UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE SOBRAL

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

FIAMA CARLA MARTINS DE SOUSA

O DESAFIO DOS NÚMEROS DE RAMSEY: EXPLORANDO A EFICIÊNCIA DE ALGORITMOS ALEATÓRIOS E GENÉTICOS NA BUSCA POR CONTRAEXEMPLOS

SOBRAL

2025

FIAMA CARLA MARTINS DE SOUSA

O DESAFIO DOS NÚMEROS DE RAMSEY: EXPLORANDO A EFICIÊNCIA DE ALGORITMOS ALEATÓRIOS E GENÉTICOS NA BUSCA POR CONTRAEXEMPLOS

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

Orientador: Prof. Dr. Antônio Josefran de O. Bastos.

SOBRAL

2025

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)

S696d Sousa, Fiama Carla Martins de.
O Desafio dos Números de Ramsey: Explorando a Eficiência de Algoritmos Aleatórios e Genéticos na Busca por Contraexemplos / Fiama Carla Martins de Sousa. – 2025.
53 f.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Sobral, Curso de Engenharia da Computação, Sobral, 2025.
Orientação: Prof. Dr. Antônio Josefran de O. Bastos.

1. Grafos. 2. Ramsey. 3. Heurística. I. Título.

CDD 621.39

FIAMA CARLA MARTINS DE SOUSA

O DESAFIO DOS NÚMEROS DE RAMSEY: EXPLORANDO A EFICIÊNCIA DE ALGORITMOS ALEATÓRIOS E GENÉTICOS NA BUSCA POR CONTRAEXEMPLOS

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

Aprovada em: 14 de Março de 2025

BANCA EXAMINADORA

Prof. Dr. Antônio Josefran de O. Bastos (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Yuri Victor Lima de Melo
Universidade Federal do Ceará - UFC

Mestranda Vanessa Carvalho do Nascimento
Universidade Federal do Ceará - UFC

A Deus.

À minha mãe, Irene Clara.

AGRADECIMENTOS

Agradeço imensamente ao meu orientador, Dr. Antonio Josefran de O. Bastos, que, com paciência e dedicação, me acompanhou ao longo de toda a realização deste trabalho. Sua orientação foi fundamental para o meu crescimento acadêmico e pessoal. Agradeço por ter me instigado a persistir, por suas palavras de sabedoria e, sobretudo, por desmistificar o TCC, tornando essa jornada mais tranquila e menos temida do que muitos imaginam. Sua capacidade de guiar com clareza e dedicação fez toda a diferença em minha formação.

Expresso também minha profunda gratidão à Professora Dra. Jermana Lopes de Morais, que com excelência, me auxiliou na construção e revisão da minha monografia, tornando esse processo mais simples e acessível. Seu apoio e sua competência foram essenciais para que eu conseguisse superar essa etapa com êxito.

Agradeço à Mestranda Vanessa Carvalho do Nascimento e ao Professor Yuri Victor Lima de Melo, pela disponibilidade e generosidade em aceitar o convite para integrar a banca avaliadora deste trabalho. Suas contribuições serão sempre lembradas com grande respeito e apreço.

Por fim, agradeço à Universidade Federal do Ceará e a todos os professores, colegas e funcionários que, de alguma forma, fizeram parte da minha trajetória acadêmica. Cada um desempenhou um papel importante na minha formação, e sou imensamente grata por cada experiência vivida e aprendizado adquirido ao longo dessa caminhada.

"Imagine um universo no qual o número de Ramsey é pequeno. São números que conhecemos o existir, mas não temos ideia de como calculá-los." (HOFFMAN, 1988, p. 80.)

RESUMO

O presente trabalho aborda conceitos fundamentais da Teoria dos Grafos, incluindo grafos completos, grafos vazios e o conceito de complemento de grafo. Revisa conceitos essenciais, como o grau de um vértice, vizinhança e destaca a importância do Teorema de Turán na definição dos limites de arestas para evitar subgrafos completos. Detalha o problema original de Ramsey e a sua formulação moderna. Destaca a dificuldade de encontrar números exatos de Ramsey e a contribuição de Paul Erdős para a teoria. O objetivo principal desse estudo é explorar a aplicação de heurísticas, especificamente algoritmo genético e aleatório, para encontrar contraexemplos para o problema de Ramsey. O estudo busca analisar como essas abordagens podem ser utilizadas para resolver problemas complexos em Teoria dos Grafos, focando na determinação de grafos que satisfaçam determinadas propriedades de Ramsey. Conclui que a utilização de algoritmos genéticos é uma abordagem viável para encontrar contraexemplos ao problema de Ramsey, apesar das limitações e desafios computacionais associados.

Palavras-chave: grafos; Ramsey; algoritmo genético, Combinatória

ABSTRACT

This work addresses fundamental concepts of Graph Theory, including complete graphs, empty graphs, and the concept of graph complement. It reviews essential concepts, such as the degree of a vertex, neighborhood, and highlights the importance of Turán's Theorem in defining edge bounds to avoid complete subgraphs. It details the original Ramsey problem and the modern formulation. It highlights the difficulty of finding exact Ramsey numbers and the contribution of Paul Erdős to the theory. The main objective of this study is to explore the application of heuristics, specifically genetic and random algorithms, to find counterexamples to the Ramsey problem. The study seeks to analyze how these approaches can be used to solve complex problems in Graph Theory, focusing on the determination of graphs that satisfy certain Ramsey properties. It concludes that the use of genetic algorithms is a viable approach to find counterexamples to the Ramsey problem, despite the associated limitations and computational challenges.

Keywords: graphs; Ramsey; algorithm, Combinatorics

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SÍMBOLOS

$V(G)$	Conjunto de vértices do grafo G
$E(G)$	Conjunto de arestas do grafo G
$e = \{u, v\}$	Representação de uma aresta conectando u e v
uv	Forma simplificada de representar uma aresta entre u e v
K_n	Grafo completo com n vértices
$R(k, l)$	Número de Ramsey
$G - H$	Diferença de grafos
$d(v)$	Grau do vértice v
$N(v)$	Conjunto de vizinhos do vértice v
$\mathbb{N}, \mathbb{Z}, \mathbb{R}$	Conjuntos numéricos
$\cup, \cap$	União e interseção
$\subseteq, \supseteq$	Subconjunto e superconjunto
$\rightarrow, \Rightarrow, \iff$	Implicação e equivalência lógica
$\max, \min$	Funções máximo e mínimo

SUMÁRIO

1 INTRODUÇÃO

A Teoria dos Grafos é uma área da matemática que se dedica ao estudo de grafos, estruturas compostas por vértices e arestas. Essa teoria possui ampla aplicabilidade em diversas áreas, como redes de computadores, biologia, logística e telecomunicações (WEST, 2001). Problemas relacionados a grafos envolvem questões combinatórias que demandam métodos matemáticos e computacionais sofisticados para sua resolução.

Dentre os diversos problemas que emergem no estudo dos grafos, um dos mais notáveis é o problema de Ramsey, que ocupa um lugar de destaque devido à sua complexidade e às implicações teóricas que oferece para diversas subáreas da matemática discreta e da computação (DIESTEL, 2017). A Teoria de Ramsey, originalmente proposta por Frank P. Ramsey na década de 1930, investiga a existência de padrões específicos em grafos, como cliques ou conjuntos independentes de determinado tamanho. Um clique é um subconjunto de vértices onde todos estão conectados entre si, enquanto um conjunto independente é um subconjunto de vértices onde nenhum par de vértices está conectado (GRAHAM *et al.*, 1990). A questão central dessa teoria busca determinar o menor número de vértices necessários para garantir que qualquer coloração das arestas de um grafo contenha necessariamente um desses padrões estruturais.

Apesar de seu conceito ser de fácil compreensão, a resposta para o problema de Ramsey torna-se extremamente desafiadora conforme o espaço de soluções aumenta, devido à natureza exponencial do problema. Atualmente, apenas alguns valores exatos são conhecidos, sendo que a maioria dos resultados envolve apenas limites superiores e inferiores para diferentes pares de parâmetros k e s. Esse comportamento sugere que, à medida que os grafos aumentam de tamanho, a exploração exaustiva de todas as possibilidades torna-se inviável (RADZISZOWSKI, 2024).

Diante dessa dificuldade, métodos heurísticos e algoritmos de otimização surgem como alternativas viáveis para a busca de contraexemplos. Tais algoritmos permitem explorar grandes espaços de busca de maneira eficiente, reduzindo significativamente o tempo de execução e aumentando as chances de identificar padrões que violem as condições teóricas. Um exemplo disso é o trabalho *Subgraph Counting Identities and Ramsey Numbers* (MCKAY; RADZISZOWSKI, 1997), que utiliza técnicas que combinam Teoria Combinatória com algoritmos computacionais para explorar as propriedades estruturais de grafos e identificar padrões relacionados ao problema de Ramsey, demonstrando a eficácia de abordagens algorítmicas na resolução de problemas combinatórios complexos.

Nesse contexto, a presente pesquisa se concentra na busca por contraexemplos ao problema de Ramsey e, para atingir esse objetivo, serão utilizados três algoritmos principais: o algoritmo aleatório, o algoritmo genético e uma otimização do algoritmo genético baseada na estruturação dos grafos em células e alelos. Tais algoritmos são reconhecidos pela capacidade de lidar com problemas de alta complexidade e de convergir rapidamente para soluções plausíveis (MELANIE, 1998).

Além disso, a pesquisa propõe a análise detalhada das técnicas empregadas, assim como a avaliação do desempenho desses algoritmos na busca por contraexemplos. Os resultados obtidos serão discutidos de maneira a destacar a relevância dessas metodologias para o avanço do estudo do problema de Ramsey e demonstrar o potencial das heurísticas como ferramentas eficientes para a solução de problemas combinatórios complexos.

2 OBJETIVOS

2.1 Objetivo Geral

Avaliar a eficácia de métodos computacionais utilizando algoritmos, como algoritmos aleatórios e genéticos, na identificação e validação de contraexemplos no contexto da Teoria de Ramsey, considerando sua capacidade de lidar com a alta complexidade e de explorar grandes espaços de busca de maneira eficiente.

2.2 Objetivos Específicos

- Avaliar os contraexemplos descobertos verificando sua aderência aos conceitos estabelecidos, como a formação de cliques e conjuntos independentes;
- Analisar como otimizações no algoritmo genético podem impactar na identificação de contraexemplos;
- Identificar e analisar aplicações práticas da Teoria de Ramsey em diversas áreas, como matemática discreta, Ciência da Computação e Teoria dos Grafos.

3 FUNDAMENTAÇÃO TEÓRICA

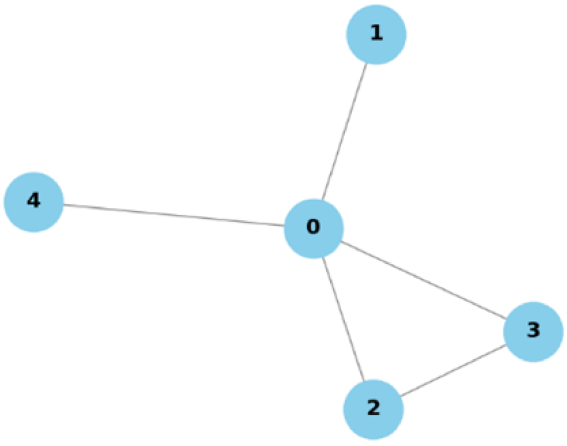
3.1 Introdução à Teoria dos Grafos

Um grafo G é uma estrutura composta por um conjunto de vértices $V(G)$ e um conjunto de arestas $E(G)$ que conectam pares de vértices distintos (BOTLER *et al.*, 2021). Uma aresta $e \in E(G)$ é representada por $e = \{u, v\}$, ou simplesmente uv, quando interliga dois vértices u e v de V. Um vértice universal é um vértice que está conectado a todos os outros vértices do grafo.

Em um grafo simples, não há laços (arestas que conectam um vértice a si mesmo) nem múltiplas arestas entre o mesmo par de vértices. Dois vértices ligados por uma aresta são denominados vizinhos ou adjacentes. Se não houver uma aresta entre eles, diz-se que são não adjacentes. Um conjunto de vértices sem arestas entre si é chamado de conjunto independente (WEST, 2001).

O grau de um vértice v, chamado de $d_G(v)$, é a quantidade de vizinhos de v, enquanto o conjunto dos vizinhos de v é denotado por $N_G(v)$. A Figura 1, ilustra a representação usual de um grafo, em que cada vértice é representado por um ponto e dois pontos são conectados por uma linha sempre que os vértices correspondentes formam uma aresta (BONDY; MURTY, 2008).

Figura 1 – Grafo em V = {0, ... ,5} com arestas definidas em E = { (0,1), (0,2), (0,3), (0,4), (2,3)}

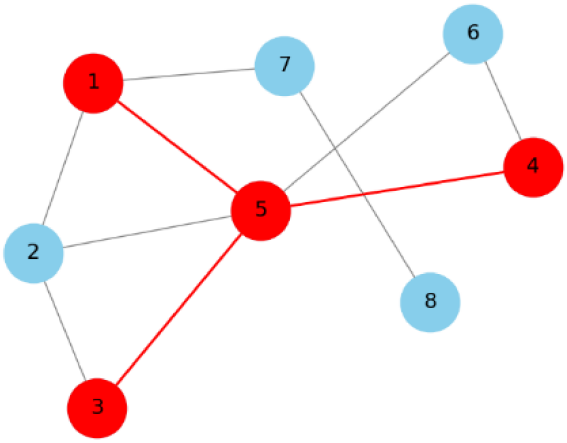


Fonte: Próprio autor

Apresentado em vermelho na Figura 2, temos um subgrafo H que é definido como um conjunto de vértices e arestas que são extraídos de um grafo maior G, em azul, preservando a estrutura de conectividade entre os vértices selecionados, assim, um subgrafo contém o

subconjunto dos vértices e das arestas de um grafo maior (DIESTEL, 2017).

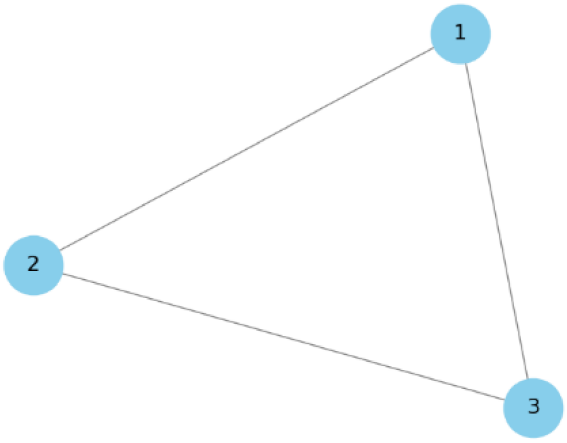
Figura 2 – Subgrafo H (em vermelho) extraído de um grafo G (em azul)



Fonte: Próprio autor

Os dois grafos mais simples são o grafo completo, também chamado de clique, em que todos os pares de vértices são arestas; e o grafo vazio, também chamado de conjunto independente, que não possui nenhuma aresta (BONDY; MURTY, 2008). O grafo completo com n vértices é denotado por K_n. Na Figura 3 é apresentado um exemplo de um grafo completo K_3, em que temos 3 vértices e 3 arestas, formando um triângulo.

Figura 3 – Exemplo de um grafo completo K_3

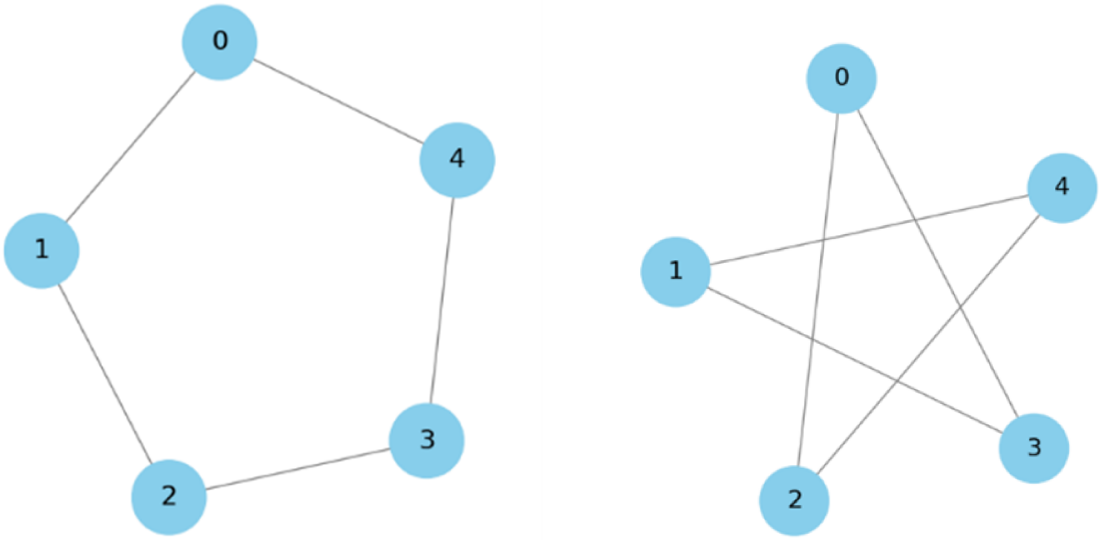


Fonte: Próprio autor

O complemento de um grafo simples G, denotado por $\bar{G}$, é o grafo simples que possui o mesmo conjunto de vértices de G, de modo que dois vértices são adjacentes em $\bar{G}$, se e somente se não são adjacentes em G. De tal forma que para encontrar o complemento de um grafo, se usa o mesmo conjunto de vértices, mas se preenche com as arestas que faltam no grafo

original (BOTLER *et al.*, 2021). Na Figura 4 é apresentado um exemplo de um grafo simples e um grafo complementar.

Figura 4 – Comparativo de grafo simples (a) e grafo complementar (b)



(a) Grafo simples G (b) Grafo complementar

Fonte: Próprio autor

A coloração de arestas é outro conceito muito útil na análise de grafos quando se busca identificar padrões ou propriedades específicas. Esse processo envolve atribuir cores às arestas de um grafo. Um grafo cujas arestas são coloridas com a mesma cor é chamado de monocromático. Em situações em que a coloração é realizada com duas cores pode-se considerar que uma das cores simboliza a presença de uma aresta entre dois vértices, enquanto a outra cor representa a ausência dessa conexão (DIESTEL, 2017).

Além disso, a Teoria dos Grafos inclui o Teorema de Turán, que é fundamental para vários problemas de otimização e análise combinatória. Esse teorema estabelece que para um grafo simples G com n vértices, o número máximo de arestas que esse grafo pode ter sem conter um subgrafo completo K_{k+1} é dado por $\left(1-\frac{1}{k}\right)\cdot\frac{n^2-x^2}{2}+\frac{x\cdot(x-1)}{2}$, sendo x o resíduo da divisão de n por k, o que é importante para ajustar a distribuição dos vértices em subconjuntos no grafo, caso a divisão não seja exata, para garantir que todos tenham tamanhos tão uniformes quanto possível. A primeira parte da fórmula, $\left(1-\frac{1}{k}\right)\cdot\frac{n^2-x^2}{2}$, calcula a contribuição principal, baseada na distribuição uniforme dos vértices enquanto a segunda parte, $\frac{x\cdot(x-1)}{2}$, ajusta o número de arestas para o caso em que a divisão dos vértices não é exatamente uniforme. A função retorna o máximo de arestas que o grafo pode ter sem conter um subgrafo K_k (BOLLOBÁS, 2001).

Grafos são utilizados para modelar uma ampla variedade de sistemas em diversas

disciplinas, incluindo redes de computadores, biologia, logística, entre outras áreas. Para mais informações sobre os conceitos e aplicações da Teoria de Grafos, sugerimos a leitura de (DIESTEL, 2017).

3.2 Teoria de Ramsey

O problema de Ramsey tem suas raízes em um artigo publicado em 1930 pelo matemático britânico Frank P. Ramsey, intitulado "On a Problem of Formal Logic". Em sua forma mais simples, o problema é frequentemente apresentado da seguinte maneira: considere um grupo de pessoas em que cada pessoa pode conhecer ou não conhecer a outra; qual é o menor tamanho que esse grupo deve ter para que pelo menos três dessas pessoas se conheçam mutuamente ou para que pelo menos três delas não se conheçam? A Teoria de Ramsey inicia-se pela generalização deste enunciado para grafos. Traduzindo essa assertiva para a linguagem de Teoria de Grafos teremos: qual a menor quantidade de vértices que um grafo deve ter para que haja um clique ou um conjunto independente de tamanho três (DIESTEL, 2017).

Os números de Ramsey podem ser assim definidos: fixados inteiros positivos k e l, define $R(k,l)$ como sendo o menor inteiro tal que para qualquer grafo completo K_n com $n \geq R(k,l)$, cujas arestas são coloridas com duas cores, sempre existe um subgrafo completo K_k monocromático em uma das cores (clique) ou um subgrafo completo K_l na outra cor (conjunto independente) (DIESTEL, 2017).

Paul Erdős foi um renomado matemático do século XX, um dos fundadores da Teoria dos Grafos moderna, conhecido por sua vasta contribuição em várias áreas da matemática dentre elas a Combinatória, a Teoria dos Números e a Teoria da Probabilidade (Hoffman, 1998). Fez contribuições significativas para essa área, como a introdução do conceito dos números de Ramsey além de alguns limites superiores e inferiores para eles (ROSTA, 2004).

A teoria desenvolvida por Erdős avançou a compreensão do problema de Ramsey, permitindo abordagem mais técnica para a resolução desse problema. Retomando a pergunta inicial, a resposta é dada pelo caso específico de $R(3,3) = 6$. Isso significa que para um grafo com cinco vértices a condição imposta não é satisfeita (RADZISZOWSKI, 2024).

O progresso na determinação exata dos números de Ramsey tem sido desafiador ao longo do tempo, pois a complexidade de um grafo aumenta drasticamente conforme é adicionado vértices a ele, uma vez que o número de maneiras de colorir as arestas de um grafo cresce exponencialmente com o número de vértices (GRAHAM *et al.*, 1990). Para casos pequenos,

como um grafo de seis vértices, o espaço de busca já envolve 2^{15} possibilibilades. Embora esse número seja relativamente pequeno para um computador, fazer isso manualmente é inviável. Para um grafo com 40 vértices, o número de arestas é dado por $\frac{40 \cdot 39}{2} = 780$ arestas. Como cada aresta pode ser colorida de duas formas, o total de possibilidades cresce exponencialmente, resultando em 2^{780} formas de aplicação de duas cores.

Esse é um problema computacionalmente intensivo, pois envolve verificar todas as possíveis colorações das arestas no grafo e muitos dos valores exatos de $R(k,l)$, mesmo para números pequenos como o $R(5,5)$, são desconhecidos. Com o advento da computação moderna, os pesquisadores começaram a usar computadores para ajudar na determinação desses números e através do emprego de algoritmos de computador foram obtidos progressos consideráveis nesta área. Contudo, ainda não se conseguiu determinar esses valores com precisão. (RADZISZOWSKI, 2024).

Além disso, à medida que os valores de Ramsey aumentam, a discrepância entre as cotas inferiores e superiores cresce rapidamente, como mostrado na Tabela 1. Ao contrário de muitos problemas matemáticos em que padrões ou fórmulas podem ser encontrados, os números de Ramsey não seguem padrões simples e não possuem generalizações diretas.

Essa divergência torna ainda mais difícil estabelecer limites confiáveis para os números de Ramsey, dificultando até mesmo uma estimativa precisa da função. Isso justifica a busca por heurísticas e abordagens computacionais para melhorar os limites conhecidos.

O Teorema de Ramsey mostra que a desordem total é impossível, pois garante que, em qualquer partição de um grafo suficientemente grande, uma subestrutura ordenada sempre existirá (ROSTA, 2004). Um maneira de determinar os limites precisos em que essas subestruturas começam a surgir é encontrando os contraexemplos, uma situação que refuta uma proposição ou uma hipótese.

Uma abordagem interessante para auxiliar na busca de contraexemplos de números de Ramsey é utilizar o Teorema de Turán para avaliar os grafos no pré-processamento do algoritmo, pois ele fornece uma condição sobre a quantidade máxima de arestas que um grafo pode ter sem conter um subgrafo completo de determinado tamanho e assim reduzir o espaço de busca do algoritmo (BOLLOBÁS, 2001).

Tabela 1 – Valores e limites para $R(k,l)$, $k \leq 10, l \leq 15$

l / k	3	4	5	6	7	8	9	10	11	12	13	14	15
3	6	9	14	18	23	28	36	40 42	47 50	53 59	60 68	67 77	74 87
4		18	25	36 41	48 61	59 84	73 115	92 149	102 191	128 238	138 291	147 349	158 417
5			43 48	58 87	80 143	101 216	133 316	149 442	183 633	203 848	233 1138	267 1461	275 1878
6				102 165	115 298	134 495	183 780	204 1171	262 1804	294 2566	347 3703	5033	401 6911
7					205 540	219 1030	252 1713	292 2826	405 4553	417 6954	511 10578	15263	22112
8						282 1870	329 3583	343 6090	457 10630	16944	817 27485	41525	873 63609
9							565 6588	581 12677	22325	38832	64864		
10								798 23556	45881	81123			1313

Fonte: (RADZISZOWSKI, 2024)

3.3 Busca por Contrexemplos

Na Teoria dos Grafos, as heurísticas são especialmente importantes para resolver problemas de otimização, uma vez que são projetadas para explorar grandes espaços de busca evitando a necessidade de uma busca exaustiva, já que podem ser ajustadas para incorporar conhecimento específico sobre o problema, melhorando sua eficácia. Em vez de tentar todas as possibilidades, uma heurística procura soluções que sejam boas o suficiente com base em certos critérios (MELANIE, 1998). Dada a complexidade do problema de Ramsey, torna-se essencial a utilização de métodos eficientes para explorar o espaço de soluções possíveis, ainda que sem garantir a otimização perfeita.

Neste trabalho foi implementado três algoritmos distintos para encontrar contraexemplos para os números de Ramsey: O algoritmo aleatório que utiliza simplesmente força bruta; o algoritmo genético que se baseia no processo de evolução natural, que apesar de apresentar algum nível de refinamento, ainda é ineficiente; e o algoritmo genético modificado que divide o grafo em células de tamanho r, buscando preservar determinadas estruturas do grafo original.

3.3.1 Algoritmo Aleatório

O uso de algoritmos aleatórios é uma abordagem comum em diversos problemas devido à sua simplicidade e eficiência. Essa técnica permite gerar soluções de forma aleatória, e, caso nenhuma solução seja encontrada após a avaliação de todas as combinações possíveis, conclui-se que não há uma solução dentro do conjunto de possibilidades definidas.

Diferente de alguns métodos heurísticos que podem convergir para uma solução aceitável ao longo do tempo, algoritmos aleatórios não têm um mecanismo de convergência, de modo que podem continuar explorando o espaço de solução indefinidamente sem se aproximar de uma solução, o que exige um número extremamente alto de iterações que se traduz em um custo computacional alto, tanto em termos de tempo quanto de recursos. Para aprofundar-se nesse tema, recomenda-se a leitura do livro "Algoritmos: Teoria e Prática"de (THOMAS *et al.*, 2002), que discute a aplicação de algoritmos aleatórios e suas implicações em termos de eficiência e complexidade computacional.

3.3.2 Algoritmo Genético

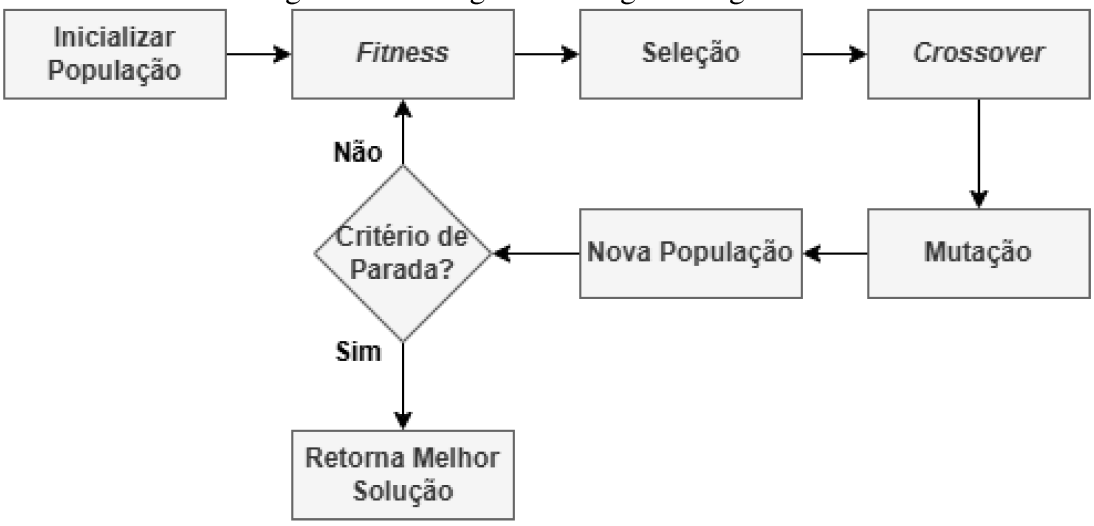
O algoritmo genético é uma técnica de otimização baseada no processo de evolução natural. Seu funcionamento se dá por meio da simulação dos processos evolutivos como seleção, cruzamento e mutação para encontrar soluções ótimas ou satisfatórias. Embora seja capaz de lidar com espaço amplo de busca, além de ser flexível e aplicável a diversos problemas, o algoritmo genético não garante que a solução seja encontrada (MELANIE, 1998).

O funcionamento básico desse algoritmo, delineado na Figura 5 se dá por meio da criação de uma população inicial de soluções (indivíduos), representada como um cromossomo, que é normalmente uma cadeia de bits. Cada indivíduo da população é avaliado com base em uma função de aptidão (*fitness*), que mede a qualidade da solução que ele representa. O objetivo do algoritmo é maximizar ou minimizar essa função com base nas regras do problema abordado. Soluções melhores recebem valores de aptidão mais altos, enquanto soluções piores recebem valores mais baixos de modo que os indivíduos com melhores valores de aptidão têm uma probabilidade maior de serem selecionados para reprodução (*crossover*) (GOLDBERG, 1989).

O operador de seleção é um componente essencial de algoritmos genéticos. A literatura identifica cinco principais mecanismos de seleção: proporcional, por torneios, com truncamento, por normalização linear e por normalização exponencial. Após a seleção, indivíduos

(pais) são escolhidos para combinar seus cromossomos, gerando novos indivíduos (filhos). Isso simula a troca de material genético na reprodução biológica, uma vez que os filhos herdarão as características dos pais (PACHECO, 1999).

Figura 5 – Fluxograma do algoritmo genético



Fonte: Próprio autor

Após o *crossover* e a mutação, a nova geração de indivíduos substitui a antiga. Pode-se substituir todos os indivíduos da população anterior ou apenas os menos aptos, e o processo se repete até que um critério de parada seja alcançado. Esse critério pode ser atingir um número máximo de gerações, encontrar uma solução que satisfaça um nível mínimo de aptidão, ou então, parar quando não houver mais melhora significativa na aptidão (MELANIE, 1998).

Para uma compreensão aprofundada acerca de algoritmos genéticos, incluindo seus princípios de funcionamento, operadores genéticos e aplicações práticas, recomenda-se a leitura do artigo “Algoritmos Genéticos: Princípios e Aplicações” de Mauro A. C. Pacheco. Este material oferece uma visão detalhada sobre o tema.

3.3.3 Algoritmo Genético com Células e Alelos

O algoritmo genético com células e alelos foi desenvolvido buscando preservar a estrutura dos grafo durante as operações genéticas, evitando alterações abruptas. Para representar melhor os grafos, em vez de tratar diretamente as arestas do grafo, a técnica divide os vértices em células, cada uma contendo um subconjunto fixo de vértices de tamanho r. Além das células estruturadas, há também a presença da "caspa", que representa os vértices que não se encaixam perfeitamente na divisão celular, garantindo que sua influência na conectividade geral do grafo

seja devidamente avaliada.

A utilização de alelos para representar as conexões entre essas células foi projetada para facilitar o processo de recombinação genética, tornando o *crossover* e a mutação mais estruturados durante a recombinação genética, evitando combinações aleatórias entre os grafos, que poderiam comprometer a convergência do algoritmo. Esses alelos armazenam todas as possíveis formas de ligação entre os vértices de diferentes células e caspa. Para gerenciar essas informações, o algoritmo constrói uma biblioteca de alelos, que contém todas as possíveis configurações de conexões e também gera uma biblioteca com todas as possíveis configurações de células. Essas matrizes servem como base para as operações de *crossover* e mutação proporcionando uma evolução mais organizada e estratégica dos grafos durante o processo genético.

A combinação das células entre dois grafos é realizada aleatoriamente, seguindo uma probabilidade p para determinar a origem de cada célula. Após a seleção da célula de cada pai, os alelos dentro desses grafos são avaliados quanto ao seu *fitness*, e o alelo com o menor *fitness* é escolhido. Esse critério é utilizado para garantir que as conexões mais favoráveis sejam transmitidas para os descendentes. Essa abordagem reduz a complexidade da manipulação dos grafos, organizando-os em componentes menores que podem ser combinados de maneira controlada.

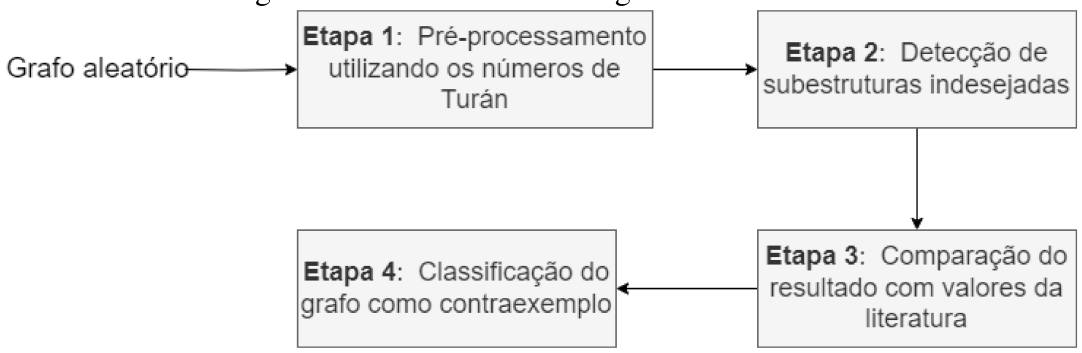
4 METODOLOGIA PROPOSTA

Fundamentado nas limitações encontradas na literatura sobre heurísticas para o problema de Ramsey, desenvolvemos três abordagens distintas para a busca de contraexemplos: *Algoritmo Aleatório*, *Algoritmo Genético* e *Algoritmo Genético Modificado*. O objetivo de cada um desses algoritmos é verificar a propriedade de Ramsey em grafos aleatórios e encontrar contraexemplos de maneira eficiente, com diferentes estratégias de otimização e busca. O código-fonte desenvolvido para os algoritmos apresentados neste trabalho pode ser acessado no repositório GitHub: https://github.com/Fiama-Carla/Ramsey.git.

Para nosso estudo, estamos interessados principalmente na quantidade média de tentativas necessárias para encontrar um grafo que satisfaça a propriedade de Ramsey, assim como no tempo médio de execução do algoritmo. Além desses dados, outras métricas relevantes incluem o tempo máximo e mínimo de execução que é importante para entender a variabilidade do algoritmo, a frequência de sucesso, comparar o desempenho para diferentes valores de *R(k,s)* que permite analisar como essas variações influenciam o número de tentativas e o tempo de execução.

O algoritmo aleatório gera grafos aleatórios e verifica se eles atendem às condições do problema de Ramsey seguindo o delineamento apresentado na Figura 6, o qual consiste basicamente em 4 etapas: 1) pré-processamento dos grafos utilizando os números de Turán; 2) detecção de subestruturas indesejadas a partir da verificação da presença de vértices universais; 3) análise dos vizinhos de v e seu grau d_v com base nos números conhecidos de Ramsey; e 4) classificação do grafo como contraexemplo.

Figura 6 – Delineamento do algoritmo aleatório

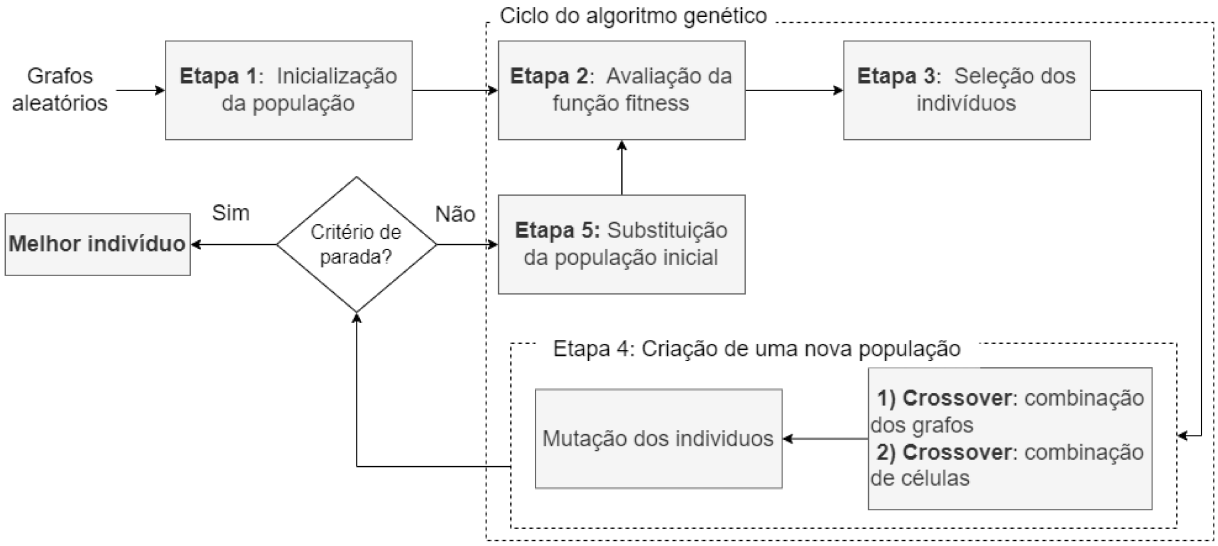


Fonte: Próprio autor

De outro modo, o algoritmo genético evolui uma população de grafos, aplicando operadores de *crossover* e mutação para melhorar a qualidade das soluções. O delineamento

do algoritmo genético é apresentado na Figura 7, o qual consiste basicamente em 5 etapas principais: 1) inicialização da população; 2) avaliação de *fitness*; 3) seleção; 4) criação de uma nova população; e 5) substituição da população. O algoritmo genético modificado segue a mesma estrutura básica do algoritmo genético tradicional, mas com uma modificação na etapa de *crossover*. Em vez de tratar o grafo como um todo, o algoritmo genético modificado opera sobre *células* e *alelos*. Cada célula no grafo representa um conjunto de vértices que podem ser conectados de diferentes formas, e cada alelo representa uma possível conexão entre essas células.

Figura 7 – Delineamento do algoritmo genético



Fonte: Elaborado pelo autor.

4.1 Etapas de Desenvolvimento do Algoritmo Aleatório

O algoritmo foi desenvolvido utilizando a linguagem de programação Python e suas quatro etapas de desenvolvimento, elencadas na Figura 6, são detalhadas nas subseções a seguir.

4.1.1 Etapa 1: Pré-Processamento

Os grafos utilizados foram criados por meio do Algoritmo 1 - $criar\_grafo(n,k,s,tempo\_limite)$, que utiliza o modelo de Erdős-Rényi $G(n,p)$ para gerar grafos aleatórios com n vértices, onde cada aresta tem uma probabilidade p de existir independentemente das outras. A escolha de $\frac{1}{2}$ implica que cada aresta é criada com igual chance de estar presente ou ausente, o que é uma estratégia equilibrada, pois mantém a geração aleatória sem favorecer muito cliques ou conjuntos

independentes, permitindo uma busca eficiente. Esse processo é repetido até que um grafo que satisfaça a propriedade de Ramsey seja encontrado ou até que o tempo limite seja atingido.

Algoritmo 1: Criar grafo aleatório

```
/* Função criar_grafo(n, k, s, tempo_limite)                    */
while True do
    G ← erdos_renyi(n, 0.5);
    if verifica(G, k, s) is true then
        return G;
    end
    if tempo_execucao > tempo_limite then
        imprimir("Falha em encontrar um grafo") ;
    end
end
```

Os grafos criados são avaliados utilizando o Algoritmo 2 - $turan(k,n)$, que implementa a função Turan, o que reduz o espaço de busca, e economiza tanto tempo de processamento quanto memória. A função só faz sentido se $k \geq 2$, já que um clique K_1 ou K_0 não é relevante em contextos típicos de grafos.

Algoritmo 2: Função Turán

```
/* Função turan(k, n)                                           */
if r ≥ 2 then
    x ← n  mod k;
    turan ← (1 − 1/k) · (n² − x²)/2 + x·(x−1)/2;
else
    turan ← 0;
end
return turan;
```

Em seguida, o Algoritmo 3 - $verifica(grafo,k,s)$, compara o número de arestas do grafo e o número de arestas do grafo complementar com os números de Turán, se as condições forem verdadeiras, isso significa que o grafo pode ser um possível contraexemplo.

Algoritmo 3: Verificação de condição com Turán

```
/* Função verifica(grafo, k, s)                                    */
if E(G) > turan(k−1, E(G)) then
    return false;
end
if E(Ḡ) > turan(s−1, E(G)) then
    return false;
end
```

4.1.2 Etapa 2: Detecção de Subestruturas Indesejadas

A função central do Algoritmo 4 - $verifica(grafo, k, s)$, recebe como parâmetros um grafo, um valor k, tamanho da clique indesejada, e um valor s, tamanho do conjunto independente indesejado. Inicialmente é verificado para cada vértice $v \in V(G)$ do grafo se é um vértice que está conectado a todos os outros vértices. Nesse caso, a função *verifica* é chamada recursivamente com o subgrafo $G[N(v)]$ de modo a verificar se eles estão conectados entre si. A mesma lógica é implementada para encontrar os conjuntos independentes.

Algoritmo 4: Iteração sobre os vértices de G

```
/* Função verifica(grafo, k, s)                                    */
foreach v ∈ V(G) do
    if d(v) = V(G) − 1 then
        return verifica(G[N(v)], k−1, s);
    end
    if d(v) = 0 then
        return verifica(G[N(v)], k, s−1);
    end
end
```

4.1.3 Etapa 3: Comparação do Resultado com a Literatura

Em seguida, é necessário validar o grafo que está sendo avaliado, portanto um laço de repetição no Algoritmo 5 - $verifica(grafo, k, s)$, itera sobre cada vértice v, de modo a verificar se $v \geq R(k-1, s)$, o que significa que $G[N(v)]$ possui um clique de tamanho $k-1$, que juntamente com v formam um clique de tamanho k; ou $G[N(v)]$ possui um conjunto independente de tamanho

$s-1$, que juntamente com v formam um conjunto independente de tamanho s. Seguindo a mesma lógica, é verificado se cada vértice forma um conjunto independente. Caso o grafo possua alguma dessas configurações o código retorna *false*, pois o grafo avaliado não é um contraexemplo e retorna para a etapa 1, caso contrário segue para a próxima verificação.

Algoritmo 5: Iteração sobre os vértices de G

```
/* Função verifica(grafo, k, s)                                    */
foreach v ∈ V(G) do
    if d(v) ≥ R(k−1,s) then
        return false;
    end
    if V(G) − d(v) − 1 ≥ R(k,s−1) then
        return false;
    end
end
```

4.1.4 Etapa 4: Classificação do Grafo como Contraexemplo

O Algoritmo 6 - $verifica(grafo, k, s)$, verifica se o subgrafo induzido pelos vizinhos de v e o subgrafo induzido pelos não vizinhos de v satisfazem as etapas anteriores. Se qualquer uma das verificações falhar, então o grafo não é um contraexemplo. Se todas as verificações passarem para todos os vértices avaliados, então o algoritmo encontrou um contraexemplo para Ramsey e a função retorna *verdadeiro*.

Algoritmo 6: Verificação sobre os vizinhos de v

```
/* Função verifica(grafo, k, s)                                    */
foreach v ∈ V(G) do
    if !verifica(G[N(v)],k−1,s) then
        return false;
    end
    if !verifica(G[N̄(v)],k,s−1) then
        return false;
    end
end
return true;
```

4.2 Etapas de Desenvolvimento do Algoritmo Genético

O algoritmo foi desenvolvido utilizando a linguagem de programação Python e é importante destacar que a principal diferença entre o algoritmo genético tradicional e a abordagem aprimorada proposta neste trabalho reside no método de cruzamento dos grafos. Enquanto o algoritmo genético tradicional utiliza técnicas convencionais de *crossover*, a segunda abordagem introduz um mecanismo específico para combinar as estruturas dos grafos, preservando características que aumentam a probabilidade de encontrar contraexemplos ao problema de Ramsey. As cinco etapas de desenvolvimento, elencadas na Figura 7, são detalhadas nas subseções a seguir.

4.2.1 Etapa 1: Inicialização da População

O Algoritmo 7 - $criar\_grafo(n)$, utiliza o modelo de Erdős-Rényi para gerar grafos aleatórios, obtém a matriz de adjacência $A(G)$ desse grafo e extrai os elementos acima da diagonal principal para um vetor binário e_G, pois é uma matriz simétrica, que representam todas as arestas do grafo de modo que será 1 quando houver uma aresta entre os vértices v_i e v_j e 0 quando não houver.

Algoritmo 7: Função `criar grafo`

```
/* Função criar_grafo(n)                                    */
```

$G \leftarrow$ `erdos_renyi`$(n, 0.5)$;

$A(G) \leftarrow G$;

$e_G \leftarrow a_{ij}$, para $i < j$;

return e_G;

O Algoritmo 8 - $indice\_vetor(i, j, n)$, é usado para converter os índices i e j da matriz $A(G)$ em um índice único a_{ij} no vetor binario e_G. Inicialmente é preciso garantir que o par de índices esteja sempre na ordem $i > j$, de modo a evitar duplicação dos valores, em seguida é calculado a posição a_{ij} do par de índices $e_G(i, j)$ usando a fórmula $\frac{i \cdot (2 \cdot n - i - 1)}{2} + (j - i - 1)$, em que o primeiro termo da equação calcula o número total de elementos acima da diagonal principal para todas as linhas anteriores à linha i, e a posição de início da linha i na parte superior da matriz. Enquanto o segundo termo calcula a posição do par (i, j) dentro da linha.

Algoritmo 8: Encontrar o índice no vetor

```
/* Função indice_vetor(i, j, n)                                    */
if i > j then
    i, j ← j, i;
end
return (i·(2·n−i−1))/2 + (j − i − 1);
```

O Algoritmo 9 - $gerar\_populacao(tam\_populacao, k, s, n)$, recebe como entrada o tamanho da população inicial, o número de grafos que serão gerados e os parâmetros k e s que serão usados para verificar as propriedades de Ramsey. Para cada grafo, um valor de *fitness* é calculado, avaliando a qualidade do grafo em termos de suas propriedades de Ramsey e retorna essa população inicial para ser usada nas outras fases do algoritmo.

Algoritmo 9: Gerar a população inicial

```
/* gerar_populacao(tam_populacao, k, s, n)                         */
populacao ← [ ] ;          // Inicializa o vetor da população inicial
foreach individuo ∈ tam_populacao do
    individuo ← criar_grafo(n);
    fitness ← fitness(individuo, k, s, n);
    populacao.append([individuo, fitness]);
end
return populacao;
```

4.2.2 Etapa 2: Avaliação da Função Fitness

A *função fitness* tem como objetivo calcular um valor de aptidão para cada indivíduo da população, baseado na quantidade de cliques de tamanho k e conjuntos independentes de tamanho s que existem no grafo representado pelo vetor binário e_G. Para encontrar os cliques de cada grafo é gerado todas as combinações possíveis de k vértices a partir da lista de vértices $C(V(G),k)$ fornecida.

A variável *clique* é inicializada e assume que a combinação de vértices é um clique até que se prove o contrário. Em seguida dois *loops* aninhados são usados para iterar sobre cada vértice i na combinação de k vértices e sobre todos os vértices que vêm depois do vértice i de $C(V(G),k)$. Isso garante que cada vértice é considerado exatamente uma vez.

Se $e_G[index] \neq 1$, então *C(V(G), k)* não é um clique, a variável *clique* é marcada

como falsa e os laços são encerrados. Entretanto, se $e_G[index] = 1$ então *C(V(G), k)* é um clique de tamanho *k*, e o contador *count* é incrementado. A verificação da quantidade de conjuntos independentes segue a mesma lógica. A função retorna a soma da quantidade de cliques e conjuntos independentes presentes no grafo avaliado, quanto maior o valor da *fitness*, menos apto será o grafo, uma vez que para ser um contraexemplo a *fitness* deve ser 0, ou seja, não conter nenhuma dessas estruturas.

Outros modelos poderiam ser utilizados para avaliar a aptidão dos grafos, como modelos que estimam a probabilidade de um vértice pertencer a um clique ou um conjunto independente, contudo, tais abordagens exigem cálculos mais complexos. O modelo utilizado é satisfatório, pois reflete diretamente a propriedade de Ramsey, penalizando os grafos que possuem estuturas proibidas, sendo simples e objetivo além de permitir uma minimização direta pelos algoritmos heurísticos, tornando-se uma escolha eficiente para a busca de contraexemplos.

Algoritmo 10: Calcular o fitness do grafo

```
/* Função fitness(vetor_binario, k, s, n)                    */
count_clique ← 0;
count_conj_independente ← 0;
foreach C ∈ C(V(G),k) do
    clique ← True;
    foreach i ∈ range(k) do
        foreach j ∈ range(i+1,k) do
            index ← indice_vetor(v[i],v[j],n);
            if e_G[index] ≠ 1 then
                clique ← False;
            end
        end
    end
    if clique then
        count_clique ← count_clique + 1;
    end
end
foreach C ∈ C(V(G),s) do
    conj_independente ← True;
    foreach i ∈ range(s) do
        foreach j ∈ range(i+1,s) do
            index ← indice_vetor(v[i],v[j],n);
            if e_G[index] ≠ 0 then
                conj_independente ← False;
            end
        end
    end
    if conj_independente then
        count_conj_independente ← count_conj_independente + 1;
    end
end
fitness ← count_clique + count_conj_independente;
return fitness;
```

4.2.3 Etapa 3: Seleção dos Indivíduos

Os indivíduos são selecionados através do Algoritmo 11 - $roleta(populacao)$, que recebe como parâmetro a população de indivíduos e verifica o $fitness$ de cada indivíduo, pois caso seja 0 então o contraexemplo foi encontrado. Indivíduos com $fitness$ menores são considerados mais desejáveis, então para priorizar esses indivíduos durante o processo de seleção, será utilizado o inverso do valor de $fitness$, para que recebam um peso maior na seleção, aumentando suas chances de serem escolhidos para a próxima geração. Foi utilizado a técnica da Roleta para a seleção dos cromossomos que foram cruzados.

Após calcular o inverso do valor de $fitness$ de cada indivíduo, é realizado a soma total dos inversos com o intuito de normalizar as aptidões dos indivíduos e então cada valor de inverso do $fitness$ é dividido pela soma total dos inversos, garantindo que as probabilidades de seleção sejam proporcionais aos $fitness$. Usando as probabilidades calculadas, a função seleciona aleatoriamente indivíduos da população, com uma probabilidade proporcional ao inverso de seus $fitness$, simulando uma "roleta", de forma que indivíduos com menor valor de $fitness$ (e maior inverso) terão maior chance de serem escolhidos. A função retorna a lista de indivíduos selecionados que serão utilizados para a geração seguinte no algoritmo.

Algoritmo 11: Selecionar os indivíduos

```
/* Função roleta(populacao)                                          */
foreach individuo ∈ populacao do
    if fitness(individuo) = 0 then
        return individuo;
    end
end
inverso_fitness ← [ ] ;                          // Inicializa vetor vazio
foreach individuo ∈ populacao do
    inverso_fitness. ← (1 / fitness(individuo));
end
soma_total ← soma(inverso_fitness);
probabilidade ← [ ] ;                            // Inicializa vetor vazio
foreach fitness ∈ inverso_fitness do
    probabilidade. ← (fitness / soma_total);
end
selecionados ← random.choices(populacao, probabilidade);
return selecionados;
```

4.2.4 Etapa 4: Criação de Uma Nova População

Ao longo das gerações, a população de grafos evolui sofrendo cuzamentos, refinando as configurações até que um grafo que satisfaça as condições de um contraexemplo seja encontrado. O algoritmo também pode incluir mutações, onde uma porcentagem da população recebe alterações aleatórias para aumentar a diversidade e evitar a convergência prematura para soluções subótimas. Para a mutação, o Algoritmo 12 - $mutacao(cromossomo, taxa\_probabilidade)$, recebe como parâmetro o vetor que será mutado e a taxa de probabilidade que decide qual bit será modificado baseada em um valor aleatório.

Algoritmo 12: Mutação da população

```
/* Função mutacao(cromossomo, taxa_probabilidade)                    */
```

$$probabilidade\_mutacao \leftarrow \begin{cases} 0.1, & \text{se } random() < taxa\_probabilidade \\ 0, & \text{caso contrário} \end{cases};$$

foreach $bit \in e_G$ **do**
  if $random() < taxa\_probabilidade$ **then**
    $e_G[bit] \leftarrow 1 - e_G[bit]$;
  end
end
return e_G;

4.2.4.1 *Crossover: combinação de grafos*

O Algoritmo 13 - $crossover(pai_1, pai_2)$, realiza a operação de cruzamento (ou recombinação) de dois vetores (grafos) usando o método de *crossover* de um ponto, em que cada filho é criado combinando a primeira metade de um pai com a segunda metade da mãe, e vice-versa. É fundamental garantir que ambos os vetores são do mesmo comprimento, pois o ponto de corte precisa ser aplicável em ambos os pais. Nesse algoritmo, o ponto é definido como o meio do vetor, mas poderia ser qualquer outro ponto dentro dos limites do vetor. A função retorna os dois novos filhos gerados.

Algoritmo 13: *Crossover* de um ponto

```
/* Função crossover(pai_1, pai_2)                                    */
```

$ponto \leftarrow \frac{\texttt{comprimento}(pai_1)}{2}$;
$filho_1 \leftarrow parteInicial(pai_1, ponto) + parteFinal(pai_2, ponto)$;
$filho_2 \leftarrow parteInicial(pai_2, ponto) + parteFinal(pai_1, ponto)$;
return $filho_1, filho_2$;

4.2.4.2 *Crossover: combinação de células*

Para encontrar contraexemplos para os números de Ramsey, o algoritmo segue uma abordagem baseada na divisão dos grafos em células e na combinação de alelos, que representam as ligações entre essas células. Inicialmente, o algoritmo gera uma biblioteca contendo todas as

possíveis formas de conexão entre os vértices dentro de uma célula, bem como as configurações possíveis para a "caspa", que são os vértices que não se encaixam perfeitamente em uma célula de tamanho fixo. Essa biblioteca é usada para identificar e classificar células nos grafos gerados ao longo da execução do algoritmo.

Algoritmo 14: Gerar biblioteca de células

$biblioteca \leftarrow \emptyset$;
$num\_celulas \leftarrow \lfloor n/r \rfloor$;
$num\_vertices\_caspa \leftarrow n \mod r$;
$indice \leftarrow 0$;
if $biblioteca \neq \emptyset$ **then**
    return $biblioteca$;
end
foreach $e_{(G)}$ *na célula de tamanho r* **do**
    $biblioteca[indice] \leftarrow$ {'tipo': 'célula', 'arestas': configuração};
    $indice \leftarrow +1$;
end
if $num\_vertices\_caspa > 1$ **then**
    foreach $e_{(G)}$ *na célula de tamanho r* **do**
        $biblioteca[indice] \leftarrow$ {'tipo': 'caspa', 'arestas': configuração};
        $indice \leftarrow +1$;
    end
end
if $num\_vertices\_caspa = 1$ **then**
    $biblioteca[indice] \leftarrow$ {'tipo': 'célula', 'arestas': []};
end
return $biblioteca$

Em seguida, o algoritmo gera uma matriz de alelos, que armazena todas as possíveis conexões entre pares de células. Cada combinação de células possui um conjunto de alelos associados, que são avaliados por meio de uma função de *fitness*. Essa função mede a propensão do grafo gerado em conter cliques ou conjuntos independentes.

Algoritmo 15: Gerar biblioteca de alelos

$num\_celulas \leftarrow |biblioteca|$;
$matriz\_alelos \leftarrow num\_celulas \times num\_celulas$;
$index \leftarrow 0$;
$fitness\_list \leftarrow []$;
$alelo\_indices \leftarrow []$;
if $alelo\_dict \neq \emptyset$ **then**
    return $alelo\_dict$;
end
foreach $celula\_i \in biblioteca$ **do**
    foreach $celula\_j \in biblioteca$ **do**
        $alelo \leftarrow \binom{celula\_i}{celula_j}$;
        foreach $a \in alelo$ **do**
            $fitness\_list \leftarrow fitness(a)$;
            $alelo\_indices \leftarrow (alelo\_index)$;
            $index \leftarrow +1$;
        end
        $alelos\_dict \leftarrow \{$'alelos' : $[alelo]$, 'fitness' : $[fitness\_list]\}$;
    end
end
return $alelos\_dict$;

No *crossover*, cada célula do novo grafo é escolhida aleatoriamente entre um dos pais de acordo com uma probabilidade p, e os alelos são selecionados de acordo com seu *fitness*. Isso garante que alelos mais favoráveis à busca por contraexemplos tenham maior chance de serem transmitidos para as próximas gerações.

Algoritmo 16: Combinar grafos

```
celulas_escolhidas ← [];
for i até [celulas_pai] − 1 do
    for i + 1 até [celulas_mae] do
        if Random() < p then
            celulas_escolhidas ← (celulas_pai[i]);
        end
        else
            celulas_escolhidas ← (celulas_mae[i]);
        end
    end
end
alelo_escolhido ← [];
indice ← 0;
for i até [celulas_escolhidas] − 1 do
    for i + 1 até [celulas_escolhidas] − 1 do
        if fit_alelo_pai < fit_alelo_mae then
            alelo_escolhido ← (alelo_pai);
        end
        else
            alelo_escolhido ← (alelo_mae);
        end
        indice ← +1;
    end
end
novo_grafo ← criar_grafo_completo(celulas_escolhidas, alelo_escolhido, n);
return novo_grafo;
```

4.2.5 Etapa 5: Critério de Parada

A função evoluir é a etapa em algoritmos genéticos, em que a população de soluções é atualizada por meio da seleção, *crossover* e mutação. Inicialmente o Algoritmo 17 - *evoluir*(), cria uma lista vazia para armazenar os novos indivíduos gerados após a evolução e seleciona os

pais par a par, para reprodução usando o método de roleta, que escolhe indivíduos com base em suas probabilidades ponderadas.

Se a roleta selecionar um único indivíduo com *fitness* igual a zero, a função retorna imediatamente esse indivíduo, pois é considerado ideal. É realizado a operação de *crossover* para os pais selecionados e em seguida uma taxa de mutação determina quais filhos serão modificados. Por fim, os novos indivíduos da população são adicionados à lista *filhos* junto com suas respectivas avaliações de *fitness*.

Algoritmo 17: Evoluir a população

```
/* evoluir(populacao,taxa_mutacao,taxa_probabilidade, k, s, n)    */
filhos ← [ ] ;                    // Inicializa o vetor vazio para filhos
pais_selecionados ← roleta(populacao, 2);
if tamanho(pais_selecionados) = 1 e fitness(pais_selecionados[0]) = 0 then
    return pais_selecionados[0];
end
foreach pai ∈ pais_selecionados do
    filho_1, filho_2 ← crossover(pai_1, pai_2);
    if taxa_mut then
        filho_1 ← mutacao(filho_1, taxa_prob);
    end
    if taxa_mut then
        filho_2 ← mutacao(filho_2, taxa_prob);
    end
    filhos ← ([filho_1, fitness(filho_1, k, s, n)]);
    filhos ← ([filho_2, fitness(filho_2, k, s, n)]);
end
return filhos;
```

O Algoritmo 18 - *encontrar_solucao*(), implementa o ciclo do algoritmo genético para encontrar uma solução ideal dentro de um espaço de soluções. É criado a população inicial de indivíduos aleatórios usando o Algoritomo 9 - $gerar\_populacao(tam\_populacao, k, s, n)$, e é inicializado o contador de gerações do algoritmo. Para cada nova população é verificado se existe um indivíduo cuja $fitness$ é 0 e caso exista ele é retornado como solução, caso não, o contador de gerações é incrementado e o loop de evolução continua o processo de evolução da

população enquanto o limite de gerações não for atingido. Se o algoritmo não encontrar nenhum contraexemplo, ele retornará o indivíduo melhor se aproximar de um contraexemplo.

Algoritmo 18: Encontrar a solução

```
/* Função encontrar_solucao(tamanho_populacao, k, s, n, r, p,
   max_geracoes, tempo_limite)                                  */
geracao ← 0;
populacao ← gerar_populacao(tamanho_populacao, k, s, n);
while geracao < max_geracoes do
    populacao ← evoluir(populacao, taxa_mutacao, taxa_probabilidade, k, s, n);
    melhor_individuo ← min(populacao);
    if fitness(melhor_individuo) = 0 then
        return melhor_individuo;
    end
    geracao ← geracao + 1;
end
melhor_individuo ← min(populacao);
return melhor_individuo;
```

5 RESULTADOS

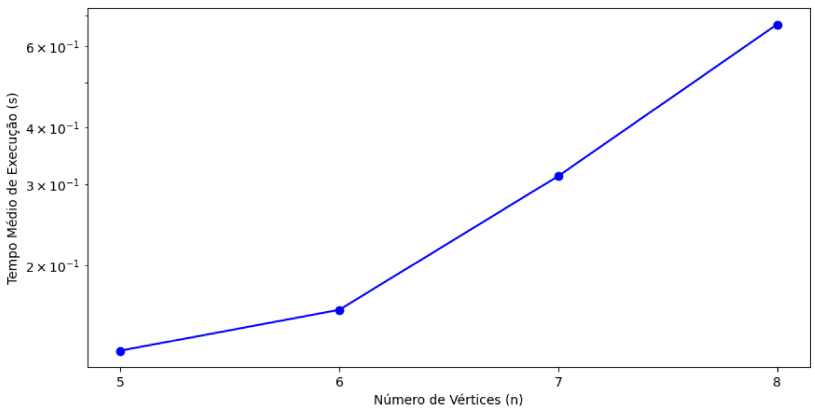
Nesta seção, apresentamos os resultados obtidos na busca por contraexemplos para os números de Ramsey. Os experimentos foram realizados em um notebook Samsung Book X30, equipado com um processador Intel Core i5-10210U (4 núcleos, 8 threads, 1,6 GHz – 4,2 GHz), 8 GB de memória RAM DDR4 e sistema operacional Windows 11. Os algoritmos foram implementados e executados utilizando Python 3.12.3.

Foram realizados testes para os casos $R(3,4)$, $R(3,5)$ e $R(4,4)$, variando o número de vértices no grafo. Os principais aspectos analisados para o algoritmo aleatório foram o tempo de execução e o número de tentativas necessárias para encontrar um contraexemplo. Na execução do algoritmo genético, foram realizados testes para o caso $R(4,4)$ e os resultados obtidos foram analisados em três aspectos principais: o número médio de gerações, o *fitness* da solução e o tempo médio de execução.

5.1 Resultados do Algoritmo Aleatório

O algoritmo aleatório foi capaz de encontrar os resultados esperados para $R(3,4)$, confirmando que, para esse caso, todos os testes realizados levaram à solução correta. Cada configuração foi testada 10 mil vezes, levando aproximadamente 4 (quatro) horas para finalizar os testes e os valores apresentados correspondem à média dos tempos de execução e números de tentativa ao longo dessas execuções. Na Figura 8, observa-se que para grafos menores, o tempo médio de execução do algoritmo permanece próximo de zero, indicando que a busca por contraexemplos é rápida e eficiente nessa faixa de valores.

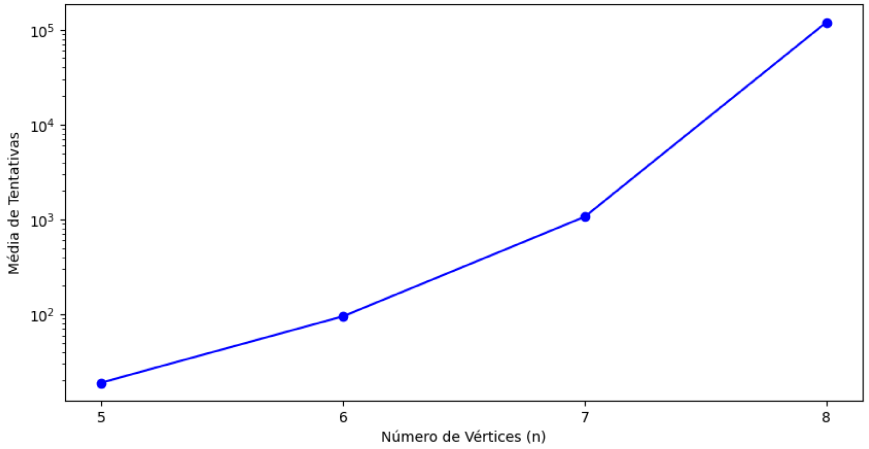
Figura 8 – Tempo médio de execução x Número de vértices R(3,4)



Fonte: Próprio autor

A Figura 9 apresenta a evolução do número médio de tentativas necessárias para encontrar um contraexemplo para o caso $R(3,4)$. Conforme o número de vértices cresce, há um padrão de crescimento exponencial, evidenciado pelo eixo logarítmico da métrica. Esse comportamento reforça a ideia de que a dificuldade do problema aumenta significativamente com o tamanho do grafo, tornando o método aleatório progressivamente menos eficiente.

Figura 9 – Média de tentativa x Número de vértices R(3,4)



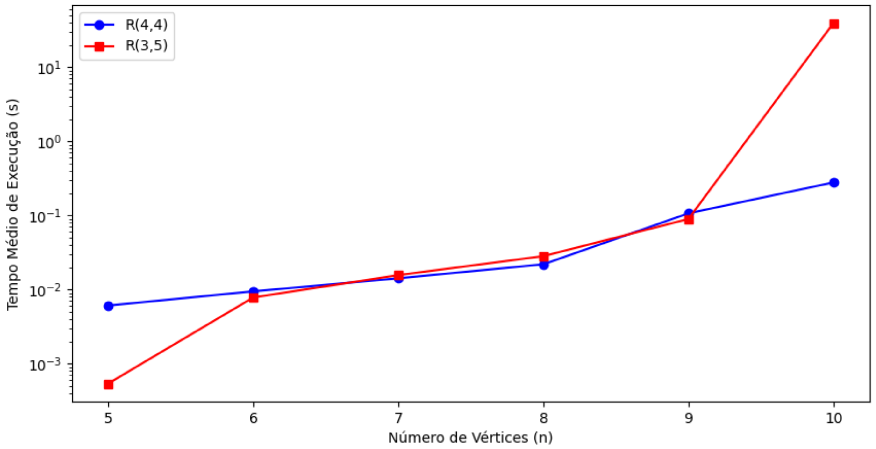
Fonte: Próprio autor

Enquanto para valores pequenos de n há mais liberdade na distribuição das arestas e mais grafos que podem ser contraexemplos, conforme o número de vértices cresce, as possibilidades de encontrar grafos válidos diminuem. Embora a resposta exata para o número de Ramsey $R(3,5)$ seja menor que para $R(4,4)$, vide Tabela 1, isso não significa que a busca pelo algoritmo será mais fácil ou rápida, pois o espaço de grafos possíveis para $R(3,5)$ é mais restrito e difícil de explorar.

Para $R(4,4)$, os testes foram realizados 10 mil vezes para cada valor de n. Já para $R(3,5)$, os testes também foram realizados 10 mil vezes para $n < 10$, mas para $n = 10$, devido ao aumento significativo no tempo de execução, foi possível realizar apenas 1.000 execuções, levando no total aproximadamente 1(um) dia e meio para finalizar os testes. Os valores apresentados na Figura 10 correspondem à média dos tempos de execução.

Além disso, a Figura 11 compara a média de tentativas necessárias para encontrar o contraexemplos para esses dois casos. Observa-se que, para valores menores de n, ambos os casos apresentam um comportamento similar, com um crescimento moderado no número de tentativas. No entanto, a partir de $n = 9$, a complexidade de $R(3,5)$ torna-se evidente, com um

Figura 10 – Comparação entre o tempo médio de execução de R(3,5) e R(4,4)

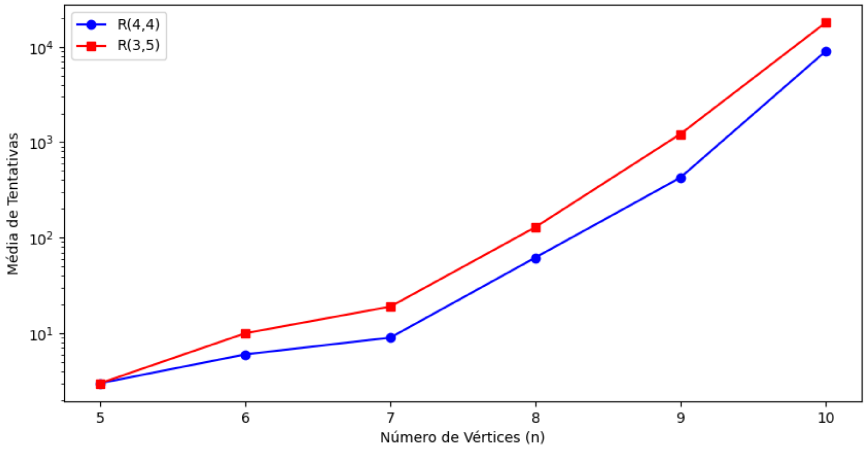


Fonte: Próprio autor

aumento expressivo no número de tentativas em relação a $R(4,4)$.

Esse comportamento evidencia que encontrar contraexemplos para $R(3,5)$ é consideravelmente mais difícil, refletindo a maior restrição imposta pela propriedade de Ramsey, para $n = 11$ o número médio de tentativas passa de 5×10^6 enquanto para o caso $R(4,4)$ permanece abaixo de 10^4.

Figura 11 – Comparação entre a média de tentativas de R(3,5) e R(4,4)



Fonte: Próprio autor

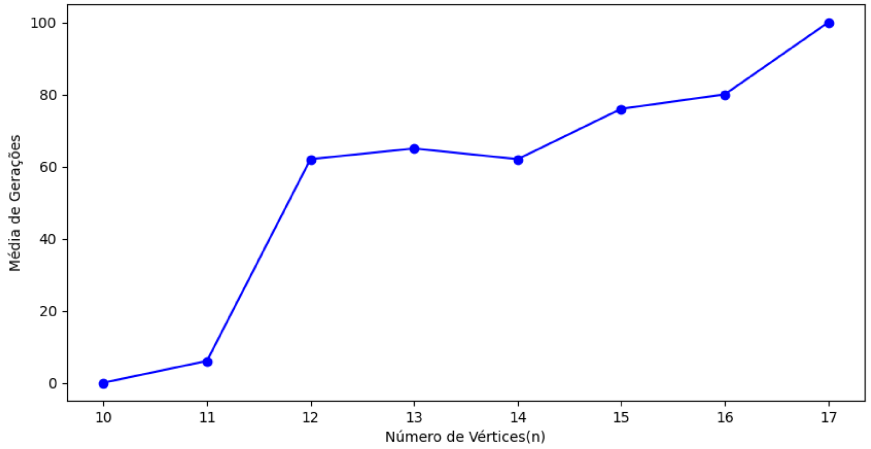
Os resultados indicam que, para instâncias menores do problema, o algoritmo aleatório é eficiente, encontrando soluções rapidamente e com poucas tentativas. No entanto, à medida que o número de vértices cresce, o tempo de execução e o número de tentativas aumentam rapidamente, evidenciando a limitação desse método para problemas maiores. Isso

reforça a necessidade de heurísticas mais sofisticadas, como os algoritmos genéticos, para abordar instâncias mais complexas do problema de Ramsey.

5.2 Resultados do Algoritmo Genético I

A Figura 12 ilustra a relação entre o número médio de gerações e o número de vértices do grafo. Para n variando de 5 a 11, o algoritmo foi capaz de encontrar o contraexemplo, convergindo rapidamente nas primeiras gerações, enquanto que para $n \geq 12$ apresentou um crescimento gradual no número de gerações, porém sem conseguir chegar a uma solução ótima.

Figura 12 – Média de gerações x Número de vértices R(4,4)

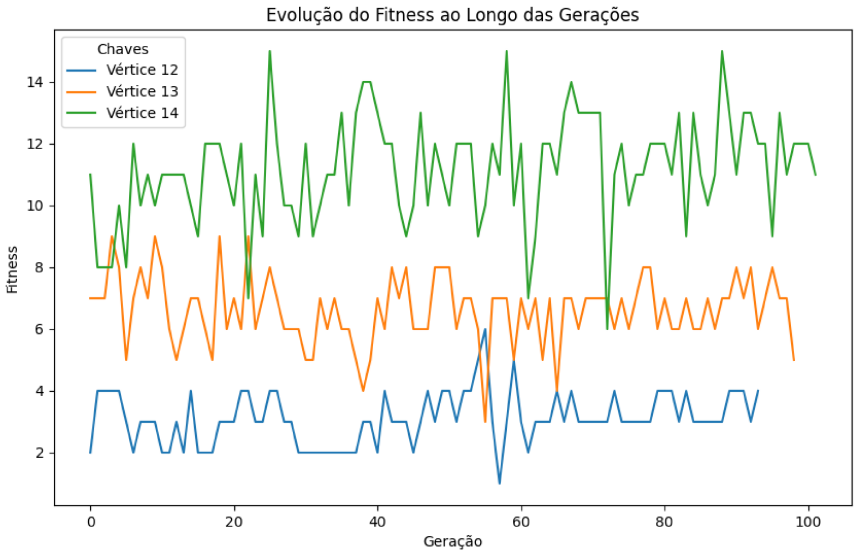


Fonte: Próprio autor

A Figura 13 mostra a média móvel do *fitness*, para grafos com 12, 13, e 14 vértices, com uma variação significativa ao longo das gerações apresentando altos e baixos evidentes. Este comportamento indica que, durante a execução do algoritmo, o *fitness* pode temporariamente melhorar ou piorar à medida que as populações evoluem. Esse comportamento é atribuído ao fato de que, o *crossover* é realizado sem a preocupação de preservar as boas características dos grafos durante a recombinação das soluções.

O processo de *crossover* de um ponto não garante a manutenção das propriedades desejáveis dos indivíduos, o que faz com que o algoritmo explore o espaço de soluções aleatoriamente de maneira ineficiente. Como resultado, o algoritmo acaba se comportando de forma semelhante a um processo aleatório, em que o *fitness* oscila sem uma convergência clara para a solução ótima. A variação no *fitness*, portanto, reflete essa falta de direção no processo evolutivo, tornando o algoritmo mais imprevisível à medida que o número de vértices aumenta.

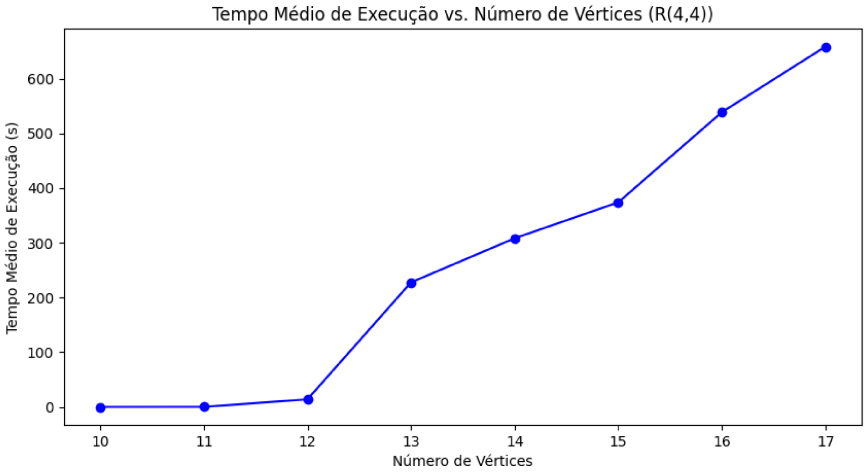
Figura 13 – Evolução do fitness x Gerações



Fonte: Próprio autor

A Figura 14 mostra um aumento progressivo no tempo de execução à medida que o número de vértices n cresce. A partir de $n = 12$, observa-se um aumento mais acentuado no tempo necessário para que o algoritmo encontre uma solução satisfatória, já que grafos maiores demandam mais iterações e avaliações para verificar a propriedade de Ramsey. Para cada valor de n, foram realizadas 100 execuções independentes, e os tempos apresentados correspondem à média desses testes. No total, a execução completa dos experimentos levou aproximadamente 2,3 dias, evidenciando o custo computacional crescente à medida que o problema se torna mais complexo.

Figura 14 – Tempo médio de execução x Número de vértices

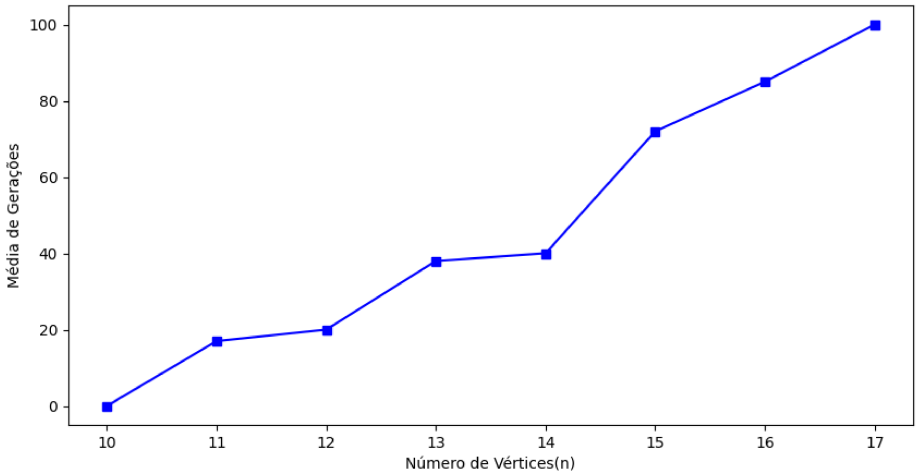


Fonte: Próprio autor

5.3 Resultados do Algoritmo Genético II

Além do algoritmo genético com *crossover* de um ponto, também foram realizados testes com um algoritmo genético em que o grafo é dividido em células e alelos. Esse algoritmo mostrou um comportamento distinto, particularmente nas métricas de número de gerações, *fitness* e tempo de execução. É possível observar na Figura 15 que o número de gerações permanece quase constante, o que indica uma convergência mais rápida e mais estável do algoritmo. Apenas para $n = 17$ o algoritmo exigiu mais iterações, chegando a 100 gerações, mas ainda assim mantendo uma taxa de crescimento controlada.

Figura 15 – Média de gerações x Número de vértices R(4,4)



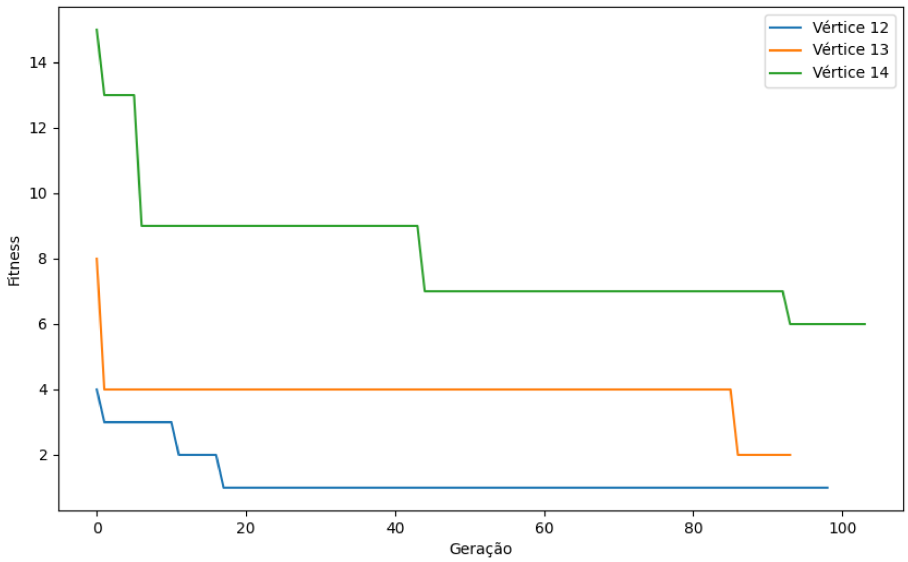
Fonte: Próprio autor

A média móvel do *fitness* para o algoritmo genético com células e alelos apresentada na Figura 16 mostra uma convergência rápida nas primeiras gerações, estabilizando-se em um valor constante após poucas iterações. Essa característica demonstra que o algoritmo está conseguindo otimizar eficientemente as soluções sem oscilações significativas.

Embora o algoritmo não tenha conseguido alcançar a solução ótima, as soluções encontradas são significativamente melhores do que as obtidas pelo Algoritmo Genético I. Isso sugere que a incorporação da estrutura baseada em células e alelos foi eficaz na melhoria da qualidade dos indivíduos gerados, permitindo uma busca mais direcionada e eficiente dentro do espaço de soluções viáveis.

O tempo de execução do algoritmo genético com células e alelos foi significativamente mais eficiente, conforme observado na Figura 17, além de apresentar pouca variação entre

Figura 16 – Evolução do fitness ao longo das gerações

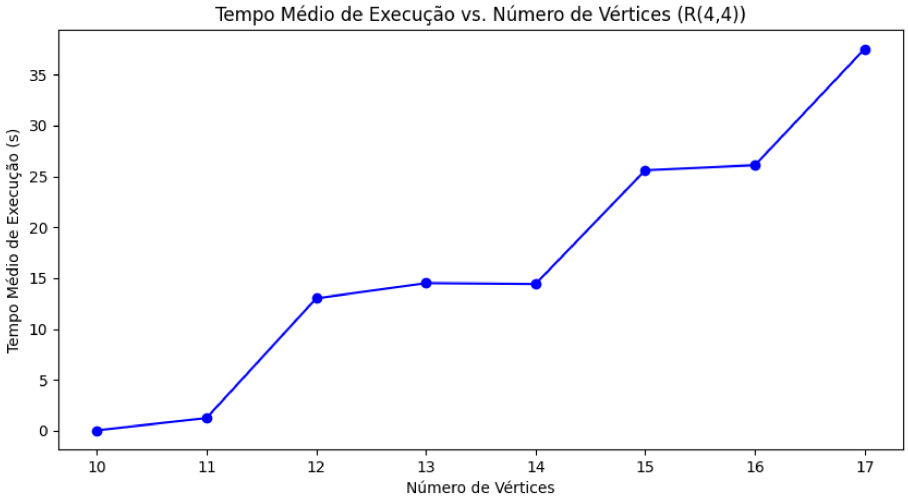


Fonte: Próprio autor

os vértices $n = 12$ a $n = 14$, sugerindo que o algoritmo lida de maneira eficiente com grafos de diferentes tamanhos, sem grandes variações no custo computacional à medida que o número de vértices cresce.

Para cada valor de n, foram realizadas 100 execuções para cada geração. No geral, a execução completa dos testes levou aproximadamente 4 horas, evidenciando a eficiência computacional do algoritmo mesmo para valores mais elevados de n.

Figura 17 – Tempo médio de execução x Número de vértices



Fonte: Próprio autor

6 DISCUSSÃO DOS RESULTADOS

Nesta seção, discutimos os resultados obtidos na avaliação das abordagens heurísticas utilizadas para encontrar contraexemplos para os números de Ramsey. Os três algoritmos analisados apresentaram desempenhos distintos, evidenciando vantagens e limitações em relação à eficiência e à escalabilidade da busca.

6.1 Algoritmo Aleatório

O algoritmo aleatório mostrou-se eficiente para instâncias menores do problema, com tempo de execução reduzido e poucas tentativas necessárias para encontrar contraexemplos. No entanto, à medida que o número de vértices aumentou, houve um crescimento exponencial no tempo de execução e no número de tentativas, destacando uma clara limitação dessa abordagem para problemas maiores. Esse comportamento está alinhado com a natureza combinatória do problema de Ramsey, onde a busca aleatória se torna cada vez menos eficaz conforme o espaço de soluções cresce exponencialmente.

Um aspecto interessante a se considerar é que, mesmo sendo um algoritmo puramente baseado em força bruta, ele pode, em certos casos, encontrar soluções rapidamente por pura sorte. Isso levanta um ponto sobre a aleatoriedade: em um espaço de busca grande o suficiente, existem momentos em que o resultado é alcançado sem qualquer inteligência embutida no método. Essa imprevisibilidade é uma vantagem e uma desvantagem ao mesmo tempo. Enquanto a sorte pode acelerar a busca em alguns casos, também pode fazer com que, para instâncias maiores, o tempo para encontrar uma solução se torne inaceitável.

Diferente de métodos heurísticos que podem direcionar a busca para regiões mais promissoras do espaço de soluções, o algoritmo aleatório não apresenta um mecanismo de convergência eficiente. Isso o torna altamente dependente do tamanho da amostra testada, resultando em um custo computacional elevado, conforme discutido por (THOMAS *et al.*, 2002).

6.2 Algoritmo Genético I

O primeiro algoritmo genético (que utiliza no *crossover* a técnica de um ponto) apresentou um desempenho intermediário. A média do número de gerações indicou que, para grafos menores, a convergência ocorre rapidamente. No entanto, para valores de $n \geq 12$, observou-se um aumento na quantidade de gerações necessárias para encontrar um contraexemplo, sugerindo

uma escalabilidade limitada.

O *crossover* de um ponto é um método clássico, mas seu comportamento revela um problema central dos algoritmos genéticos, a destruição prematura de boas soluções. No processo evolutivo real, os organismos tendem a preservam características vantajosas ao longo do tempo, mas no *crossover* de um ponto não há garantia de que as melhores partes dos pais serão mantidas. Como resultado, o algoritmo frequentemente perde boas soluções, levando a oscilações no *fitness*. Essa degradação genética é ainda mais acentuada pelo fato de que os indivíduos são recombinados sem um critério específico para a conservação das estruturas mais úteis. Como apontado por (GOLDBERG, 1989), a seleção e a recombinação são fatores críticos para a eficiência dos algoritmos genéticos, e a abordagem adotada impactou negativamente na busca.

6.3 Algoritmo Genético II

O algoritmo genético baseado na divisão dos grafos em células e alelos demonstrou um desempenho superior em relação ao *crossover* de um ponto. A estabilidade observada no número de gerações e na evolução do *fitness* indica que essa abordagem permitiu uma otimização mais eficiente das soluções. Ela difere radicalmente da anterior por permitir que a informação estruturada seja preservada durante a evolução. Ao dividir o grafo em células menores e considerar alelos na recombinação, permitiu a preservação de padrões específicos do grafo original, reduzindo a degradação do *fitness* ao longo das gerações e facilitando a convergência.

Além disso, essa estruturação permite que o algoritmo explore padrões dentro do grafo que seriam ignorados por abordagens mais tradicionais, como por exemplo as conexões entre vértices que favorecem a formação de um contraexemplo e que são mantidos durante a evolução. Como apontado por (PACHECO, 1999), o uso de operadores genéticos bem projetados pode melhorar significativamente a eficiência dos algoritmos evolutivos, o que se reflete nos resultados obtidos com essa abordagem.

Ademais, essa abordagem distribui melhor o processamento, com o tempo de execução mantendo-se relativamente estável, permitindo que seja aplicada a grafos maiores sem um aumento exponencial na complexidade.

7 CONCLUSÃO

Os resultados evidenciam que heurísticas mais elaboradas são necessárias para lidar com o problema de Ramsey em grafos maiores. A análise comparativa entre os algoritmos mostrou que abordagens simples, como a busca aleatória, são eficientes apenas para casos pequenos, mas rapidamente se tornam inviáveis conforme a complexidade do problema cresce. Isso destaca a necessidade de técnicas mais avançadas para explorar o espaço de busca de maneira inteligente.

O algoritmo genético com *crossover* de um ponto apresentou melhorias em relação ao aleatório, mas sofreu com a falta de uma direção evolutiva bem definida. A escolha dos operadores genéticos desempenha um papel crítico e embora o *crossover* de um ponto seja uma técnica amplamente utilizada, para problemas complexos como o de Ramsey, é essencial adotar métodos que preservem certas características ao longo das gerações.

Por fim, a técnica baseada em células mostrou-se a mais promissora entre as testadas. A capacidade de preservar subestruturas importantes e a estabilidade na convergência indicam que esse método representa um avanço significativo na busca por contraexemplos. Além disso, manteve um tempo de execução mais eficiente para diferentes tamanhos de grafo, o que sugere que pode ser aplicada a instâncias ainda maiores, o que abre possibilidades para futuras pesquisas.

Com base nesses resultados, estudos futuros podem investigar o uso de técnicas híbridas que combinem os pontos fortes dos diferentes métodos testados. A inclusão de mecanismos melhores para a seleção e a mutação, pode melhorar ainda mais o desempenho do algoritmo genético com células e alelos. Além disso, explorar estratégias baseadas em aprendizado de máquina para guiar a busca pode representar um novo salto de eficiência na resolução do problema de Ramsey.

REFERÊNCIAS

BOLLOBÁS, B. **Random Graphs**. London: Springer, 2001. v. 2. ISBN 978-0521809207.

BONDY, J.; MURTY, U. **Graph Theory**. Londres: Springer, 2008. v. 2. ISBN 978-1-84628-969-9.

BOTLER, F.; COLLARES, M.; MARTINS, T.; MENDONçA, W.; MORRIS, R.; MOTA, G. **Combinatória**. Brasil: IMPA, 2021. v. 1. ISBN 978-65-89124-53-5.

DIESTEL, R. **Graph Theory**. London: Springer, 2017. v. 5.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Boston, MA: Addison-Wesley, 1989. ISBN 978-0201157673.

GRAHAM, R. L.; ROTHSCHILD, B. L.; SPENCER, J. H. **Ramsey Theory**. New Jersey: Wiley-Interscience, 1990. v. 2. ISBN 978-0471500469.

MCKAY, B. D.; RADZISZOWSKI, S. P. Subgraph counting identities and ramsey numbers. **Journal of Combinatorial Theory, Series B**, v. 69, n. 2, p. 193–209, 1997. ISSN 0095-8956. Disponível em: https://www.sciencedirect.com/science/article/pii/S0095895696917414.

MELANIE, M. **An Introduction to Genetic Algorithms**. Cambridge: The MIT Press, 1998. v. 1. ISBN 0262133164.

PACHECO, M. A. C. Algoritmos genéticos: Princípios e aplicações. **ICA: Laboratório de Inteligência Computacional Aplicada, Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro**, 1999. Disponível em: https://www.inf.ufsc.br/~mauro.roisenberg/ine5377/Cursos-ICA/CE-intro_apost.pdf.

RADZISZOWSKI, S. P. Ramsey theory applications. **Small Ramsey Numbers**, p. 04–16, jun. 2024. Disponível em: http://doi.org/10.37236/21.

ROSTA, V. Ramsey theory applications. **Ramsey Theory Applications**, p. 04–16, dez. 2004. Disponível em: https://doi.org/10.37236/34.

THOMAS, H. C.; CHARLES, E. L.; RONALD, L. R.; CLIFFORD, S. **Introduction to algorithms**. Massachusetts: The MIT Press, 2002. v. 2. ISBN 85-352-0926-3.

WEST, D. B. **Introduction to Graph Theory**. New Jersey: Prentice Hall, 2001. v. 2. ISBN 81-7808-830-4.