.
- `population_factor`: parâmetro inteiro positivo responsável por determinar o tamanho da população, isto é, o número de cromossomos (soluções) em cada geração. Neste trabalho, o BRKGA utiliza um tamanho de população constante ao longo das gerações,

conforme a prática mais comum na literatura. O tamanho da população é definido em função da ordem do grafo, sendo calculado como $|V(G)| \div population\_factor$.

- `pe`: parâmetro com domínio no intervalo real [0, 1] que define o percentual da população que deve ser classificada como elite.
- `pm`: parâmetro com domínio no intervalo real [0, 1] que define o percentual da população que deve ser substituída por mutantes.
- `rhoe`: parâmetro com domínio no intervalo real [0, 1] que define a probabilidade de um novo cromossomo herdar um gene de um indivíduo elite.
- `K`: parâmetro inteiro positivo que define a quantidade de populações independentes.
- `MAXT`: parâmetro inteiro positivo que define o número de *threads* utilizadas no processo de decodificação paralela.
- `X_INTVL`: parâmetro inteiro positivo que define a quantidade de iterações a serem processadas até a troca dos melhores indivíduos entre as populações.
- `X_NUMBER`: parâmetro inteiro positivo que define a quantidade de indivíduos que serão trocados entre as populações.
- `MAX_GENS`: parâmetro inteiro positivo que define a quantidade máxima de iterações que o algoritmo executará, caso não haja outro critério de parada.
- `MAX_STAGT`: parâmetro inteiro positivo que define o número máximo de iterações consecutivas sem melhoria na qualidade das soluções encontradas.

Como saída, o algoritmo produz o resultado do experimento, contemplando o nome do grafo, sua ordem, tamanho e densidade, bem como o melhor valor de *fitness* alcançado e o tempo total de execução, medido em segundos.

6.2 *Decoder*

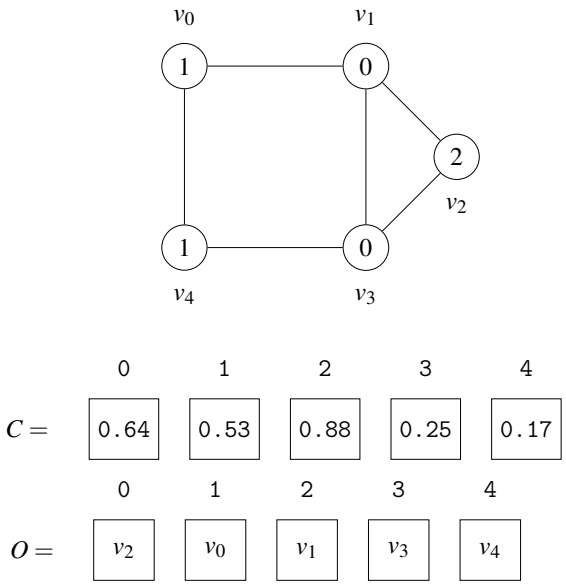
O decodificador é uma função que recebe como entrada um grafo e um vetor de chaves aleatórias enviesadas (cromossomo). Com estas duas informações, o decodificador deve interpretar o cromossomo e dele extrair uma solução viável (uma FDRP) para o grafo. O valor de retorno do decodificador é o valor de aptidão (*fitness*) dessa solução forjada.

Neste trabalho, optou-se por utilizar as chaves aleatórias exclusivamente como um mecanismo de ordenação dos vértices do grafo, responsável por definir a prioridade de rotulação dos vértices.

Dado um grafo G com vértices $V(G) = \{v_0, \ldots, v_{n-1}\}$ e um cromossomo $C =$

$[0, 1, \ldots, n-1]$, cada vértice v_i do grafo está associado à chave aleatória armazenada na posição i do vetor C, a qual determina a posição do vértice v_i na ordenação que será gerada. A posição do vértice determina a ordem em que ele será considerado no processo de construção da solução. A Figura 26 apresenta como uma ordenação pode ser obtida a partir de um cromossomo.

Figura 26 – Representação visual de um grafo G, seguida do cromossomo de chaves aleatórias C e da ordenação O obtida a partir das chaves de C. Os vértices são rearranjados com base na ordem decrescente de suas chaves aleatórias.



Fonte: autoria própria.

Inicialmente, o algoritmo gera uma ordenação dos vértices com base na ordem decrescente de suas chaves aleatórias e atribui o rótulo 0 a todos eles. Em seguida, os vértices são processados sequencialmente de acordo com essa ordenação. Para cada vértice u, verifica-se primeiramente se ele já se encontra corretamente dominado; em caso afirmativo, nenhuma ação adicional é realizada e o algoritmo prossegue para o próximo vértice.

Na sequência, observa-se se o vértice u apresenta um nível de dominação acima do limite permitido pelo PDRP, isto é, se u tem mais de um vizinho com rótulo 2. Caso essa condição seja satisfeita, atribui-se a u o rótulo 1, encerrando-se o processamento desse vértice. Nos dois casos descritos, nenhuma outra decisão é tomada para u.

Caso nenhuma das condições anteriores seja verdadeira, ou seja, se u não tem nenhum vizinho com rótulo 2, inicia-se uma análise da vizinhança de u. Verifica-se a existência de algum vizinho $v \in N(u)$ tal que $f(v) = 0$ e v já esteja dominado. Se tal vértice existir, u não pode receber o rótulo 2, sendo então rotulado com 1. Caso tal vizinho não exista, atribui-se a u o rótulo 2, o que implica a dominação de todos os vértices em sua vizinhança.

Após a etapa de rotulação, a solução construída é submetida a um procedimento de redução de peso, cujo objetivo é preservar a viabilidade da solução enquanto se busca minimizar o seu peso total. Por fim, o valor de *fitness* da solução é calculado e retornado ao algoritmo genético. O Algoritmo 8 demonstra o procedimento de decodificação, enquanto o Algoritmo 3 ilustra o procedimento de redução de peso.

Algoritmo 8: decoder

Entrada: grafo G, vetor de chaves aleatórias C

Valor de *fitness* associado ao cromossomo.

início

  Seja $f = [0, 0, \ldots, 0]$ um vetor de rótulos de tamanho $|V(G)|$

  Seja O um conjunto ordenado dos vértices do grafo em ordem decrescente segundo as chaves aleatórias

  for *cada vértice u na ordem O* **do**

    if $f[u] \neq 0$ **then**

      continue

    end

    if *u tem exatamente um vizinho com rótulo 2* **then**

      continue

    end

    if *u tem mais do que um vizinho com rótulo 2* **then**

      $f[u] \leftarrow 1$

      continue

    end

    if *existe* $v \in N(u)$ *tal que* $f[v] = 0$ *e v está dominado* **then**

      $f[u] \leftarrow 1$

    end

    else

      $f[u] \leftarrow 2$

    end

  end

  reduceWeight(G, f)

  fitness $\leftarrow \sum_{u \in V(G)} f(u)$

  return fitness

fim

7 EXPERIMENTOS E RESULTADOS OBTIDOS

Este capítulo apresenta e discute os resultados obtidos pelos experimentos executados com o GA e BRKGA.

Todos os experimentos foram realizados em um ambiente computacional totalmente controlado, utilizando um computador equipado com processador Intel(R) Core(TM) i5-10400F CPU @ 2.90 GHz, 16 GB de memória RAM Kingston HyperX (2×8 GB) DIMM DDR4 (2666 MHz, dual-channel) e sistema operacional Microsoft Windows 10 Pro, 64 bits — versão 22H2 (compilação 19045.6466).

Os experimentos foram executados em um ambiente Linux provido pelo *Windows Subsystem for Linux* (WSL 2), o qual utiliza um kernel Linux real executado em uma máquina virtual leve baseada em Hyper-V, compartilhando diretamente os recursos de hardware da máquina hospedeira.

As implementações foram desenvolvidas na linguagem `C++` e compiladas com o compilador `G++` versão 13.3.0, utilizando as flags de compilação `-std=c++17`, `-O3`, `-fopenmp`, `-Wextra`, `-Wall`, `-pedantic`, `-Woverloaded-virtual`, `-Wcast-align` e `-Wpointer-arith`.

A análise comparativa das duas metaheurísticas foi conduzida considerando tanto o tempo de execução quanto a qualidade das soluções obtidas. Adicionalmente, as instâncias do problema foram resolvidas por meio de um modelo de Programação Inteira (PI), utilizando o *solver* Gurobi (Gurobi Optimization, LLC, 2024), cuja implementação também foi desenvolvida na linguagem `C++`.

Para cada instância, foi estabelecido um tempo máximo de execução de 900 segundos (15 minutos) tanto para as metaheurísticas quanto para o modelo exato, sendo reportada a melhor solução encontrada ao término desse intervalo, a qual pode ou não corresponder à solução ótima.

No caso das metaheurísticas, cada instância foi executada 30 vezes, com o objetivo de proporcionar uma análise estatística mais robusta e precisa, levando em consideração a natureza estocástica desses métodos.

O objetivo deste trabalho consiste em comparar o desempenho das metaheurísticas entre si, avaliando os algoritmos GA e BRKGA. Por fim, os resultados obtidos por ambos são comparados com aqueles produzidos pela execução do modelo de PLI nas mesmas instâncias de grafos.

7.1 Bases de dados utilizadas

Os experimentos foram conduzidos utilizando um conjunto diversificado de grafos, contemplando diferentes tipos de instâncias. Inicialmente, foram consideradas 75 matrizes simétricas esparsas provenientes da coleção *Harwell–Boeing* (Duff *et al.*, 1989), a qual tem como objetivo disponibilizar um conjunto padrão e representativo de matrizes de teste oriundas de problemas reais das áreas de ciência e engenharia. Essa coleção é amplamente utilizada pela comunidade científica para a avaliação, verificação e comparação de algoritmos destinados à resolução de sistemas lineares esparsos, problemas de mínimos quadrados e cálculos de autovalores, de forma reprodutível.

De maneira análoga, foram utilizados 75 grafos da base *DIMACS* (Johnson; Trick, 1996), cuja finalidade é estabelecer um ambiente de referência comum (*benchmark*), permitindo que pesquisadores avaliem suas implementações algorítmicas e as comparem com aquelas disponíveis na literatura, especialmente no contexto de problemas de otimização combinatória de alta complexidade computacional (NP-difíceis).

Adicionalmente, foram gerados 26 grafos cúbicos aleatórios de diferentes ordens e 24 grafos aleatórios gerais, com variados tamanhos e densidades. Estes últimos foram produzidos da seguinte forma: a ordem dos grafos variou de 150 a 500 vértices, em incrementos de 50 unidades, e, para cada ordem, foram gerados três grafos com densidades aproximadas de 20%, 50% e 80%. Ambas as classes de grafos foram produzidas por meio da biblioteca *NetworkX* (Hagberg *et al.*, 2008), da linguagem Python. Dessa forma, o conjunto experimental é composto por um total de 200 grafos distintos.

7.2 Metodologia de comparação

A comparação entre os algoritmos foi conduzida com base em três critérios principais: (i) o tempo de execução, medido em segundos, utilizado para avaliar a eficiência computacional das abordagens; (ii) a qualidade das soluções, analisada por meio do valor de aptidão (*fitness*), comparando os resultados obtidos pelas metaheurísticas com as soluções ótimas fornecidas pelo modelo de PI; e (iii) o *gap* relativo, calculado com base na melhor solução encontrada por alguma das metaheurísticas.

Este último critério permite mensurar a proximidade das soluções heurísticas em relação ao ótimo global obtido pelo PI, constituindo uma métrica objetiva e confiável para a

avaliação do desempenho dos algoritmos considerados.

7.3 Ajuste automático dos parâmetros

Os parâmetros dos algoritmos GA e BRKGA foram ajustados automaticamente por meio do software `irace` (*Iterated Racing for Automatic Algorithm Configuration*) (Lopez-Ibanez *et al.*, 2016), uma ferramenta de calibração projetada para otimizar a configuração de parâmetros de algoritmos em problemas de otimização. Esse procedimento reduz a necessidade de ajustes manuais e contribui para a melhoria do desempenho das metaheurísticas.

7.3.1 Configurações dos algoritmos

Para a obtenção dos melhores parâmetros por meio do `irace`, as metaheurísticas foram executadas sobre um conjunto representativo de 61 grafos, correspondendo a aproximadamente 30% da base total de instâncias. Esse subconjunto foi selecionado por meio de uma seleção aleatória estratificada, na qual foram escolhidos aproximadamente 30% dos grafos de cada uma das bases consideradas (Harwell-Boeing, DIMACS e grafos gerados), de modo a garantir a presença de instâncias de todas as classes e evitar vieses no processo de calibração. Dessa forma, os parâmetros ajustados refletem um compromisso adequado entre qualidade das soluções e robustez, permitindo uma configuração que generaliza de maneira satisfatória para as demais instâncias utilizadas nos experimentos.

7.3.2 Processo de calibração dos parâmetros

Os valores dos parâmetros utilizados nos algoritmos GA e BRKGA foram definidos a partir de um processo de ajuste automático realizado com o software `irace`, resultando em configurações otimizadas para cada abordagem. Para cada parâmetro, foi especificado um domínio de busca (intervalo de valores), permitindo ao `irace` conduzir o procedimento de corrida iterativa (*iterated racing*) e identificar a melhor configuração com base em um conjunto representativo de 61 grafos. As Tabelas 1 e 2 apresentam os valores finais ajustados para cada parâmetro, bem como os respectivos intervalos de busca considerados durante o processo de calibração.

Tabela 1 – Intervalos de busca utilizados pelo `irace` e parâmetros do GA.

Parâmetro	**Intervalo de busca**	**Valor ajustado**
stagnant	{200, 300, 400}	400
maxGenerations	{500, 700, 900, 1000, 1200}	1000
popFactor	[4, 10]	5
tournSize	[2, 4]	3
eliSize	[0.1, 0.5]	0.4
mutRate	[0.0, 0.9]	0.2

Tabela 2 – Intervalos de busca utilizados pelo `irace` e parâmetros do BRKGA.

Parâmetro	**Intervalo de busca**	**Valor ajustado**
MAX_STAGT	{200, 300, 400}	300
MAX_GENS	{500, 700, 900, 1000, 1200}	700
population_factor	[4, 10]	7
rhoe	[0.6, 0.9]	0.9
pe	[0.1, 0.4]	0.3
pm	[0.0, 0.4]	0.2
K	{1, 2, 3}	2
MAXT	[1, 5]	3
X_INTVL	{50, 100, 150}	100
X_NUMBER	[2, 5]	2

7.4 Desempenho e comparações dos algoritmos

Nas tabelas a seguir, $|V|$ e $|E|$ representam, respectivamente, a ordem e o tamanho do grafo, enquanto D indica sua densidade. As colunas F/GA e F/BR correspondem aos valores médios de *fitness* obtidos pelo GA e pelo BRKGA, respectivamente. Os valores destacados em negrito nas colunas F/GA e F/BR correspondem às instâncias em que o algoritmo atingiu o valor ótimo. A sigla F/PI indica o valor de *fitness* encontrado pela formulação de Programação Inteira (PI), sendo que o simbolo * nesta coluna significa que o valor apresentado é o ótimo. Os tempos de execução dos algoritmos são apresentados nas colunas T/GA e T/BR. As colunas MIN/GA e MIN/BR reportam os melhores valores de *fitness* obtidos por cada algoritmo ao longo das execuções, enquanto STD/GA e STD/BR indicam os respectivos desvios padrão. Por fim, a coluna GAP indica o gap relativo entre a melhor solução encontrada dentre os dois algoritmos e o resultado obtido pelo PI.

7.4.1 Resultados dos algoritmos em grafos aleatórios

A análise dos resultados para as instâncias de grafos aleatórios, apresentadas na Tabela 3, evidencia diferenças marcantes entre o desempenho do GA e do BRKGA, especialmente

no que diz respeito à qualidade das soluções, tempo de execução e estabilidade.

De forma geral, ambos os algoritmos foram capazes de atingir, em diversas instâncias, valores de *fitness* iguais ou muito próximos aos obtidos pela formulação de Programação Inteira. Esse comportamento é particularmente evidente em instâncias como `rd_450_0.8`, `rd_200_0.8` e `rd_300_0.8`, nas quais tanto o GA quanto o BRKGA alcançaram o valor ótimo, demonstrando elevada eficácia em grafos de alta densidade.

No entanto, em instâncias de grande porte e densidade baixa, observa-se uma vantagem consistente do GA em termos de qualidade da solução. Em grafos como `rd_400_0.2`, `rd_450_0.2` e `rd_200_0.2`, o GA apresentou valores médios de *fitness* significativamente inferiores aos do BRKGA, além de melhores valores mínimos, indicando maior capacidade de exploração do espaço de busca nessas configurações mais desafiadoras.

Em relação ao tempo de execução, o BRKGA mostrou-se sistematicamente mais eficiente, com valores substancialmente menores que os do GA em todas as instâncias analisadas. Essa diferença torna-se ainda mais expressiva à medida que o número de vértices e arestas aumenta, como observado nas instâncias `rd_500_0.2` e `rd_500_0.5`, nas quais o tempo do GA cresce de forma mais acentuada.

Quanto à estabilidade, os desvios padrão indicam um comportamento mais consistente do BRKGA. Em grande parte das instâncias, o desvio padrão do BRKGA é nulo ou significativamente inferior ao do GA, sugerindo menor variabilidade entre execuções. Por outro lado, o GA apresenta maior dispersão nos resultados, o que, apesar de indicar menor estabilidade, também reflete uma abordagem mais exploratória, capaz de encontrar soluções de melhor qualidade em cenários mais complexos.

Os valores de gap variam de forma considerável ao longo da tabela, permanecendo nulos em instâncias onde algum dos algoritmos atinge o ótimo, mas atingindo valores mais elevados em grafos maiores e mais esparsos, como `rd_400_0.2` e `rd_450_0.2`.

Em síntese, os resultados obtidos para as instâncias de grafos aleatórios indicam que o GA e o BRKGA apresentam desempenhos complementares, com vantagens distintas dependendo das características do grafo analisado. Enquanto ambos os algoritmos são capazes de alcançar soluções ótimas ou próximas ao ótimo em grafos mais densos, o GA mostra-se mais eficaz na obtenção de soluções de melhor qualidade em instâncias de grande porte e baixa densidade, ao custo de um maior tempo de execução e maior variabilidade. Por sua vez, o BRKGA destaca-se pela elevada eficiência computacional e maior estabilidade dos resultados,

sendo particularmente adequado quando o tempo de processamento e a consistência entre execuções são fatores prioritários.

Tabela 3 – Resultados obtidos para a base de grafos aleatórios.

Graph	$\|V\|$	$\|E\|$	D	F/GA	F/BR	F/PI	T/GA	T/BR	MIN/GA	MIN/BR	STD/GA	STD/BR	GAP
rd_150_0.2	150	2260	20.22%	74	88	64	1.842	0.162	66	79	4.29	5.12	15.62%
rd_150_0.5	150	5576	49.90%	61	61	53*	1.571	0.086	61	61	0.0	0.0	15.09%
rd_150_0.8	150	8933	79.94%	**20**	**20**	20*	1.681	0.092	20	20	0.0	0.0	0.0%
rd_200_0.2	200	4021	20.21%	108	131	86	3.204	0.221	97	131	6.41	0.0	25.58%
rd_200_0.5	200	9907	49.78%	80	80	75	3.195	0.162	80	80	0.0	0.0	6.67%
rd_200_0.8	200	15844	79.62%	**30**	**30**	30*	4.497	0.141	30	30	0.0	0.0	0.0%
rd_250_0.2	250	6284	20.19%	138	167	122	5.112	0.322	130	167	4.55	0.0	13.11%
rd_250_0.5	250	15544	49.94%	105	107	98	7.397	0.235	102	107	2.07	0.0	7.14%
rd_250_0.8	250	24806	79.70%	**36**	**36**	36*	7.586	0.201	36	36	0.0	0.0	0.0%
rd_300_0.2	300	9066	20.21%	169	218	142	8.554	0.406	157	218	6.05	0.0	19.01%
rd_300_0.5	300	22366	49.87%	128	128	121	9.663	0.333	128	128	0.0	0.0	5.79%
rd_300_0.8	300	35742	79.69%	**44**	**44**	44*	12.274	0.259	44	44	0.0	0.0	0.0%
rd_350_0.2	350	12311	20.16%	201	259	173	12.186	0.502	186	259	6.54	0.0	16.18%
rd_350_0.5	350	30572	50.06%	149	149	140	14.52	0.432	149	149	0.0	0.0	6.43%
rd_350_0.8	350	48730	79.79%	**50**	**50**	50*	16.998	0.39	50	50	0.0	0.0	0.0%
rd_400_0.2	400	16057	20.12%	229	293	181	17.895	0.722	215	293	7.7	0.0	26.52%
rd_400_0.5	400	39927	50.03%	**174**	**174**	174	19.378	0.645	174	174	0.0	0.0	0.0%
rd_400_0.8	400	63677	79.80%	**60**	**60**	60*	26.994	0.49	60	60	0.0	0.0	0.0%
rd_450_0.2	450	20245	20.04%	262	333	218	18.059	0.986	241	333	8.91	0.0	20.18%
rd_450_0.5	450	50489	49.98%	199	199	196	26.793	0.703	199	199	0.0	0.0	1.53%
rd_450_0.8	450	80679	79.86%	**64**	**64**	64*	36.471	0.631	64	64	0.0	0.0	0.0%
rd_500_0.2	500	25031	20.06%	295	373	249	24.446	1.202	280	373	9.07	0.0	18.47%
rd_500_0.5	500	62388	50.01%	220	220	218	35.722	0.825	220	220	0.0	0.0	0.92%
rd_500_0.8	500	99649	79.88%	**78**	**78**	78*	48.199	0.785	78	78	0.0	0.0	0.0%

7.4.2 Resultados dos algoritmos em grafos cúbicos

A análise dos resultados para as instâncias de grafos cúbicos, apresentadas na Tabela 4, revela um comportamento relativamente equilibrado entre o GA e o BRKGA, com diferenças pontuais em termos de qualidade da solução, tempo de execução e estabilidade.

De forma geral, o BRKGA obteve valores médios de *fitness* ligeiramente melhores aos do GA em grande parte das instâncias, indicando melhor qualidade das soluções encontradas.

Esse comportamento é particularmente evidente em grafos de maior porte, como `cubic_700`, `cubic_750` e `cubic_800`, nos quais o BRKGA apresenta menores valores médios e melhores valores mínimos, enquanto se sobressai no que se diz respeito a custo computacional.

Em diversas instâncias menores, como `cubic_50` e `cubic_100`, ambos os algoritmos foram capazes de atingir valores bem próximos ao ótimo indicado pelo PI, evidenciando que, para grafos de baixa ordem, as duas abordagens são igualmente eficazes em termos de qualidade da solução. Nessas instâncias, o BRKGA se destaca por apresentar tempos de execução significativamente inferiores, tornando-se mais eficiente sob a perspectiva computacional.

No que se refere ao tempo de execução, observa-se que o BRKGA é sistematicamente mais rápido em todas as instâncias avaliadas, com valores consideravelmente menores que os do GA, especialmente à medida que o número de vértices cresce. Essa diferença torna-se mais pronunciada em instâncias como `cubic_850` e `cubic_876`, nas quais o tempo do GA aumenta de forma mais acentuada.

Quanto à estabilidade, os desvios padrão apresentados nas colunas indicam que o BRKGA tende a produzir resultados mais consistentes entre diferentes execuções. Em praticamente todas as instâncias, o desvio padrão do BRKGA é inferior ou muito próximo ao do GA, sugerindo menor variabilidade e maior previsibilidade do comportamento do algoritmo. O BRKGA, embora menos sujeito a flutuações, demonstra maior capacidade exploratória, refletida nos melhores valores mínimos de *fitness* obtidos.

Os valores de gap permanecem relativamente baixos em toda a tabela, geralmente abaixo de 8%, o que indica que algum dos algoritmos consegue produzir soluções próximas às obtidas pela formulação de Programação Inteira, mesmo em grafos maiores. Ainda assim, nota-se uma leve tendência de aumento do gap à medida que a ordem do grafo cresce, evidenciando o impacto da escalabilidade do problema.

Em síntese, os resultados obtidos para as instâncias de grafos cúbicos indicam que o BRKGA apresenta desempenho globalmente superior, sobretudo quando considerados simultaneamente a qualidade das soluções, o tempo de execução e a estabilidade entre execuções. Embora ambos os algoritmos sejam capazes de produzir soluções próximas ao ótimo, especialmente em instâncias de menor porte, o BRKGA demonstra maior robustez e eficiência computacional à medida que a dimensão do problema aumenta. Assim, para essa classe de grafos, o BRKGA mostra-se a alternativa mais adequada na maioria dos cenários, enquanto o GA permanece competitivo em termos de qualidade, porém com maior custo computacional e variabilidade nos

resultados.

Tabela 4 – Resultados obtidos para instâncias de grafos cúbicos. A coluna UB refere-se ao melhor limitante superior conhecido, conforme reportado na literatura.

Graph	$\|V\|$	$\|E\|$	D	UB	F/GA	F/BR	F/PI	T/GA	T/BR	MIN/GA	MIN/BR	STD/GA	STD/BR	GAP
cubic_50	50	75	6.12%	37.50	29	27	26*	0.109	0.015	28	26	0.96	1.0	3.85%
cubic_76	76	114	4.00%	57.00	**42**	**42**	42*	0.359	0.033	42	42	1.36	0.96	0.0%
cubic_100	100	150	3.03%	75.00	58	56	54*	0.621	0.04	56	54	1.17	1.14	3.7%
cubic_150	150	225	2.01%	112.50	85	84	80*	2.005	0.122	82	82	1.85	1.36	5.0%
cubic_176	176	264	1.71%	132.00	101	100	94*	2.802	0.182	98	96	1.75	1.61	6.38%
cubic_200	200	300	1.51%	150.00	115	113	106*	3.638	0.188	110	112	2.16	1.36	6.6%
cubic_250	250	375	1.20%	187.50	143	141	131*	6.546	0.376	140	138	2.13	1.67	7.63%
cubic_276	276	414	1.09%	207.00	158	156	146*	8.237	0.468	154	154	2.27	1.71	6.85%
cubic_300	300	450	1.00%	225.00	172	171	162	9.812	0.492	168	168	2.41	2.0	5.56%
cubic_350	350	525	0.86%	262.50	201	200	188	15.205	0.664	196	196	2.59	2.01	6.38%
cubic_376	376	564	0.80%	282.00	216	215	200	19.234	0.877	212	212	2.75	1.92	7.5%
cubic_400	400	600	0.75%	300.00	231	228	213	18.144	0.96	228	224	2.15	2.29	7.04%
cubic_450	450	675	0.67%	337.50	261	257	244	25.916	1.428	254	252	3.32	2.77	5.33%
cubic_476	476	714	0.63%	357.00	276	273	258	31.23	1.476	271	268	3.15	2.56	5.81%
cubic_500	500	750	0.60%	375.00	289	286	270	34.251	1.703	282	280	3.47	2.98	5.93%
cubic_550	550	825	0.55%	412.50	320	316	298	42.741	2.11	316	310	3.12	2.8	6.04%
cubic_576	576	864	0.52%	432.00	335	330	314	46.233	2.288	328	324	3.67	2.63	5.1%
cubic_600	600	900	0.50%	450.00	350	345	324	50.386	2.401	344	338	3.75	2.67	6.48%
cubic_650	650	975	0.46%	487.50	380	372	354	60.116	3.076	376	366	2.94	3.0	5.08%
cubic_676	676	1014	0.44%	507.00	396	386	364	59.217	4.077	390	378	3.55	3.58	6.04%
cubic_700	700	1050	0.43%	525.00	411	402	380	68.952	4.598	406	396	2.68	3.21	5.79%
cubic_750	750	1125	0.40%	562.50	445	430	410	82.574	5.924	440	426	2.65	2.55	4.88%
cubic_776	776	1164	0.39%	582.00	460	445	420	87.527	6.267	454	436	3.6	3.98	5.95%
cubic_800	800	1200	0.38%	600.00	476	458	436	95.855	4.791	468	452	4.47	3.51	5.05%
cubic_850	850	1275	0.35%	637.50	507	487	464	107.63	8.184	500	482	3.68	3.25	4.96%
cubic_876	876	1314	0.34%	657.00	526	502	470	108.718	8.626	518	494	3.57	4.24	6.81%

7.4.3 Resultados dos algoritmos na base de grafos DIMACS

A análise dos resultados para a base *DIMACS* apresentada na Tabela 5 evidencia diferenças claras entre o desempenho do GA e do BRKGA. De modo geral, o GA apresentou melhor desempenho em relação à qualidade das soluções, obtendo, na maioria das instâncias, valores médios de *fitness* inferiores aos do BRKGA. Em diversas instâncias da base, como `hamming6-4`, `queen8_8`, e `p_hat500-1`, o GA foi capaz de alcançar soluções bem próximas ou exatamente iguais ao fornecido pela formulação de Programação Inteira (PI), demonstrando sua eficácia em explorar o espaço de busca mesmo em grafos de densidade intermediária.

Por outro lado, o BRKGA apresentou tempos de execução consistentemente menores, o que o torna uma alternativa atrativa em cenários onde o custo computacional é um fator crítico. Em algumas instâncias específicas, como `le450_5c` e `le450_5d`, o BRKGA não apenas superou o GA em tempo, mas também obteve valores médios de *fitness* inferiores, indicando que, para determinadas estruturas de grafo, sua estratégia baseada em chaves aleatórias é mais adequada.

Um aspecto importante a ser destacado é a estabilidade dos algoritmos. O BRKGA apresentou, de forma geral, desvios padrão significativamente menores do que os do GA, sugerindo um comportamento mais consistente entre diferentes execuções. Esse fator indica que o BRKGA tende a convergir para soluções próximas a um mesmo patamar, enquanto o GA, embora frequentemente encontre soluções de melhor qualidade, apresenta maior variabilidade, o que pode ser explicado pela maior diversidade introduzida por seus operadores evolutivos.

Em algumas instâncias, como `queen13_13`, `queen14_14` e `queen11_11`, observa-se que ambos os algoritmos apresentam dificuldades em atingir os valores ótimos fornecidos pelo PL, resultando em gaps mais elevados. Em contrapartida, nas instâncias `brock800_3`, `brock800_1` e `brock800_2`, os algoritmos superaram o PI, produzindo valores de gap negativos.

Apesar disso, o GA apresenta, em geral, soluções de melhor qualidade, enquanto o BRKGA se destaca pelo menor tempo de execução e maior estabilidade, o que evidencia o caráter complementar das duas abordagens.

Em síntese, os resultados indicam que o GA é mais indicado quando a prioridade é a obtenção de soluções de melhor qualidade, mesmo ao custo de um maior tempo de execução, ao passo que o BRKGA se mostra mais adequado em cenários onde rapidez e consistência são essenciais. Dessa forma, a escolha do algoritmo mais apropriado depende do compromisso desejado entre qualidade da solução, tempo computacional e robustez dos resultados.

Tabela 5 – Resultados obtidos para a base *DIMACS*.

Graph	$\|V\|$	$\|E\|$	D	F/GA	F/BR	F/PI	T/GA	T/BR	MIN/GA	MIN/BR	STD/GA	STD/BR	GAP
myciel3	11	20	36.36%	**7**	**7**	7*	0.006	0.004	7	7	0.0	0.0	0.0%
queen7_7	49	476	40.48%	22	26	19*	0.117	0.013	20	26	1.4	0.0	15.79%
hamming6-4	64	704	34.92%	**24**	**24**	24*	0.2	0.024	24	43	0.0	0.0	0.0%
queen8_8	64	728	36.11%	28	38	24*	0.203	0.021	24	38	1.19	0.0	16.67%
hamming6-2	64	1824	90.48%	**8**	**8**	8*	0.14	0.017	8	8	0.0	0.0	0.0%
johnson8-4-4	70	1855	76.81%	**18**	**18**	18*	0.179	0.023	18	18	0.0	0.0	0.0%
myciel6	95	755	16.91%	16	49	15*	0.382	0.042	15	49	1.61	0.0	6.67%
queen10_10	100	1470	29.70%	46	66	32	0.609	0.047	32	66	4.12	0.0	43.75%
johnson16-2-4	120	5460	76.47%	**30**	**30**	30*	0.902	0.052	30	30	0.0	0.0	0.0%
queen11_11	121	1980	27.27%	54	82	37	0.674	0.089	37	82	7.7	0.0	45.95%
miles250	125	387	4.99%	**45**	**45**	45*	0.743	0.091	45	45	0.5	0.85	0.0%
DSJC125.1	125	736	9.50%	57	61	50	0.968	0.101	52	59	2.91	1.5	14.0%
DSJC125.9	125	6961	89.82%	**6**	**6**	6*	0.645	0.054	6	6	0.0	0.0	0.0%
zeroin.i.1	126	4100	52.06%	**4**	**4**	4*	0.977	0.098	4	4	0.57	0.0	0.0%
miles500	128	1170	14.39%	22	22	21*	0.764	0.083	21	21	1.42	1.11	4.76%
miles750	128	2113	26.00%	19	20	18*	0.631	0.085	18	19	2.16	1.01	5.56%
anna	138	493	5.22%	**50**	**50**	50*	0.778	0.078	50	67	0.37	0.0	0.0%
mulsol.i.1	138	3925	41.52%	**4**	**4**	4*	0.898	0.082	4	4	0.0	0.0	0.0%
zeroin.i.2	157	3541	28.92%	**4**	**4**	4*	0.929	0.143	4	4	0.25	0.0	0.0%
queen13_13	169	3328	23.44%	86	122	46	1.347	0.122	56	122	8.13	0.0	86.96%
mulsol.i.2	173	3885	26.11%	**4**	**4**	4*	1.34	0.127	4	4	0.0	0.0	0.0%
mulsol.i.4	175	3946	25.92%	**4**	**4**	4*	1.287	0.132	4	4	0.0	0.0	0.0%
mulsol.i.5	176	3973	25.80%	**4**	**4**	4*	1.194	0.15	4	4	0.0	0.0	0.0%
myciel7	191	2360	13.01%	**18**	**18**	18*	1.127	0.178	18	97	1.17	0.0	0.0%
queen14_14	196	4186	21.90%	103	146	54	2.555	0.206	77	146	7.96	0.0	90.74%
c-fat200-5	200	8473	42.58%	**32**	**32**	32*	1.45	0.155	32	32	0.0	0.0	0.0%
brock200_3	200	12048	60.54%	**67**	**67**	67*	2.113	0.157	67	67	0.0	0.0	0.0%
brock200_4	200	13089	65.77%	**54**	**54**	54*	2.074	0.173	54	54	0.0	0.0	0.0%
sanr200_0.7	200	13868	69.69%	**40**	**40**	40*	2.118	0.168	40	40	0.0	0.0	0.0%
san200_0.7_1	200	13930	70.00%	**46**	**46**	46*	2.274	0.138	46	46	0.0	0.0	0.0%
gen200_p0.9_44	200	17910	90.00%	**11**	**11**	11*	2.513	0.139	11	11	0.0	0.0	0.0%
gen200_p0.9_55	200	17910	90.00%	**11**	**11**	11*	2.438	0.137	11	11	0.0	0.0	0.0%
san200_0.9_3	200	17910	90.00%	**14**	**14**	14*	2.547	0.125	14	14	0.0	0.0	0.0%
san200_0.9_2	200	17910	90.00%	**13**	**13**	13*	2.673	0.148	13	13	0.0	0.0	0.0%
DSJC250.1	250	3218	10.34%	133	138	118	4.441	0.405	123	123	5.04	5.81	12.71%
DSJC250.5	250	15668	50.34%	101	104	95	4.994	0.233	101	104	0.69	0.0	6.32%
hamming8-2	256	31616	96.86%	**10**	**10**	10*	3.816	0.198	10	10	0.0	0.0	0.0%
fpsol2.i.1	269	11654	32.33%	**4**	**4**	4*	3.67	0.271	4	4	0.0	0.0	0.0%
p_hat300-2	300	21928	48.89%	**72**	**72**	72	4.422	0.334	72	72	0.0	0.0	0.0%
school1_nsh	352	14612	23.65%	116	116	112	12.661	0.522	111	116	2.6	0.0	3.57%
fpsol2.i.3	363	8688	13.22%	**4**	**4**	4*	4.116	0.489	4	4	0.0	0.0	0.0%
school1	385	19095	25.83%	**101**	**101**	101	14.9	0.634	101	102	0.31	0.0	0.0%
sanr400_0.5	400	39984	50.11%	**168**	**168**	168	10.047	0.552	168	168	0.0	0.0	0.0%

Continua na próxima página

Tabela 5 – continuação

Graph	$\|V\|$	$\|E\|$	D	F/GA	F/BR	F/PI	T/GA	T/BR	MIN/GA	MIN/BR	STD/GA	STD/BR	GAP
san400_0.7_2	400	55860	70.00%	**97**	**97**	97*	12.547	0.519	97	97	0.0	0.0	0.0%
san400_0.7_3	400	55860	70.00%	**94**	**94**	94*	12.86	0.582	94	94	0.0	0.0	0.0%
san400_0.7_1	400	55860	70.00%	**100**	**100**	100*	12.196	0.517	100	100	0.0	0.0	0.0%
sanr400_0.7	400	55869	70.01%	**91**	**91**	91	12.581	0.528	91	91	0.0	0.0	0.0%
brock400_3	400	59681	74.79%	**79**	**79**	79	12.747	0.494	79	79	0.0	0.0	0.0%
brock400_2	400	59786	74.92%	**73**	**73**	73	12.835	0.516	73	73	0.0	0.0	0.0%
san400_0.9_1	400	71820	90.00%	**27**	**27**	27*	13.41	0.536	27	27	0.0	0.0	0.0%
gen400_p0.9_75	400	71820	90.00%	**21**	**21**	21*	13.63	0.478	21	21	0.0	0.0	0.0%
gen400_p0.9_55	400	71820	90.00%	**26**	**26**	26*	13.334	0.486	26	26	0.0	0.0	0.0%
le450_5a	450	5714	5.66%	251	249	217	18.893	1.514	234	227	8.02	9.5	14.75%
le450_5b	450	5734	5.68%	249	249	207	16.474	1.582	238	229	5.71	7.36	20.29%
le450_25a	450	8260	8.18%	248	262	210	16.512	1.554	238	250	6.43	6.92	18.1%
le450_5d	450	9757	9.66%	294	272	214	13.117	1.309	273	251	9.61	9.93	27.1%
le450_5c	450	9803	9.70%	293	264	201	13.728	1.331	271	241	9.49	10.85	31.34%
le450_15d	450	16750	16.58%	259	294	216	14.389	1.592	242	257	8.85	12.76	19.91%
DSJR500.1	500	3555	2.85%	114	117	98	16.984	1.849	106	110	3.73	3.93	16.33%
c-fat500-1	500	4459	3.57%	**60**	**60**	60*	12.621	0.856	60	64	1.63	0.0	0.0%
c-fat500-2	500	9139	7.33%	**38**	**38**	38*	7.171	0.844	38	38	0.0	0.0	0.0%
c-fat500-5	500	23191	18.59%	**41**	**41**	41*	9.31	0.828	41	41	0.0	0.0	0.0%
p_hat500-1	500	31569	25.31%	271	297	270	19.235	1.005	255	297	8.36	0.0	0.37%
c-fat500-10	500	46627	37.38%	**68**	**68**	68*	12.277	0.773	68	128	0.0	0.0	0.0%
DSJC500_5	500	62624	50.20%	215	215	197	16.939	0.957	215	215	0.0	0.0	9.14%
p_hat500-3	500	93800	75.19%	**49**	**49**	49	22.689	0.881	49	49	0.0	0.0	0.0%
DSJC500.9	500	112437	90.13%	**30**	**30**	30*	22.463	0.8	30	30	0.0	0.0	0.0%
DSJR500.1c	500	121275	97.21%	**4**	**4**	4*	22.452	0.734	4	4	0.0	0.0	0.0%
inithx.i.1	519	18707	13.92%	**4**	**4**	4*	21.204	1.008	4	4	0.0	0.0	0.0%
homer	556	1628	1.06%	**270**	**270**	270*	27.957	1.677	270	409	1.12	1.27	0.0%
inithx.i.2	558	13979	9.00%	**4**	**4**	4*	13.618	1.226	4	4	0.0	0.0	0.0%
brock800_3	800	207333	64.87%	243	243	248	71.579	2.643	243	243	0.0	0.0	-2.02%
brock800_1	800	207505	64.93%	241	241	249	85.935	3.078	241	241	0.0	0.0	-3.21%
brock800_4	800	207643	64.97%	**236**	**236**	236	79.873	3.079	236	236	0.0	0.0	0.0%
brock800_2	800	208166	65.13%	235	235	250	96.129	2.803	235	235	0.0	0.0	-6.0%

7.4.4 Resultados dos algoritmos na base de grafos Harwell-Boeing

A análise dos resultados apresentados na Tabela 6 evidencia um comportamento heterogêneo entre o GA e o BRKGA, refletindo a diversidade estrutural das instâncias avaliadas. De maneira geral, ambos os algoritmos demonstram elevada capacidade de encontrar soluções de boa qualidade, frequentemente próximas ou iguais às obtidas pela formulação de Programação Inteira, especialmente em grafos de pequeno e médio porte.

Em um número significativo de instâncias, como `dwt__245`, `can__62`, `bcsstk20` e `bcspwr04`, ambos os algoritmos atingem valores de *fitness* iguais ou muito próximos ao ótimo fornecido pelo PI, indicando que, para grafos de pequena ordem e estrutura esparsa, as duas abordagens são igualmente eficazes. Nesses casos, o BRKGA se destaca por apresentar tempos de execução substancialmente inferiores, reforçando sua eficiência computacional.

Por outro lado, em instâncias de maior porte e baixa densidade, como `bcsstk10`, `eris1176` e `can_1054`, observa-se uma vantagem do GA em termos de qualidade da solução, com valores médios de *fitness* inferiores aos do BRKGA e melhores valores mínimos. Esse comportamento sugere maior capacidade exploratória do GA, que se mostra mais eficaz em escapar de ótimos locais em cenários mais desafiadores, ainda que à custa de maior tempo computacional.

No que diz respeito ao tempo de execução, o BRKGA mantém desempenho superior em praticamente todas as instâncias avaliadas, apresentando tempos significativamente menores que os do GA. Essa diferença torna-se especialmente relevante à medida que o número de vértices aumenta, evidenciando melhor escalabilidade do BRKGA em termos computacionais.

A análise dos desvios padrão indica que o BRKGA tende a apresentar maior estabilidade, com valores frequentemente inferiores aos do GA. Em diversas instâncias, o desvio padrão do BRKGA é nulo, sugerindo forte convergência para soluções semelhantes entre execuções. Em contrapartida, o GA apresenta maior variabilidade, o que, embora indique menor estabilidade, reflete uma busca mais diversificada, capaz de alcançar soluções de melhor qualidade em instâncias complexas.

Os valores de gap reforçam essas observações, permanecendo baixos ou nulos em grafos de menor porte e aumentando em instâncias maiores ou com estruturas menos favoráveis. Ainda assim, os valores são relativamente controlados, demonstrando boa proximidade em relação às soluções ótimas do PI.

Em síntese, os resultados indicam que o GA e o BRKGA apresentam características

complementares: o GA destaca-se pela qualidade das soluções em instâncias de maior porte e baixa densidade, enquanto o BRKGA sobressai em eficiência computacional e estabilidade. Assim, a escolha do algoritmo mais adequado depende das características estruturais do grafo e do compromisso desejado entre qualidade da solução, tempo de execução e robustez dos resultados.

Tabela 6 – Resultados obtidos para a base *Harwell-Boeing*.

Graph	$\|V\|$	$\|E\|$	D	F/GA	F/BR	F/PI	T/GA	T/BR	MIN/GA	MIN/BR	STD/GA	STD/BR	GAP
can___24	24	68	24.64%	**8**	**8**	8*	0.023	0.004	8	8	0.0	0.0	0.0%
bcspwr01	39	46	6.21%	**23**	**23**	23*	0.055	0.011	23	23	0.37	0.0	0.0%
bcspwr02	49	59	5.02%	**28**	**28**	28*	0.08	0.017	28	29	0.0	0.96	0.0%
dwt___59	59	104	6.08%	**29**	**29**	29*	0.153	0.017	29	29	0.72	0.59	0.0%
can___61	61	248	13.55%	**19**	**19**	19*	0.131	0.018	19	24	0.65	1.09	0.0%
can___62	62	78	4.12%	**36**	**36**	36*	0.153	0.018	36	36	0.57	0.55	0.0%
bfw62b	62	140	7.40%	32	33	31*	0.145	0.02	31	33	0.91	0.25	3.23%
dwt___66	66	127	5.92%	28	28	26*	0.167	0.022	26	26	1.52	0.92	7.69%
dwt___72	72	75	2.93%	47	46	45*	0.212	0.026	45	45	0.93	0.61	2.22%
can___73	73	152	5.78%	**35**	**35**	35*	0.197	0.027	35	39	0.86	1.03	0.0%
ash85	85	219	6.13%	35	35	33*	0.294	0.043	33	34	1.0	0.89	6.06%
dwt___87	87	227	6.07%	37	39	35*	0.304	0.045	35	38	1.82	0.79	5.71%
can___96	96	336	7.37%	**26**	**26**	26*	0.322	0.035	26	26	0.0	0.92	0.0%
nos4	100	247	4.99%	44	50	42*	0.535	0.049	42	49	1.5	0.64	4.76%
bcspwr03	118	179	2.59%	64	67	63*	0.566	0.087	63	65	1.43	0.98	1.59%
bcsstk04	132	1758	20.33%	42	58	40*	0.717	0.063	40	58	4.39	0.0	5.0%
lund_b	147	1147	10.69%	**26**	**26**	26*	1.039	0.102	26	26	2.38	1.35	0.0%
lund_a	147	1151	10.73%	**26**	**26**	26*	1.26	0.092	26	26	2.97	1.07	0.0%
bcsstk05	153	1135	9.76%	**38**	**38**	38*	1.658	0.118	38	38	1.72	0.25	0.0%
can__161	161	608	4.72%	43	42	41*	1.006	0.151	41	41	4.58	3.5	2.44%
dwt__162	162	510	3.91%	54	53	52*	1.349	0.156	52	52	1.96	1.4	1.92%
dwt__198	198	597	3.06%	57	60	56*	2.004	0.22	56	56	1.89	2.88	1.79%
will199	199	660	3.35%	87	96	81	2.151	0.23	81	91	3.59	2.94	7.41%
dwt__209	209	767	3.53%	75	76	68*	2.165	0.264	70	72	2.65	2.37	10.29%
can__229	229	774	2.96%	76	79	74*	2.469	0.327	74	75	2.27	2.88	2.7%
dwt__234	234	300	1.10%	125	134	124*	2.799	0.382	124	127	1.01	4.26	0.81%
nos1	237	390	1.39%	133	135	132*	3.281	0.287	132	134	0.92	1.48	0.76%
dwt__245	245	608	2.03%	118	121	113*	3.291	0.391	116	117	1.63	2.23	4.42%
can__268	268	1407	3.93%	89	129	83*	5.672	0.482	85	122	2.07	4.58	7.23%
bcspwr04	274	669	1.79%	133	141	131*	4.01	0.553	131	137	1.72	2.85	1.53%
ash292	292	958	2.25%	105	109	99*	8.418	0.603	100	106	3.2	2.53	6.06%
can__292	292	1124	2.65%	106	108	99*	6.942	0.508	102	100	1.4	5.32	7.07%
dwt__310	310	1069	2.23%	98	99	88*	9.227	0.562	92	92	3.48	3.9	11.36%
dwt__346	346	1440	2.41%	153	156	145	15.4	0.799	146	150	4.19	2.78	5.52%
dwt__361	361	1296	1.99%	113	114	100*	10.611	0.801	108	106	2.8	3.63	13.0%
bfw398b	398	1256	1.59%	188	187	172	17.764	1.013	182	184	2.84	2.26	8.72%

Continua na próxima página

Tabela 6 – continuação

Graph	$\|V\|$	$\|E\|$	D	F/GA	F/BR	F/PI	T/GA	T/BR	MIN/GA	MIN/BR	STD/GA	STD/BR	GAP
dwt__419	419	1572	1.80%	150	151	138	9.51	1.212	144	144	3.7	2.55	8.7%
bcsstm07	420	3416	3.88%	134	134	116	18.094	0.997	125	132	5.36	2.24	15.52%
bcsstk06	420	3720	4.23%	128	116	112	25.535	1.255	116	112	5.62	3.77	3.57%
nos5	468	2352	2.15%	202	207	189	15.271	1.379	194	199	5.49	5.14	6.88%
bcsstk20	485	1325	1.13%	187	189	182*	24.546	1.421	184	186	2.71	2.2	2.75%
dwt__492	492	1332	1.10%	189	194	178*	27.428	1.875	183	185	4.04	4.53	6.18%
494_bus	494	586	0.48%	298	307	293*	20.028	2.011	295	303	1.63	2.99	1.71%
dwt__503	503	2762	2.19%	129	142	117*	15.292	1.852	121	130	4.38	6.77	10.26%
lshp_577	577	1656	1.00%	225	226	193	21.808	2.338	218	214	4.87	6.0	16.58%
dwt__592	592	2256	1.29%	150	166	129*	22.226	2.65	138	147	8.92	10.58	16.28%
dwt__607	607	2262	1.23%	186	194	170	23.813	2.768	174	185	5.28	5.75	9.41%
662_bus	662	906	0.41%	374	384	358*	53.725	4.189	370	378	2.46	4.2	4.47%
can__715	715	2975	1.17%	200	207	190	57.601	3.429	197	203	2.21	2.43	5.26%
nos7	729	1944	0.73%	344	329	298	35.78	5.364	335	319	5.15	5.28	10.4%
dwt__758	758	2618	0.91%	226	234	198*	37.12	4.718	208	226	7.95	6.19	14.14%
can__838	838	4586	1.31%	328	451	307	133.403	6.558	318	441	6.0	5.61	6.84%
young1c	841	1624	0.46%	435	421	358*	46.811	6.65	427	407	3.15	4.37	17.6%
dwt__869	869	3208	0.85%	265	265	223	51.124	7.339	249	249	8.45	8.13	18.83%
dwt__878	878	3285	0.85%	257	268	203*	52.505	6.213	224	255	12.83	7.44	26.6%
gr_30_30	900	3422	0.85%	258	269	200*	92.14	6.547	226	246	16.06	10.16	29.0%
jagmesh1	936	2664	0.61%	400	366	315	58.398	7.673	390	356	4.75	7.14	16.19%
nos2	957	1590	0.35%	553	553	532*	60.347	8.581	550	546	1.66	2.82	3.95%
sherman1	1000	1375	0.28%	665	661	631	128.625	10.583	656	657	4.1	3.38	4.75%
jagmesh2	1009	2928	0.58%	435	395	337	71.777	11.007	413	383	8.38	7.07	17.21%
can_1054	1054	5571	1.00%	401	522	364	166.042	12.567	387	504	7.61	9.85	10.16%
can_1072	1072	5686	0.99%	404	533	383	177.8	13.885	395	509	6.0	9.96	5.48%
bcsstk09	1083	8677	1.48%	441	328	309	183.512	15.886	415	291	15.34	16.27	6.15%
bcsstk10	1086	10492	1.78%	116	136	104*	99.948	11.788	110	115	3.22	12.24	11.54%
jagmesh3	1089	3136	0.53%	476	426	373	125.039	12.629	465	411	6.21	7.33	14.21%
1138_bus	1138	1458	0.23%	696	700	647*	107.107	21.844	689	688	3.48	5.52	7.57%
jagmesh7	1138	3156	0.49%	517	450	402	90.344	14.258	496	439	8.4	6.03	11.94%
jagmesh8	1141	3162	0.49%	518	452	404	91.701	18.083	507	437	6.34	8.34	11.88%
eris1176	1176	8688	1.26%	506	576	488*	206.794	19.196	503	573	2.27	2.55	3.69%
jagmesh5	1180	3285	0.47%	536	468	409	137.536	15.871	524	445	6.83	8.21	14.43%
bcsstm27	1224	27451	3.67%	63	71	52*	92.371	9.508	58	52	2.36	9.42	21.15%
dwt_1242	1242	4592	0.60%	473	420	355	279.068	19.259	447	395	10.67	8.82	18.31%
lshp1270	1270	3699	0.46%	576	499	410	233.234	18.434	564	484	5.9	7.51	21.71%
jagmesh9	1349	3876	0.43%	618	531	449	330.484	21.045	600	501	5.85	10.6	18.26%
jagmesh6	1377	3808	0.40%	650	546	491	303.952	23.978	636	534	6.6	6.89	11.2%

8 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresenta as primeiras metaheurísticas baseadas em *algoritmo genético* (GA) aplicadas ao *problema de dominação romana perfeita* (PDRP), problema introduzido por Henning, Klostermeyer e MacGillivray em 2017 (Henning *et al.*, 2017). Foram implementados tanto um modelo clássico de GA quanto uma versão baseada em *algoritmo genético de chaves aleatórias enviesadas* (BRKGA). Além disso, o trabalho contribui com a formulação adequada de um Programa Inteiro (PI), utilizado para a obtenção de soluções exatas e para a validação dos métodos heurísticos propostos.

Como parte das contribuições experimentais, foram gerados aleatoriamente 50 grafos, sendo 26 grafos cúbicos aleatórios e outros 24 grafos aleatórios, contemplando diferentes tamanhos e densidades. Para a maioria dessas instâncias, foram calculados os valores exatos de γ_R^P, resultando em um conjunto de testes consistente que pode servir como base para pesquisas futuras. A análise comparativa considerou tanto a eficiência computacional das duas metaheurísticas baseadas em GA quanto a precisão das soluções obtidas em relação às soluções ótimas fornecidas pelo PI. Para essa avaliação, foi utilizado o gap relativo, comparando a melhor solução encontrada entre o GA e o BRKGA com as soluções exatas. Ademais, todas as implementações, bem como a documentação detalhada dos experimentos, a base de grafos e os resultados, estão disponibilizadas em um repositório público no GitHub, no endereço github.com/ErickDev1218/TCC-02.

Os resultados evidenciam que o GA e o BRKGA apresentam comportamentos complementares, com vantagens distintas a depender da base de grafos e da métrica considerada. No que se refere à qualidade das soluções (*fitness*), o GA apresentou desempenho superior na maioria das bases analisadas, em especial nos grafos aleatórios, DIMACS e Harwell–Boeing, sendo particularmente eficaz em instâncias de grande porte e baixa densidade, onde sua maior diversidade genética favorece uma exploração mais ampla do espaço de busca. A única classe em que o BRKGA se destacou de forma consistente em termos de qualidade foi a dos grafos cúbicos, sobretudo à medida que o número de vértices aumenta.

Por outro lado, em relação ao tempo de execução e à estabilidade, o BRKGA foi sistematicamente superior em todas as bases, apresentando tempos significativamente menores e menor variabilidade entre execuções, o que sugere uma convergência mais rápida e previsível.

Esses resultados indicam um claro compromisso entre qualidade e eficiência computacional: enquanto o GA tende a produzir soluções de melhor qualidade em grafos de maiores

portes e estrututa esparsa, o BRKGA é mais adequado quando rapidez e robustez são fatores prioritários. O BRKGA provavelmente converge mais rápido porque sua população elitista e o crossover enviesado reduzem drasticamente a diversidade, o que acelera a convergência, mas também aumenta o risco de estagnação em ótimos locais. Assim, a escolha do algoritmo mais apropriado depende diretamente das características estruturais do grafo e do critério de desempenho que se deseja privilegiar.

Como trabalhos futuros, pretende-se ampliar os testes das metaheurísticas, considerando um conjunto mais abrangente de parâmetros e investigando variações dos algoritmos propostos. No caso do GA, podem ser exploradas novas estratégias de seleção, cruzamento, mutação e elitismo, bem como outras abordagens estocásticas que possam aprimorar seu desempenho global. Para o BRKGA, é possível propor novas formas de decodificação das chaves aleatórias em soluções factíveis do PDRP, isto é, novos *decoders*. Adicionalmente, otimizações de implementação, como ajustes nas flags do compilador, podem ser realizadas com o objetivo de reduzir os tempos de execução e aumentar a eficiência dos métodos. Tais extensões têm o potencial de ampliar significativamente a aplicabilidade e o desempenho das metaheurísticas em uma variedade ainda maior de cenários.

REFERÊNCIAS

AGGARWAL, H.; REDDY, P. V. S. Meta-heuristic algorithms for double roman domination problem. **Applied Soft Computing**, Elsevier, v. 154, p. 111306, 2024. Disponível em: https://doi.org/10.1016/j.asoc.2024.111306. Acesso em: 04 mar. 2025.

ARQUILLA, J. "graphing"an optimal grand strategy. **Military Operations Research**, 1995. Disponível em: https://hdl.handle.net/10945/38438. Acesso em: 04 mar. 2025.

BANERJEE, S.; KEIL, J. M.; PRADHAN, D. Perfect roman domination in graphs. **Theoretical Computer Science**, Elsevier, v. 796, p. 1–21, 2019.

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. **ORSA Journal on Computing**, 1994.

BERTSIMAS, D.; WEISMANTEL, R. **Optimization Over Integers**. Belmont, MA: Dynamic Ideas, 2005.

CHAKRADHAR, P.; REDDY, P. V. S. Complexity issues of perfect roman domination in graphs. **Kyungpook Mathematical Journal**, v. 61, n. 3, p. 661–669, 2021.

CHARTRAND, G.; ZHANG, P. **A First Course in Graph Theory**. Mineola, NY: Dover Publications, 2012. ISBN 978-0-486-48368-9.

CHELLALI, M.; RAD, N. J.; SHEIKHOLESLAMI, S. M.; VOLKMANN, L. Varieties of roman domination ii. **AKCE International Journal of Graphs and Combinatorics**, Taylor & Francis, v. 17, n. 3, p. 966–984, 2020. Disponível em: https://doi.org/10.1016/j.akcej.2019.12.001. Acesso em: 10 fev. 2025.

CHELLALI, M.; RAD, N. J.; SHEIKHOLESLAMI, S. M.; VOLKMANN, L. Varieties of roman domination. In: HAYNES, T. W. *et al.* (Ed.). **Structures of Domination in Graphs**. Cham: Springer Nature Switzerland AG, 2021, (Developments in Mathematics, v. 66). Disponível em: https://doi.org/10.1007/978-3-030-58892-2_10. Acesso em: 10 fev. 2025.

COCKAYNE, E. J.; DREYER, P. A.; HEDETNIEMI, S. M.; HEDETNIEMI, S. T. Roman domination in graphs. **Discrete Mathematics**, v. 278, n. 1, p. 11–22, 2004. Disponível em: https://www.sciencedirect.com/science/article/pii/S0012365X03004473. Acesso em: 10 fev. 2025.

DARKOOTI, M.; ALHEVAZ, A.; RAHIMI, S.; RAHBANI, H. On perfect roman domination number in trees: complexity and bounds. **Journal of Combinatorial Optimization**, 2019.

DREYER, P. A. **Applications and Variations of Domination in Graphs**. Tese (Doutorado) – Rutgers, The State University of New Jersey, New Brunswick, NJ, 2000. Ph.D. Dissertation.

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

GOLDBARG, M. C.; LUNA, H. P. L. **Otimização Combinatória e Programação Linear**: Modelos e algoritmos. Rio de Janeiro: Editora Campus, 2005.

GOLDBERG, D. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Addison-Wesley, 1989. (Addison Wesley series in artificial intelligence). ISBN 9780201157673. Disponível em: https://books.google.com.br/books?id=3_RQAAAAMAAJ. Acesso em: 21 abr. 2025.

GOLDBERG, D. E.; DEB, K. A comparative analysis of selection schemes used in genetic algorithms. In: **Foundations of Genetic Algorithms**. Elsevier, 1991. v. 1, p. 69–93. Disponível em: https://doi.org/10.1016/B978-0-08-050684-5.50008-2. Acesso em: 28 mai. 2025.

GONCALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. **Journal of Heuristics**, 2011.

Gurobi Optimization, LLC. **Gurobi Optimizer Reference Manual**. 2024. Disponível em: https://www.gurobi.com. Acesso em: 22 jul. 2025.

HAGBERG, A. A.; SCHULT, D. A.; SWART, P. J. Exploring network structure, dynamics, and function using networkx. In: PYTHON IN SCIENCE CONFERENCE, 7., 2008, Pasadena. **Anais [...]**. Pasadena: SciPy, 2008. p. 11–15.

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

HENNING, M. A.; KLOSTERMEYER, W. F.; MACGILLIVRAY, G. Perfect roman domination in trees. **Discrete Applied Mathematics**, v. 236, p. 235–245, 2017. Disponível em: https://www.sciencedirect.com/science/article/pii/S0166218X17304985. Acesso em: 13 fev. 2025.

HILLIER, F. S.; LIEBERMAN, G. J. **Introdução à pesquisa operacional**. [S. l.]: McGraw Hill Brasil, 2013.

JOHNSON, D. S.; TRICK, M. A. **Cliques, Coloring, and Satisfiability**: Second dimacs implementation challenge, october 11–13, 1993. Providence, RI, USA: American Mathematical Society, 1996. v. 26. (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, v. 26). Includes benchmark graph instances widely referred to as DIMACS graphs. ISBN 9780821866092.

KHANDELWAL, K. S. A.; SARAN, G. On roman domination of graphs using a genetic algorithm. In: AL., V. S. et (Ed.). **Computational Methods and Data Engineering**. [*S. l.*]: Springer, 2021, (Advances in Intelligent Systems and Computing, v. 1227). p. 133–147.

LONDE, M. A.; PESSOA, L. S.; ANDRADE, C. E.; RESENDE, M. G. C. Biased random-key genetic algorithms: A review. **European Journal of Operational Research**, v. 321, n. 1, p. 1–22, 2025. Invited review. Disponível em: https://doi.org/10.1016/j.ejor.2024.03.030. Acesso em: 30 mai. 2025.

LOPEZ-IBANEZ, M.; DUBOIS-LACOSTE, J.; CÁCERES, L. P.; BIRATTARI, M.; STÜTZLE, T. The irace package: Iterated racing for automatic algorithm configuration. **Operations Research Perspectives**, Elsevier, v. 3, p. 43–58, 2016. Disponível em: https://doi.org/10.1016/j.orp.2016.09.002. Acesso em: 15 mai. 2025.

MANN, K.; FERNAU, H. Perfect roman domination: Aspects of enumeration and parameterization. **Algorithms**, MDPI, v. 17, n. 12, p. 576, 2024. Disponível em: https://www.mdpi.com/1999-4893/17/12/576. Acesso em: 15 fev. 2025.

MILLER, B. L.; GOLDBERG, D. E. Genetic algorithms, tournament selection, and the effects of noise. **Complex Systems**, Complex Systems Publications, v. 9, p. 193–212, 1995. Disponível em: https://www.complex-systems.com/abstracts/v09_i03_a01/. Acesso em: 28 mai. 2025.

MIRHOSEINI, S. H.; RAD, N. J. On the computational complexity aspects of perfect roman domination. **Journal of Algebraic Systems**, Shahed University, v. 10, n. 2, p. 189–202, 2023. Disponível em: https://jas.shahreza.iau.ir/article_705401.html. Acesso em: 20 abr. 2025.

NASH, J. C. The (dantzig) simplex method for linear programming. **Computing in Science & Engineering**, IEEE, v. 2, n. 1, p. 29–31, 2000.

NEMHAUSER, G. L.; WOLSEY, L. A. **Integer and Combinatorial Optimization**. New York: Wiley-Interscience, 1988.

RAJU, M. A.; REDDY, P. V. S. Metaheuristic algorithms for solving roman {2}-domination problem. **RAIRO - Operations Research**, EDP Sciences, v. 58, p. 2107–2121, 2024. Disponível em: https://doi.org/10.1051/ro/2024074. Acesso em: 26 mar. 2025.

RESENDE, M. G. C. Introdução aos algoritmos genéticos de chaves aleatórias viciadas. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 45., 2013, Natal. **Anais [...]**. Natal: SBPO, 2013. p. 3680–3691. Disponível em: http://ws2.din.uem.br/~ademir/sbpo/sbpo2013/pdf/arq0113.pdf. Acesso em: 23 abr. 2025.

REVELLE, C. S.; ROSING, K. E. Defendens imperium romanum: A classical problem in military strategy. **The American Mathematical Monthly**, Taylor Francis, Ltd., Mathematical Association of America, v. 107, n. 7, p. 585–594, 2000. Disponível em: https://www.jstor.org/stable/2589113. Acesso em: 22 abr. 2025.

SCHRIJVER, A. **Theory of Linear and Integer Programming**. New York: John Wiley & Sons, 1986.

SIVANANDAM, S. N.; DEEPA, S. N. **Introduction to Genetic Algorithms**. New York: Springer Berlin Heidelberg, 2007. ISBN 978-3-540-73189-4.

Sociedade Militar. **Exército alerta para cortes no orçamento militar do Brasil em meio à escala bélica global e risco de enfraquecimento da defesa nacional**. 2025.

SÖRENSEN, K.; GLOVER, F. Metaheuristics. In: GASS, S. I.; FU, M. C. (Ed.). **Encyclopedia of Operations Research and Management Science**. New York: Springer, 2013.

SPEARS, W. M.; JONG, K. A. D. On the virtues of parameterized uniform crossover. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 6., 1991, San Diego. **Anais [...]**. San Francisco: Morgan Kaufmann, 1991. p. 230–236.

STEWART, I. Defend the romam empire! **Scientific American**, v. 281, p. 136–138, 1999. Disponível em: https://www.jstor.org/stable/10.2307/26058532. Acesso em: 10 fev. 2025.

TOSO, R. F.; RESENDE, M. G. C. A c++ application programming interface for biased random-key genetic algorithms. **Optimization Methods and Software**, Taylor & Francis, 2014. Disponível em: https://doi.org/10.1080/10556788.2014.890197. Acesso em: 18 set. 2025.

WOLSEY, L. A. **Integer Programming**. New York: John Wiley & Sons, 1998.

YUE, J.; SONG, J. Note on the perfect roman domination number of graphs. **Applied Mathematics and Computation**, Elsevier, v. 364, p. 124685, 2020.