Universidade Federal do ABC
Centro de Matemática, Computação e Cognição
Projeto de Graduação em Computação

Um Estudo de Algoritmos Exatos para o Problema da Coloração de Grafos

Autor:
Raphael Ramos da Silva

Orientadora:
Prof. Dra. Carla Negri Lintzmayer

Abril, 2025

Raphael Ramos da Silva

Um Estudo de Algoritmos Exatos para o Problema da Coloração de Grafos

Trabalho de graduação apresentado ao Centro de Matemática, Computação e Cognição da Universidade Federal do ABC, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Prof. Carla Negri Lintzmayer

Abril, 2025

Sumário

1 Introdução

Grafos são estruturas utilizadas para representar relações par-a-par entre objetos e estão presentes em diversas situações do dia a dia. Seu uso abrange problemas como definir a menor rota em uma viagem e organizar as redes que sustentam a Internet. A teoria dos grafos é uma área amplamente estudada pela comunidade científica e existem vários temas em aberto que atraem a atenção dos pesquisadores [4].

Neste trabalho, focamos em um problema clássico conhecido como Coloração de Grafos, que teve sua primeira aparição em 1852, quando o matemático Francis Guthrie tentava colorir o mapa da Inglaterra de tal modo que dois distritos vizinhos não tivessem a mesma cor. Durante esse exercício, Guthrie conjecturou que 4 cores distintas seriam suficientes para realizar tal coloração [15]. Seu irmão mais novo, Frederick, apresentou essa conjectura ao seu professor Augustus De Morgan, que ficou bastante entusiasmado com o problema. Então, De Morgan redigiu uma carta para o matemático Sir Willian Rowan Hamilton, que não teve interesse em resolvê-lo e o tema ficou cerca de 20 anos sem grandes evoluções na comunidade científica.

O problema foi revisitado por outros estudiosos, mas apenas em 1976, com a ajuda de um IBM 360, Kenneth Appel e Wolfgang Haken apresentaram tal demonstração, data na qual foi formalmente definido o Teorema das Quatro Cores [12]. Com isso, o problema da coloração de grafos passou a ser estudado com maior profundidade não só para grafos que representam mapas e seu conceito passou a ser aplicado em diferentes áreas, como gerenciamento de redes de comunicação, otimização de recursos e planejamento de horários. Problemas de alocação, no geral, podem ser resolvidos com algoritmos de coloração. Um exemplo importante do uso desses algoritmos está na implementação de compiladores, onde o princípio de coloração abstrai o mecanismo de alocação dos registradores [1].

O problema-alvo deste estudo, denominado Coloração Mínima de Vértices, consiste em atribuir cores aos vértices de um grafo utilizando a menor quantidade possível de cores, de modo que vértices adjacentes não recebam a mesma cor. Esse problema é classificado como NP-difícil [13], o que significa que não há algoritmo eficiente que resolva todas as instâncias em tempo polinomial, a menos que P = NP.

O objetivo deste trabalho de conclusão de curso foi analisar e comparar algoritmos heurísticos e exatos para o problema da coloração mínima. O texto está organizado em seis seções. A Seção 2 apresenta a fundamentação teórica relacionada à Teoria dos Grafos e Algoritmos, introduzindo os principais conceitos e resultados. Na Seção 3, são discutidas heurísticas utilizadas para tratar o problema, enquanto a Seção 4 explora algoritmos exatos que garantem soluções ótimas. A Seção 5 exibe os resultados obtidos, e, por fim, a Seção 6 traz as conclusões do estudo.

2 Fundamentação Teórica

Nas Seções 2.1 e 2.2, definimos os conceitos da Teoria de Grafos necessários para entendimento do Problema da Coloração Mínima de Vértices, com base nos materiais de Bondy e Murty [4] e Feofiloff, Kohayakawa e Wakabayashi [11]. Na Seção 2.3, temos uma abordagem conceitual sobre algoritmos, contendo as definições necessárias para analisar e compreender os algoritmos presentes no restante do texto.

2.1 Conceitos Básicos em Grafos

Um **grafo** (simples) G é um par (V, E), onde V é um conjunto de elementos chamados de **vértices** e E é um conjunto que contém pares de elementos de V, denominados **arestas**. Vamos sempre denotar o conjunto de vértices de qualquer grafo X por $V(X)$ e o conjunto de arestas por $E(X)$ ao invés de determinar um par ordenado para cada grafo. A **ordem** de um grafo representa a cardinalidade de seu conjunto de vértices, enquanto o **tamanho** é a cardinalidade do conjunto de arestas. Os grafos podem ser representados por imagens e na Figura 1 temos um exemplo de um grafo que representa o mapa do Brasil, onde os vértices são representados por círculos e as aresta por linhas que conectam os círculos.

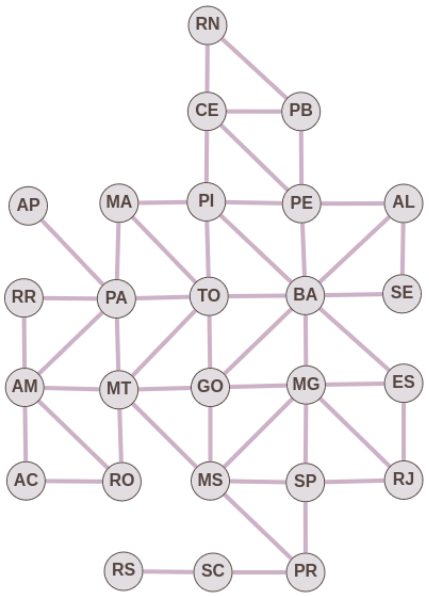


Figura 1: Grafo com vértices que simbolizam os estados brasileiros e arestas que representam as relações de divisa entre os estados.

Dado um grafo G, dois vértices $u, v \in V(G)$ são **vizinhos** se existe uma aresta $e = \{u, v\} \in E(G)$. Por simplicidade, escreveremos apenas uv ao invés de $\{u, v\}$. Neste caso, dizemos que a aresta e é **incidente** em u e v, e que u e v são vértices **adjacentes**. A **vizinhança** de um vértice v, denotada por $N_G(v)$, é o conjunto de

todos os vértices vizinhos de v. No grafo da Figura 1, por exemplo, a vizinhança do vértice SP é o conjunto de vértices que fazem divisa com São Paulo, isto é, $N_G(SP) = \{RJ, MS, MG, PR\}$.

O **grau** de um vértice v, denotado por $d_G(v)$, indica a quantidade de arestas incidentes a v, ou seja, $d_G(v) = |N_G(v)|$. O menor grau de um vértice em G determina o grau mínimo de G, e é denotado por $\delta(G)$. O maior grau de um vértice de G indica o grau máximo de G, denotado por $\Delta(G)$. No grafo da Figura 1, temos $\delta(G) = 1$, por conta do vértice RS e $\Delta(G) = 8$, pelo vértice BA.

Um grafo H é um **subgrafo** de G se os vértices e arestas de H são subconjuntos dos vértices e arestas de G, ou seja, $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Se H contém todas as arestas de G que existem entre esses vértices em $V(H)$, então H é chamado de **subgrafo induzido** de G. Esse subgrafo induzido é denotado por $G[V(H)]$ e preserva todas as conexões presentes no grafo original entre os vértices do subconjunto considerado.

O resultado a seguir é um dos mais clássicos da teoria dos grafos, e estabelece uma relação entre a quantidade de vértices e arestas de um grafo.

Lema 2.1 (Aperto de Mãos)**.** *Em um grafo G, vale que $\sum_{v \in V(G)} d_G(v) = 2\,|E(G)|$.*

Demonstração. Em G, observe que cada aresta $e \in E(G)$ conecta exatamente dois vértices distintos, ou seja, cada aresta contribui com 1 para o grau de cada um de seus dois vértices incidentes. Assim, cada aresta e é contada duas vezes quando somamos os graus de todos os vértices. Consideremos agora a contagem dos pares (v, e), onde $v \in V(G)$ é um vértice e $e \in E(G)$ é uma aresta que incide sobre v. Existem duas maneiras de contar esses pares:

(a) Por arestas: Como cada aresta $e \in E(G)$ conecta dois vértices, ela aparece exatamente em dois pares (v, e). Logo, o número total de pares (v, e) é exatamente $2\,|E(G)|$.

(b) Por vértices: Cada vértice $v \in V(G)$ está contido em exatamente $d_G(v)$ pares (v, e). Assim, a soma total dos pares (v, e) é a soma dos graus de todos os vértices, ou seja, $\sum_{v \in V(G)} d_G(v)$.

Como estamos contando o mesmo conjunto de pares (v, e) de duas maneiras diferentes, vale que $\sum_{v \in V(G)} d_G(v) = 2\,|E(G)|$. ■

Um grafo é dito **completo** se todos os seus vértices são adjacentes entre si. Os grafos completos são representados por K_n, onde n indica a quantidade de vértices do grafo. O **complemento** de um grafo G, denotado por $\overline{G}$, é o grafo que possui o mesmo conjunto $V(G)$, de modo que seus vértices são adjacentes apenas se não

forem vizinhos em G. Logo, a união de $E(G)$ e $E(\overline{G})$ resulta no conjunto das arestas de um grafo completo com $|V(G)|$ vértices.

Uma **partição** de um conjunto V é uma coleção de subconjuntos não vazios $V_1, V_2, \ldots, V_k$ que são mutuamente disjuntos e cuja união é V, ou seja, $V_i \cap V_j = \emptyset$ para todo $i \neq j$ e $\bigcup_{i=1}^{k} V_i = V$. Um grafo G é **bipartido** se o conjunto de vértices $V(G)$ admite uma partição $\{X, Y\}$, de modo que toda aresta $xy \in E(G)$ satisfaça $x \in X$ e $y \in Y$. Um exemplo clássico de grafos bipartidos é o grafo bipartido completo $K_{m,n}$, no qual o conjunto X contém m vértices, o conjunto Y contém n vértices, e cada vértice em X está conectado a todos os vértices de Y.

Em um grafo G, um **passeio** é uma sequência de vértices $(v_0, v_1, \ldots v_n)$ tal que $\{v_i, v_{i+1}\} \in E(G)$, para $0 \leq i < n$. O vértice inicial v_0 é a origem do passeio, o vértice final v_n é o término e os demais são denominados vértices internos ao passeio. Na Figura 1, o percurso entre São Paulo (SP) e Alagoas (AL), descrito por (SP, MG, BA, AL) é um exemplo de passeio.

Se um passeio não repetir arestas, ele é denominado **trilha**, se não houver repetição de vértices, é denominado **caminho**. Um **grafo caminho**, representado por P_n, é um grafo que consiste em um único caminho com n vértices. Um passeio que não repete vértices internos e a origem coincide com o término é denominado **ciclo**. Um **grafo ciclo**, representado por C_n, é um grafo que consiste em um único ciclo com n vértices.

Um grafo G é dito **conexo** se para quaisquer $u, v \in V(G)$ existir um caminho com origem em u e término em v. Caso contrário, o grafo é chamado de **desconexo**.

Um grafo G é dito **planar** se puder ser representado graficamente no plano de modo que não haja cruzamento entre suas arestas. Um resultado importante sobre planaridade foi proposto em 1930, pelo matemático Kazimiers Kurastowski, cujo resultado encontra-se descrito no teorema a seguir, e a demonstração pode ser encontrada no material de Bondy e Murty [4].

No teorema a seguir, usa-se o termo **subdivisão** de um grafo. Essa ação consiste em substituir determinadas arestas por caminhos de um ou mais vértices intermediários, sem alterar a estrutura de conectividade entre os vértices. Essas substituições preservam as relações fundamentais do grafo original, permitindo identificar certas propriedades, como a planaridade.

Teorema 2.2. *Um grafo G é planar se e somente se não contém uma subdivisão de K_5 ou $K_{3,3}$ como subgrafo.*

Toda representação de um grafo planar divide o plano onde ele está representado em regiões chamadas **faces**. O grafo da Figura 1 é planar e a região triangular

delimitada pelas arestas que formam o ciclo (RN, CE, PB, RN) representa uma face. Denotamos por $F(G)$ o **conjunto de faces** do grafo G. O **grau de uma face** f, denotado por $d(f)$, é igual ao número de arestas contidas no ciclo que a delimita. No resultado a seguir, estabelecemos uma relação entre a quantidade de faces e arestas de um grafo.

Lema 2.3. *Seja G um grafo planar com m arestas e faces $f_1 \dots f_{|F(G)|}$. Então,*

$$\sum_{i=1}^{|F(G)|} d(f_i) = 2|E(G)|.$$

Demonstração. Em um grafo planar, cada aresta está associada a duas faces, inclusive as arestas que delimitam a borda do grafo, pois são contadas tanto para a face externa quanto para a face interna adjacente. Se somarmos os graus de todas as faces, estamos contando quantas vezes cada aresta delimita uma face. Como cada aresta pertence a exatamente duas faces, ela será contada duas vezes no somatório dos graus das faces. Logo, cada aresta contribui exatamente 2 vezes ao somatório.

Dessa forma, somar os graus de todas as faces é equivalente a contar cada aresta duas vezes. ■

Além da relação existente entre a quantidade de faces e arestas, também é possível estabelecer uma relação entre a quantidade de vértices, arestas e faces pelo teorema a seguir.

Teorema 2.4 (Fórmula de Euler)**.** *Se G é um grafo conexo e planar, então qualquer representação planar de G possui $|E(G)| - |V(G)| + 2$ faces.*

O resultado descrito no Teorema 2.4 foi obtido pelo estudo de poliedros convexos, na geometria. Sua demonstração pode ser encontrada no trabalho de Bondy e Murty [4].

Teorema 2.5. *Em um grafo planar conexo G com $|V(G)| \geq 3$, sempre vale que $|E(G)| \leq 3\,|V(G)| - 6$.*

Demonstração. Sejam $n = |V(G)|$, $m = |E(G)|$. Como G é um grafo conexo planar com $n \geq 3$, o grau de cada face é no mínimo 3. Assim, partindo do Lema 2.3, temos:

$$2m = \sum_{f \in F(G)} d(f) \geq \sum_{f \in F(G)} 3 = 3|F(G)|.$$

Logo, pelo Teorema 2.4, $m - n + 2 = |F(G)| \leq 2m/3$, de modo que $m \leq 3n - 6$. ■

Partindo do Teorema 2.5, obtemos um limitante para o grau mínimo de qualquer grafo planar, resultado descrito no lema a seguir.

Lema 2.6. *Em um grafo planar há pelo menos um vértice com grau no máximo* 5.

Demonstração. Seja G um grafo planar com $n = |V(G)|$ e $m = |E(G)|$. Sabemos, pelo Lema 2.1, que $\sum_{v \in V(G)} d_G(v) = 2m$. Suponha, para fins de contradição, que $d_G(v) \geq 6$ para todo $v \in V(G)$. Então teríamos $\sum_{v \in V(G)} d_G(v) \geq 6n$ e valeria que $2m \geq 6n$. Mas, em um grafo planar, temos $m \leq 3n - 6$, pelo Teorema 2.5, de onde vale que $2m \leq 6n - 12$. Então teríamos $6n \leq 6n - 12$, o que é impossível. ■

Um **conjunto independente** em um grafo G é um subconjunto de vértices não-adjacentes dois a dois. Um conjunto independente é **maximal** se nenhum vértice puder ser adicionado ao conjunto de modo que ele continue independente. Um conjunto independente S é **máximo** se não existir nenhum outro conjunto independente de G maior que S, e a sua cardinalidade indica o **número de independência** de G, denotado por $\alpha(G)$. Um **clique** é um subconjunto de vértices adjacentes dois a dois e, de modo equivalente, a cardinalidade do clique máximo é denotada por $\omega(G)$. Note que todo clique em G é um conjunto independente em $\overline{G}$, dado que o oposto de um clique é um conjunto independente.

Seja G um grafo e u e v dois vértices quaisquer de $V(G)$. A operação de **contração** consiste em substituir u e v por um único novo vértice w, que herda todas as arestas incidentes nos vértices substituídos, excluindo os loops e arestas paralelas que podem se formar nesse processo. Assim, no novo grafo G', o conjunto de vértices é dado por

$$V(G') = (V(G) \backslash \{u, v\}) \cup \{w\},$$

e o conjunto de arestas é dado por

$$E(G') = \big(E(G) \setminus \{e \in E(G) \mid e \text{ incide em } u \text{ ou } v\}\big) \cup \{wx \mid x \in N_G(u) \cup N_G(v)\}.$$

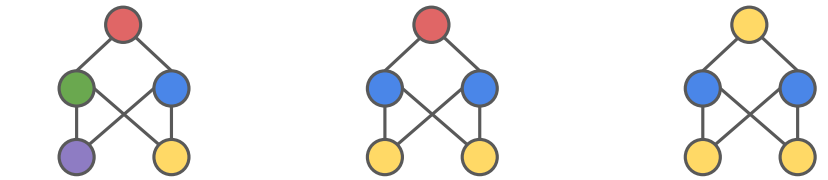
Agora, tome dois vértices a e b não adjacentes em G. A operação de **adição**, consiste em adicionar a aresta ab em $E(G)$, tornando a e b vértices adjacentes, resultando no grafo G' tal que $V(G') = V(G)$ e $E(G') = E(G) \cup \{ab\}$.

2.2 Coloração de Grafos

Uma **coloração** de um grafo G é uma função $c : V(G) \to \{1, 2, \ldots, k\}$, que atribui uma cor $c(v)$ a cada vértice $v \in V(G)$. Em uma **coloração própria**, vértices adjacentes recebem cores distintas, isto é, $\forall uv \in E(G)$, temos $c(u) \neq c(v)$. A

coloração dos vértices de G pode ser vista também como uma partição de $V(G)$ em k conjuntos independentes $V_1, V_2, \ldots, V_k$, onde cada conjunto V_i é associado à cor i.

Uma **coloração ótima** é uma coloração própria que utiliza a menor quantidade possível k de cores. Esse número k é chamado de **número cromático** de G, denotado por $\chi(G)$. Na Figura 2, encontram-se três colorações próprias de um grafo G, sendo que apenas a coloração 3 (item c) é ótima, pois utiliza a menor quantidade possível de cores.



(a) coloração 1 *(b) coloração 2* *(c) coloração 3*

Figura 2: Grafo G com três colorações próprias distintas.

A busca por uma coloração ótima é um desafio relevante que atrai diversos estudiosos. Para grafos com grande quantidade de vértices e arestas, a definição de limitantes para o número cromático pode ajudar a reduzir o conjunto de possíveis soluções. O teorema a seguir traz um resultado importante, com a definição de um limitante superior para o número cromático de grafos planares.

Teorema 2.7 (Cinco Cores)**.** *Em um grafo planar simples G, tem-se $\chi(G) \leq 5$.*

Demonstração. Suponha, por contradição, que existe um grafo planar que requer mais de 5 cores para ser colorido. Seja G um grafo minimal com essa propriedade, ou seja, um grafo planar com o menor número de vértices que não pode ser colorido com 5 cores. Isso significa que qualquer subgrafo próprio de G pode ser colorido com 5 cores.

Pelo Lema 2.6, em todo grafo planar G existe pelo menos um vértice v com $d_G(v) \leq 5$. Remova v de G, obtendo um subgrafo G' com $n-1$ vértices. Como G é o menor grafo que não pode ser colorido com 5 cores, isso implica que G' pode ser colorido com no máximo 5 cores. Agora, vamos tentar colorir G a partir da coloração de G'.

O vértice v tem no máximo 5 vizinhos. Se pelo menos uma das cores não estiver sendo usada pelos vizinhos de v, podemos atribuir essa cor a v, obtendo uma coloração válida para G, o que contradiz a suposição inicial.

Agora, vamos considerar o cenário no qual os 5 vizinhos v_1, v_2, v_3, v_4, v_5 de v possuem cores distintas. Sejam v_1 e v_3 dois vizinhos de v associados à cor 1 e 3, respectivamente. Considere uma expansão de um subgrafo partindo de v_1 apenas nas cores 1 e 3. Teremos duas possibilidades: (i) não chegaremos até v_3 (caso 1 da Figura 3) ou (ii) em algum momento chegaremos até v_3, formando uma curva fechada (caso 2 da Figura 3).

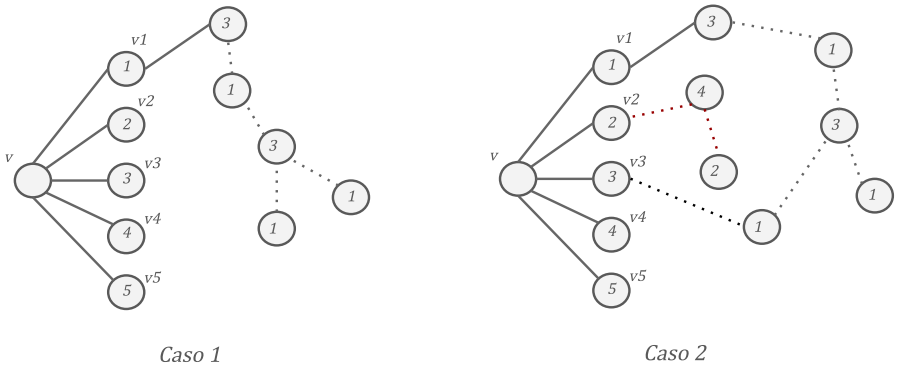


Figura 3: Ideia da demonstração do Teorema 2.7.

No Caso 1, podemos recolorir o grafo trocando as cores 1 e 3 em todo o caminho visitado. Desse modo, v_1 recebe a cor 3 e a cor 1 fica disponível para v, concluindo a coloração de G com 5 cores.

No Caso 2, dada a impossibilidade de troca entre as cores 1 e 3, podemos mudar o vértice inicial da expansão para v_2 até v_4. Pelo Teorema da Curva de Jordan [4], esse caminho atravessaria o ciclo de v_1 a v_3, o que é impossível, pois isso fere a planaridade de G. Então podemos efetuar a troca entre as cores 2 e 4, liberando uma cor para v. Dessa forma, conseguimos colorir G com até 5 cores, o que contradiz a suposição inicial. ■

Com a definição de um limitante superior para o número cromático de grafos planares (Teorema 2.7), houve diversos avanços na comunidade científica acerca do tema. Essa evolução gerou um resultado ainda mais relevante, onde esse mesmo limitante foi reduzido de 5 para 4 cores, conforme descreve o teorema a seguir.

Teorema 2.8 (Quatro Cores)**.** *Em um grafo planar simples G, tem-se $\chi(G) \leq 4$.*

A demonstração do Teorema das Quatro Cores é extremamente complexa, e só foi validada com a ajuda de um computador [12]. Após o estabelecimento do Teorema 2.8, o estudo de algoritmos relacionados ao problema da coloração de grafos começou a ganhar notoriedade. Com base nos conceitos apresentados, definimos os problemas computacionais que são alvos deste estudo.

Definição 2.1. *Problema do Número Cromático*

Instância: um grafo G.

Resposta: $\chi(G)$

Definição 2.2. *Problema da Coloração Mínima de Vértices*

Instância: um grafo G.

Resposta: uma coloração ótima de $V(G)$.

A solução esperada para o Problema do Número Cromático é um número inteiro. Para o Problema da Coloração Mínima de Vértices, a depender do algoritmo empregado, podemos ter dois formatos de solução para um grafo G: (i) uma partição de $|V(G)|$ em conjuntos independentes, ou (ii) uma função $f : V(G) \to C$ onde C é um conjunto finito de cores e para toda aresta $uv \in E(G)$ temos $f(u) \neq f(v)$.

Existem ainda algumas classes de grafos onde a solução para sua coloração pode ser obtida de forma direta, como visto na Figura 4 e nos resultados a seguir.

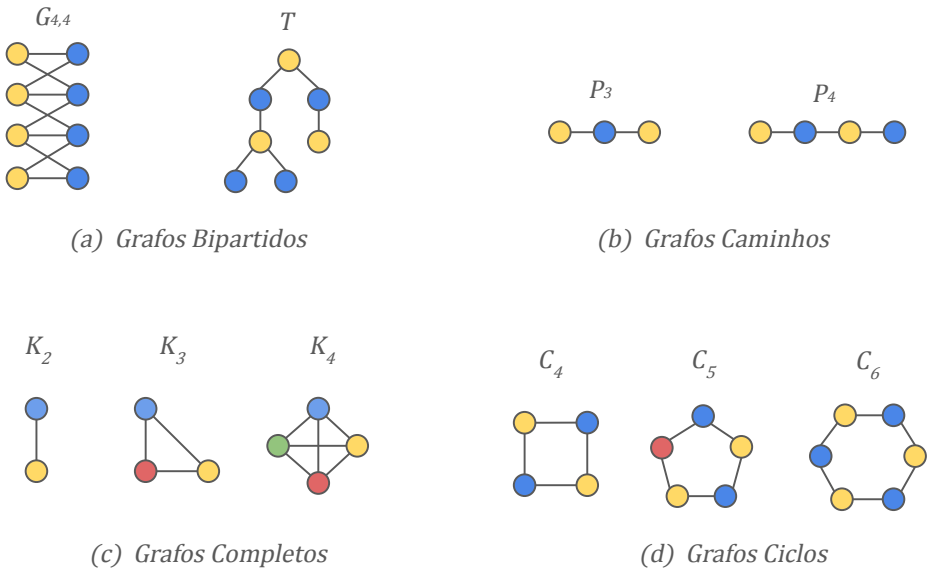


Figura 4: Exemplos de grafos com coloração trivial.

Teorema 2.9. *Um grafo G é bipartido se, e somente se, $\chi(G) \leq 2$.*

Demonstração. Se G é bipartido, então seus vértices podem ser particionados em dois subconjuntos disjuntos U e V, de modo que nenhuma aresta conecta dois vértices do mesmo subconjunto. Podemos colorir todos os vértices de U com uma cor e todos os vértices de V com outra cor, o que implica que $\chi(G) \leq 2$.

Se $\chi(G) \leq 2$, então existe uma coloração dos vértices de G com duas cores. Definimos U como o conjunto de vértices coloridos com uma cor e V como o conjunto

de vértices coloridos com a outra cor. Como nenhum vértice de U está conectado a outro vértice de U, e o mesmo vale para V, segue que G é bipartido. ■

Lema 2.10. *Para todo grafo caminho P_n, segue que $\chi(P_n) = 2$.*

Lema 2.11. *Para todo grafo completo K_n com n vértices, segue que $\chi(K_n) = n$.*

Lema 2.12. *Para todo grafo ciclo C_n, se n é par $\chi(C_n) = 2$, caso contrário $\chi(C_n) = 3$.*

Nos algoritmos de coloração, muitas vezes é necessário aplicar limites superiores e inferiores para o número cromático. Com base em características do grafo, esses limites podem ser estabelecidos de forma direta. Os principais resultados são apresentados a seguir.

Teorema 2.13 (Grau Máximo)**.** *Seja G um grafo e $\Delta(G)$ o grau máximo de G. Então*

$$\chi(G) \leq \Delta(G) + 1.$$

Demonstração. Considere os vértices $v_1, v_2, \ldots, v_n$ de G em qualquer ordem. Vamos colorir esses vértices um a um, atribuindo a menor cor disponível que não tenha sido atribuída aos seus vizinhos.

Por definição, sabe-se que cada vértice v_i tem, no máximo, $\Delta(G)$ vizinhos, logo existem no máximo $\Delta(G)$ cores já utilizadas pelos vizinhos de v_i. Como é permitido atribuir até $\Delta(G) + 1$ cores, sempre haverá pelo menos uma cor disponível para v_i que ainda não foi utilizada por seus vizinhos.

Portanto, ao final do processo, teremos uma coloração válida de G usando no máximo $\Delta(G) + 1$ cores. ■

Ao desconsiderar grafos completos e ciclos ímpares, é possível obter um limitante superior ainda menor, descrito no teorema a seguir, cuja demonstração pode ser encontrada no material de Bondy e Murty [4].

Teorema 2.14 (Brooks)**.** *Se G é um grafo conexo, não é um ciclo ímpar e não é completo, então*

$$\chi(G) \leq \Delta(G).$$

Em relação aos limitantes inferiores para o número cromático de um grafo G, existem dois resultados clássicos. O primeiro está relacionado ao tamanho do clique máximo contido em G e o segundo é baseado no número de independência de G. Ambos os casos estão descrito nos teoremas a seguir.

Teorema 2.15 (Clique Máximo)**.** *Seja G um grafo. Então:*

$$\chi(G) \geq \omega(G).$$

Demonstração. O resultado é consequência direta do Teorema 2.11, pois um clique com k vértice precisa de k cores. ■

Teorema 2.16 (Número de Independência)**.** *Seja G um grafo com n vértices. Então*

$$\chi(G) \leq \frac{n}{\alpha(G)}.$$

Demonstração. Considere uma coloração de um grafo G com $\chi(G)$ cores, em termos de conjuntos independentes:

$$V(G) = V_1 \cup V_2 \cup \ldots \cup V_{\chi(G)}.$$

Como cada conjunto V_i é independente, o tamanho de cada V_i é no máximo $\alpha(G)$. Portanto, para cada i, vale que $|V_i| \leq \alpha(G)$. Como $V_1, V_2, \ldots, V_{\chi(G)}$ é uma partição de $V(G)$, o número total de vértices é a soma do tamanho desses subconjuntos independentes:

$$n = |V_1| + |V_2| + \ldots + |V_{\chi(G)}|.$$

Considerando que $|V_i| \leq \alpha(G)$, vale que $n \leq \chi(G)\alpha(G)$, e isolando $\chi(G)$ o teorema segue. ■

2.3 Conceitos Básicos em Algoritmos

Os problemas que admitem solução via algoritmo podem ser categorizados em classes, com base na natureza da resposta esperada para o problema [14].

Um **problema de decisão** devolve uma resposta binária ("sim" ou "não"), indicando se os dados de entrada atendem ou não às restrições exigidas pelo problema. Decidir se um dado grafo admite uma coloração com k cores é um exemplo de problema de decisão.

Um **problema de busca** tem como objetivo encontrar e exibir uma configuração que atenda as restrições do problema ou demonstrar que ela não existe. Encontrar uma coloração válida para um dado grafo é um exemplo desse tipo de problema.

Em um **problema de otimização**, a busca de uma configuração válida para os dados de entrada é guiada por um critério de otimização, que consiste em determinar valores **ótimos** (máximos ou mínimos) de uma função, denominada **função objetivo**. O Problema do Número Cromático e da Coloração Mínima de Vértices são

problemas de otimização, pois dado um grafo G, busca-se minimizar a quantidade de cores necessárias para colorir G.

Além da natureza das respostas, os problemas também podem ser classificados com base na complexidade dos algoritmos conhecidos para resolvê-los. Dizemos que um problema de decisão pode ser resolvido de forma eficiente quando existe um algoritmo que o resolve em tempo polinomial no tamanho da entrada, ou seja, cujo pior caso é limitado por uma função polinomial.

A **classe P** é composta por todos os problemas de decisão que podem ser resolvidos de forma eficiente. Já a **classe NP** consiste nos problemas de decisão para os quais, dado um conjunto extra de dados (um certificado), é possível verificar se a instância é "sim" em tempo polinomial. No entanto, não se sabe se todos os problemas em NP podem ser resolvidos em tempo polinomial, ou seja, se P = NP. Esse é um dos principais problemas em aberto da teoria da complexidade computacional.

Um problema é classificado como **NP-difícil** se qualquer problema da classe NP pode ser reduzido a ele em tempo polinomial. Se, além disso, o problema pertencer à classe NP, ele é classificado como **NP-completo**. A redução de um problema A a um problema B consiste em transformar qualquer instância de A em uma instância de B de forma eficiente, garantindo que uma solução para B também resolva A. Assim, se um problema NP-completo ou NP-difícil puder ser resolvido eficientemente, então todos os problemas de NP também poderão. Por conta disso, os problemas NP-completos/NP-difíceis são amplamente estudados, tanto para compreender seus limites teóricos quanto para o desenvolvimento de algoritmos de aproximação e heurísticas que permitem tratá-los na prática.

Tanto o Problema do Número Cromático quanto o Problema da Coloração Mínima de Vértices são exemplos de problemas NP-difíceis. Os algoritmos exatos explorados na Seção 4 utilizam as técnicas que estão definidas a seguir, com base nos materiais de Bellman [2] e Goldbarg e Luna [14].

O ***backtracking*** é uma técnica de busca que explora todas as opções de solução de forma recursiva, descartando as soluções inviáveis à medida que avança. Quando uma solução é encontrada, o algoritmo retorna e continua a busca por outras soluções possíveis. Como essa técnica é baseada em força bruta, sua eficiência pode ser afetada em problemas com muitas possibilidades de solução.

O ***branch-and-bound*** é uma técnica de busca exaustiva que organiza sua recursão como uma árvore, onde cada nó representa uma chamada recursiva para um subproblema. Para evitar cálculos desnecessários, usa um critério de poda baseado em uma função de avaliação, que estima um limite inferior para o custo da solução. Se uma chamada ultrapassa esse limite, ela é descartada. Assim, o algoritmo busca

a melhor solução sem explorar todas as possibilidades. Um exemplo dessa abordagem é o Algoritmo de Zykov, descrito na Seção 4.1, que aplica *backtracking* com *branch-and-bound* para resolver o problema da coloração.

A **programação dinâmica** é uma técnica de otimização que parte da ideia de que um problema maior pode ser dividido em subproblemas menores, e a solução ótima para o problema principal pode ser construída a partir das soluções ótimas dos subproblemas. A principal característica da programação dinâmica é o uso de uma estratégia de memorização, na qual a solução de um subproblema é armazenada em uma estrutura de dados, para que possa ser reutilizada quando necessário, evitando assim o recálculo desnecessário. O Algoritmo de Lawler, abordado na Seção 4.2 foi uma das primeiras soluções baseadas em programação dinâmica para o problema da coloração, e seu formato serviu como base para estudos posteriores.

A última ferramenta abordada nesse estudo é a **Programação Linear (PL)**, que lida com problemas de otimização e tem como objetivo maximizar (ou minimizar) uma função linear − denominada **função objetivo** − sujeita a um conjunto de **restrições** lineares. Considere $m, n \in \mathbb{N}^*$, uma matriz $A \in \mathbb{Q}^{n \times m}$ e os vetores $c \in \mathbb{Q}^n, x \in \mathbb{Q}^n_+$ e $b \in \mathbb{Q}^m$, representados como matrizes coluna. A formulação canônica de um programa linear (de minimização, sem perda de generalidade) é dada por

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

O **Algoritmo Simplex**, criado por Dantzig em 1947 [8], é um método iterativo amplamente utilizado para resolver programas lineares. O método busca satisfazer o sistema de equações partindo de uma solução inicial que obedece às restrições do problema, denominada **solução viável**, e tenta melhorá-la a cada iteração, enquanto for possível, até encontrar uma **solução ótima**. Embora tenha complexidade de tempo exponencial no pior caso, o algoritmo Simplex é eficiente em muitos casos práticos.

A **Programação Linear Inteira (PLI)** é uma extensão da Programação Linear, onde o vetor x que armazena as **variáveis de decisão** tem seu domínio restrito ao conjunto dos números inteiros. O problema da Programação Linear é polinomial, enquanto a Programação Linear Inteira é da classe NP-difícil [14]. O método de *branch-and-bound* é a forma mais comum de se resolver um PLI. Esse método divide o problema em subproblemas menores, restringindo o intervalo de valores possível para as variáveis inteiras.

O Problema da Coloração de Vértices pode ser formulado como um problema de PLI. Seja G o grafo que queremos colorir, que tem n vértices. Uma primeira forma de modelar consiste em definir variáveis binárias y_{vc} que indicam se um vértice v é colorido com uma cor específica c. Assim, $y_{vc} = 1$ se e somente se v foi colorido com a cor c. Como o máximo de cores que precisaremos é o número de vértices, então $c \in \{1, 2, \ldots, n\}$. Também teremos uma variável binária x_c que indica se a cor c é usada ou não. Dessa forma, como queremos minimizar o número de cores utilizadas, basta minimizar $\sum_{c=1}^{n} x_c$. Uma primeira restrição que precisamos garantir é que cada vértice tenha uma única cor. A equação (2) da formulação garante isso. Outra restrição que devemos ter é que vértices adjacentes não podem ter a mesma cor. Assim, para toda aresta $uv \in E(G)$, se $y_{uc} = 1$, não podemos ter $y_{vc} = 1$ também. A equação (3) garante que se a cor c é usada, apenas um dentre u e v recebem tal cor. Assim, a formulação completa é descrita a seguir:

$$\text{Minimizar} \quad \sum_{c=1}^{n} x_c \tag{1}$$

$$\text{sujeito a} \quad \sum_{c=1}^{n} y_{vc} = 1, \qquad \text{para todo } v \in V(G) \text{ e todo } c \in \{1, \ldots, n\} \tag{2}$$

$$y_{uc} + y_{vc} \leq x_c, \qquad \text{para todo } uv \in E(G) \text{ e todo } c \in \{1, \ldots, n\} \tag{3}$$

$$y_{vc} \in \{0, 1\}, \qquad \text{para todo } v \in V(G) \text{ e todo } c \in \{1, \ldots, n\} \tag{4}$$

$$x_c \in \{0, 1\}, \qquad \text{para todo } c \in \{1, \ldots, n\}. \tag{5}$$

Uma segunda formulação, proposta por Mehotra e Trick [19], consiste em minimizar a quantidade de conjuntos independentes maximais de um grafo G. A obtenção do número cromático utilizando essa estratégia é garantida pelo teorema a seguir.

Teorema 2.17. *Seja G um grafo. Existe uma coloração ótima de G em que cada cor é um conjunto independente maximal.*

Demonstração. Suponha que $C_1, C_2, \ldots, C_k$ seja uma coloração ótima de G, onde cada C_i é um conjunto independente e $k = \chi(G)$. Se algum C_i não for maximal, significa que podemos adicionar mais vértices sem aparecer uma aresta.

Por construção, expandimos cada C_i até se tornar um conjunto independente maximal. Como os conjuntos não se sobrepõem e cada vértice já estava colorido antes, ainda teremos no máximo k cores.

Ou seja, sempre podemos transformar uma coloração ótima em outra com as mesmas k cores onde todos os conjuntos são independentes maximais. ■

Seja $n = |V(G)|$ e $\mathbb{S}$ o conjunto de todos os conjuntos independentes maximais

de G. Considere que uma variável binária x_S recebe valor 1 se o conjunto S faz parte da solução, e 0 caso contrário. A formulação proposta por Mehotra e Trick pode ser descrita por:

$$\text{Minimizar} \sum_{S \in \mathbb{S}} x_S \tag{1}$$

$$\text{sujeito a} \sum_{\{S \in \mathbb{S} \,:\, v \in S\}} x_S \geq 1, \qquad \text{para todo } v \in V(G) \tag{2}$$

$$x_S \in \{0, 1\}, \qquad \text{para todo } S \in \mathbb{S} \tag{3}$$

Nesta formulação, a função objetivo busca minimizar a quantidade de cores, correspondendo ao menor número de conjuntos independentes maximais necessários para particionar os vértices. A restrição representada pela Equação (2) garante que cada vértice v pertença a pelo menos um conjunto $S \in \mathbb{S}$, ou seja, receba uma cor diferente de seus vizinhos, já que S é um conjunto independente e não contém vértices adjacentes ($N(v) \cap S = \emptyset$).

3 Algoritmos Heurísticos para Coloração

Como o Problema Coloração Mínima é NP-difícil, há uma grande variedade de algoritmos heurísticos que devolvem soluções sub-ótimas em tempo polinomial. Para simplificar a interpretação dos algoritmos apresentados a seguir, as cores disponíveis para coloração estão representadas por números inteiros, e dizemos que um vértice **utiliza uma cor** k caso essa cor seja atribuída a ele.

A primeira abordagem trata-se de um algoritmo guloso que percorre sequencialmente a lista de vértices, atribuindo a cada um deles a menor cor ainda não utilizada por seus vizinhos. Por construção, é garantido que a solução devolvida corresponde a uma **coloração viável**, isto é, uma configuração na qual vértices adjacentes não tenham a mesma cor.

Algoritmo 3.1 Coloração Gulosa Sequencial

Entrada: Um grafo G com n vértices e m arestas.

Saída: Um vetor C com uma coloração de G.

1: Seja C um vetor de tamanho n, indexado por $V(G)$
2: $k \leftarrow 1$
3: **Para cada** $v \in V(G)$ **faça**
4: **Se** existe um inteiro $i \leq k$ ainda não utilizado por $N_G(v)$ **então**
5: $C[v] \leftarrow$ Menor inteiro entre 1 e k ainda não utilizado por $N_G(v)$
6: **Senão**
7: $k \leftarrow k + 1$
8: $C[v] \leftarrow k$
9: **Devolve** C

Nas linhas 1 e 2 inicializamos o vetor-resposta C e a variável k, que representa a quantidade de cores utilizadas para colorir $V(G)$. O laço da linha 3 percorre o conjunto de vértices, sendo executado $n = |V(G)|$ vezes. Para cada $v \in V(G)$, a condição da linha 4 percorre a vizinhança de v em busca do menor inteiro disponível para uso, sendo executada $d_G(v)$ vezes. Se não for encontrado um número, é atribuída uma nova cor $k + 1$ ao vértice. Pelo Lema 2.1, sendo $m = |E(G)|$, são necessárias $2m$ execuções para percorrer a vizinhança de todos os vértices de G. Desse modo, em um grafo de entrada G com n vértices e m arestas, o Algoritmo 3.1 tem complexidade de tempo $O(n + m)$.

Apesar da simplicidade, o Algoritmo 3.1 pode não oferecer a melhor solução, dado que a coloração depende da ordem de visitação dos vértices. A Figura 5 representa a aplicação do Algoritmo 3.1 em um grafo utilizando duas ordenações distintas. Considerando a sequência $S_1 = (u, v, w, x, y, z)$, obtém-se uma coloração

ótima com duas cores. Ao mudar para a sequência $S_2 = (u, x, v, y, w, z)$, o algoritmo devolve uma coloração com três cores, que não configura uma solução ótima para o grafo de entrada.

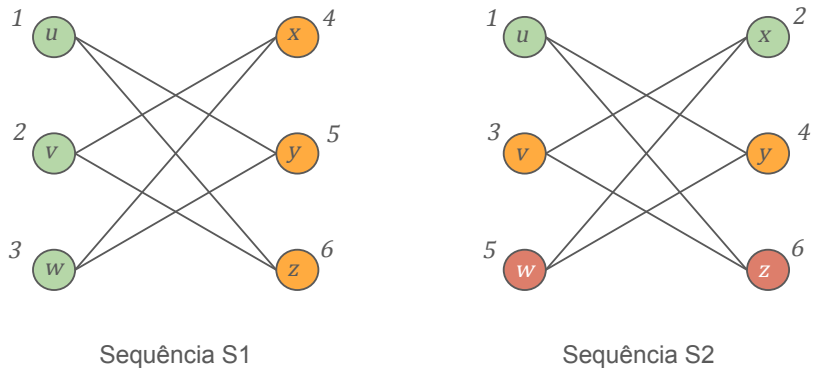


Figura 5: Aplicação do Algoritmo Guloso Sequencial.

A segunda abordagem também se trata de um algoritmo guloso, proposto por Welsh e Powell em 1967 [23]. Essa solução parte da hipótese de que vértices de maior grau são mais difíceis de serem coloridos, e utiliza uma ordenação decrescente de grau para escolher os vértices que serão coloridos. A ideia é que ao atribuir uma cor a um vértice v, essa mesma cor é replicada a todos os vértices não-vizinhos de v e não-adjacentes entre si. Isso claramente garante a viabilidade da coloração.

Algoritmo 3.2 WELSH-POWELL

Entrada: Um grafo G com n vértices e m arestas.
Saída: Um vetor C com uma coloração de $V(G)$.

1: Seja C um vetor de tamanho n inicializado com -1
2: Ordene os vértices $V(G)$ em ordem decrescente de grau
3: $k \leftarrow 1$
4: **Para** cada vértice v em $V(G)$ na ordem decrescente de grau **faça**
5: **Se** $C[v] = -1$ **então** ▷ Se o vértice ainda não foi colorido
6: $C[v] \leftarrow k$
7: **Para** cada vértice u em $V(G)$ **faça**
8: **Se** $C[u] = -1$ e u não é adjacente a nenhum vértice de cor k **então**
9: $C[u] \leftarrow k$
10: $k \leftarrow k + 1$
11: **Devolve** C

Sejam $n = |V(G)|$ e $m = |E(G)|$. O algoritmo começa com a inicialização do vetor-resposta C em -1, indicando que nenhum vértice está colorido. A ordena-

ção dos vértices na linha 2 pode ser realizada via MergeSort, com complexidade $O(n \log n)$. Na linha 3, a variável k é inicializada para representar a quantidade de cores utilizadas na coloração. O laço da linha 4 percorre todos os vértices de G, com custo $O(n)$. Para cada vértice v, o laço da linha 7 verifica todos os n vértices, resultando em $O(n^2)$ operações. Portanto, o algoritmo tem uma complexidade de $O(n^2)$.

Uma outra solução surgiu em 1979 [6], quando Brélaz propôs uma nova heurística pautada na ordenação dinâmica dos vértices, solução descrita pelo algoritmo DSATUR. Esse algoritmo inicia atribuindo uma cor ao vértice de maior grau e, em seguida, percorre os demais vértices na ordem decrescente de **grau de saturação**, que corresponde ao número de cores distintas já atribuídas aos seus vizinhos. A cada iteração, o vértice com maior grau de saturação recebe a menor cor disponível. Em caso de empate, a escolha recai sobre o vértice de maior grau no grafo original. O Algoritmo 3.3 descreve essa ideia, e devolve uma solução ótima para grafos bipartidos.

Algoritmo 3.3 DSATUR

Entrada: Um grafo G com n vértices e m arestas.

Saída: Um vetor C com uma coloração de $V(G)$.

1: Seja C um vetor de tamanho n
2: $k \leftarrow 1$
3: $v \leftarrow$ vértice com maior grau em $V(G)$
4: $C[v] \leftarrow k$
5: Inicialize um heap binário H com os vértices de $V(G) \backslash \{v\}$, usando o grau de saturação do vértice como chave, priorizando o maior grau em caso de empate
6: **Enquanto** H não estiver vazio **faça**
7: $\quad u \leftarrow$ vértice removido de H ▷ maior grau de saturação
8: $\quad$ **Se** existe um inteiro $i \leq k$ ainda não utilizado por $N_G(u)$ **então**
9: $\quad\quad C[u] \leftarrow$ Menor número entre 1 e k ainda não utilizado por $N_G(u)$
10: $\quad$ **Senão**
11: $\quad\quad k \leftarrow k + 1$
12: $\quad\quad C[u] \leftarrow k$
13: $\quad H \leftarrow$ atualiza as chaves de $N_G(u)$ com os novos graus de saturação
14: **Devolve** C

Nas linhas 1 e 2, são inicializados o vetor-resposta C e o inteiro k, que indica a quantidade de cores utilizadas na coloração. Em seguida, a cor 1 é atribuída ao vértice de maior grau em G, e os demais vértices são inseridos no *heap* H, priorizados pelo grau de saturação, com tempo $O(n)$. A cada iteração do laço da linha 6, um

vértice u é removido de H para coloração em tempo $O(\log n)$. Na linha 8, percorre-se $N_G(u)$ para identificar a menor cor disponível, o que exige tempo proporcional ao grau de u, resultando em um custo total de $O(m)$ ao longo da execução do algoritmo.

Após a coloração de u, os graus de saturação dos vértices vizinhos são atualizados no *heap* H, o que requer tempo $O(m \log n)$ no total. Como cada vértice é processado apenas uma vez, o tempo total do algoritmo é $O(m \log n)$.

Um outro algoritmo heurístico é o RLF (*Recursive Largest First*), proposto por Leighton em 1979 [18]. Com uma abordagem recursiva, o RLF, descrito no Algoritmo 3.4, utiliza um estratégia parecida com o Algoritmo 3.2, utilizando uma cor por vez para colorir todos os vértices possíveis. Esse algoritmo procura conjuntos independentes de vértices e associa a cada conjunto uma cor. De forma recursiva, cada conjunto independente já colorido é removido do grafo e o processo é reaplicado no subgrafo resultante até finalizar a coloração. Assim como o Algoritmo 3.3, o RLF também é exato para grafos bipartidos.

Algoritmo 3.4 RLF

Entrada: Um grafo G.

Saída: Uma partição C de $V(G)$ em conjuntos independentes.

1: Seja C um conjunto vazio.
2: $k \leftarrow 0$, $X \leftarrow V(G)$ e $Y \leftarrow \emptyset$
3: **Enquanto** X não estiver vazio **faça**
4: $\quad k \leftarrow k + 1$
5: $\quad C_k \leftarrow \emptyset$
6: $\quad$ **Enquanto** X não estiver vazio **faça**
7: $\quad\quad v \leftarrow$ vértice de X
8: $\quad\quad C_k \leftarrow C_k \cup \{v\}$
9: $\quad\quad Y \leftarrow Y \cup N_X(v)$ ▷ Vizinhos de v no subgrafo induzido por X
10: $\quad\quad X \leftarrow X \backslash (Y \cup \{v\})$
11: $\quad C \leftarrow C \cup C_k$
12: $\quad X \leftarrow Y$ e $Y \leftarrow \emptyset$
13: **Devolve** C

O algoritmo inicia com a criação do conjunto de resposta C, a variável k para representar o número de cores, um conjunto X que recebe os vértices de G e um conjunto vazio Y. A cada iteração do laço externo (linha 3), uma nova classe de cor C_k é criada. No laço interno (linha 5), um vértice v de X é adicionado à classe atual, e seus vizinhos no subgrafo induzido por X são movidos para Y, sinalizando que não podem mais receber a mesma cor. Tanto v quanto seus vizinhos são removidos de X. Quando X estiver vazio, a classe de cor C_k é finalizada e adicionada à solução

C, e os vértices de Y são realocados em X para formar a próxima classe.

Leighton [18] provou que a complexidade do Algoritmo 3.4 no pior caso é $O(n^3)$. Isso se deve principalmente ao custo da manutenção e atualização dos conjuntos X e Y, bem como à verificação das adjacências durante a construção das classes de cor. A cada iteração do laço interno, é necessário calcular a vizinhança de um vértice no subgrafo induzido por X, operação que pode demandar $O(n)$. Como essa verificação pode ocorrer até $O(n)$ vezes por classe de cor, e até $O(n)$ classes podem ser necessárias no pior caso, o tempo total se acumula em $O(n^3)$.

4 Algoritmos Exatos para Coloração

Nesta seção abordamos alguns algoritmos exatos para o problema de Coloração de Vértices, que devolvem uma solução ótima para qualquer grafo de entrada. Os algoritmos são baseados nas estratégias de *backtracking*, *branch-and-bound*, Programação Dinâmica e Programação Linear Inteira (PLI).

4.1 Algoritmo de Zykov

O Algoritmo de Zykov [24] utiliza operações de contração de vértices e adição de arestas, gerando a cada iteração grafos modificados que preservam as relações de vizinhança do grafo original. As soluções são enumeradas em uma estrutura denominada Árvore de Zykov, onde o ramo esquerdo armazena o resultado da operação de contração e o direito de adição. Essas duas operações exploram as duas configurações possíveis entre dois vértices não-adjacentes arbitrários: ou eles possuem a mesma cor ou possuem cores distintas. Sejam u, v dois vértices não-adjacentes em um grafo G e sejam G'_{uv} e G''_{uv} os grafos obtidos de G pela contração dos vértices u e v e pela adição da aresta uv, respectivamente. Observe que qualquer coloração de G'_{uv} nos permite gerar uma coloração de G em que u e v recebem a mesma cor e qualquer coloração de G''_{uv} nos permite gerar uma coloração de G em que u e v têm cores diferentes. Assim, uma coloração ótima para G só pode ser a melhor dentre as colorações ótimas de G'_{uv} ou G''_{uv}. A ramificação termina quando as operações geram um grafo completo, onde a coloração é trivial.

Para reduzir o espaço de busca, o algoritmo utiliza uma estratégia de *branch-and-bound*, que estabelece um limitante superior conforme uma solução com menos cores é encontrada. Como critério adicional de redução de espaço de busca, é possível incluir outros limitantes superiores, como $\Delta(G) + 1$, por exemplo.

A principal ideia da solução proposta por Zykov está descrita no Algoritmo 4.1.

Algoritmo 4.1 Cor

Entrada: Um grafo G e um inteiro q com o valor da melhor solução até o momento.
Saída: O número cromático de G.

1: $n \leftarrow |V(G)|$
2: **Se** G é um grafo completo tal que $|V(G)| = n$ **então**
3: **Devolve** $\min\{n, q\}$
4: **Senão**
5: Tome dois vértices não adjacentes $u, v \in V(G)$
6: $q' \leftarrow \text{Cor}(G'_{uv}, q)$ ▷ G'_{uv} é o grafo obtido pela contração dos vértices u e v
7: $q'' \leftarrow \text{Cor}(G''_{uv}, q)$ ▷ G''_{uv} é o grafo obtido pela adição da aresta uv
8: **Devolve** $\min\{q', q''\}$

O procedimento Cor recebe como parâmetro de entrada um grafo G e um limitante superior q para $\chi(G)$. Na linha 2, o condicional verifica se G é um grafo completo, devolvendo a resposta trivial $\chi(G) = |V(G)|$. Caso G não seja um grafo completo, dois vértices u e v não-adjacentes são selecionados e o procedimento é chamado recursivamente com o grafo G transformado pelas operações de contração e adição, nas linhas 6 e 7, respectivamente. Na linha 8 o procedimento devolve o menor número cromático obtido entre as duas ramificações, e dado que u e v não são adjacentes, esse valor também se estende como $\chi(G)$. Por fim, o Algoritmo 4.2 descreve a obtenção do número cromático com o uso do procedimento Cor.

Algoritmo 4.2 Zykov

Entrada: Um grafo G.

Saída: O número cromático de G.

1: $k \leftarrow$Cor$(G, \Delta(G) + 1)$

2: **Devolve** k

A altura de uma Árvore de Zykov é determinada pela quantidade de arestas que faltam para que o grafo seja completo. Dado um grafo G com n vértices, o pior caso se dá quando $E(G) = \emptyset$, onde precisaremos de exatamente $\frac{n(n-1)}{2}$ arestas para tornar G completo. Como a Árvore de Zykov é estritamente binária, o número máximo de nós é dado por $\sum_{i=0}^{\ell} 2^i$, onde ℓ é a altura da árvore. Substituindo ℓ pela altura máxima definida anteriormente, temos $2^{\frac{n(n-1)}{2}}$ nós em uma árvore de Zykov, justificando que o Algoritmo 4.2 tenha complexidade de $O(2^{n^2})$ no pior caso.

Com pequenas modificações nos Algoritmos 4.1 e 4.2, é possível devolver uma coloração ótima ao invés do número cromático. Basta associar uma mesma cor aos vértices contraídos, e quando a iteração chegar em um grafo completo (folha), finalizamos associando cores distintas aos vértices remanescentes (ainda não contraídos). Essa ação pode ser realizada tanto no formato de vetor, quanto no formato de conjuntos independentes, onde os vértices que sofrerem contração são incluídos em um mesmo conjunto de cor.

Na Figura 6 está representada uma execução parcial de uma árvore de Zykov, que comprova que o grafo da Figura 5 tem número cromático igual a 2. Em cada nó, a ramificação à esquerda, representada pelo grafo G', considera que os dois vértices selecionados possuem a mesma cor, enquanto a ramificação à direita, representada pelo grafo G'' assume que eles têm cores distintas. Por fim, o limitante superior q é atualizado com o menor valor obtido entre G' e G'', e como o procedimento recursivo verifica todas as possibilidades, é garantido que o valor final de q armazena o número cromático de G.

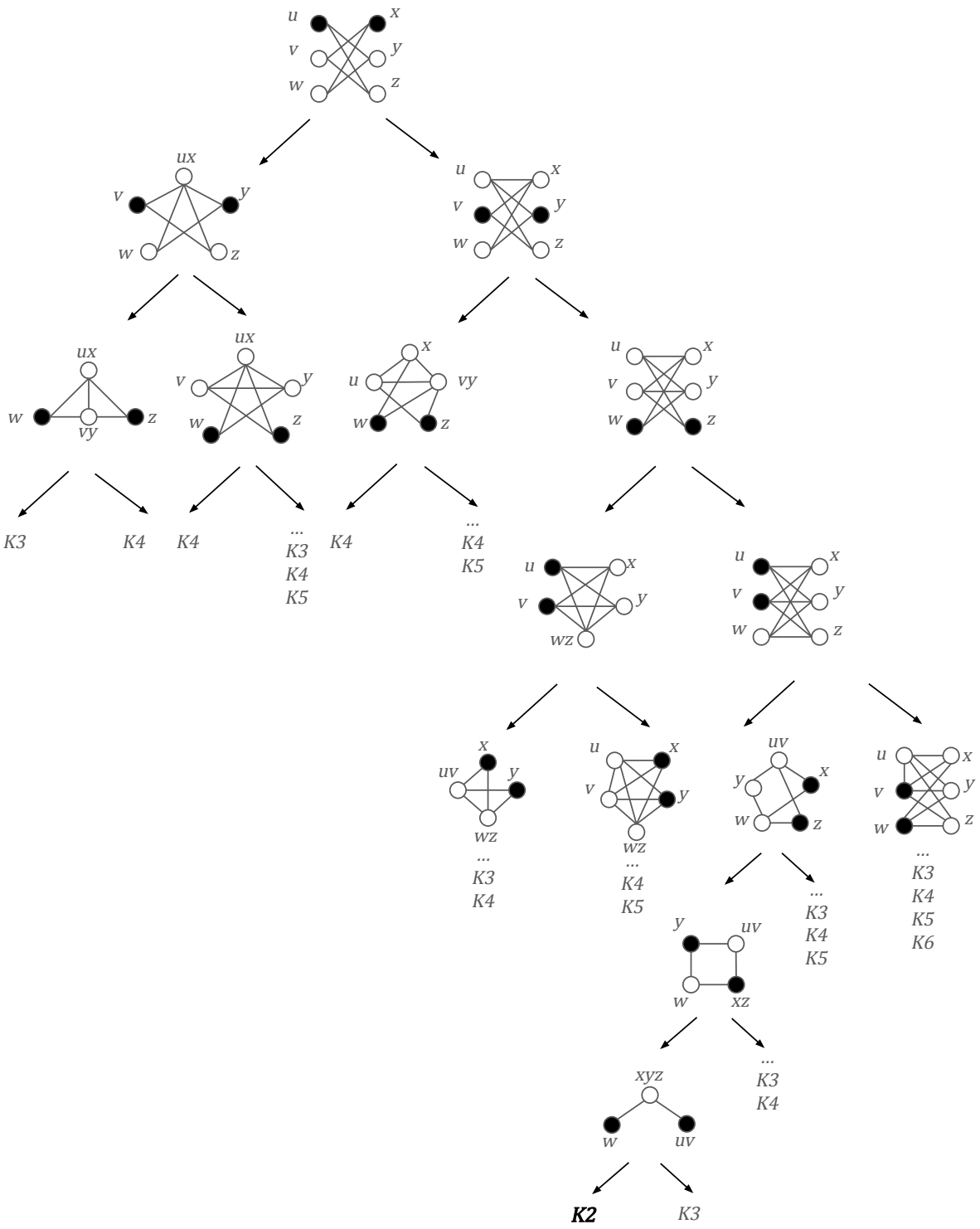


Figura 6: Exemplo da Árvore de Zykov para o grafo da Figura 5.

O Algoritmo 4.2, proposto por Zykov em 1962, foi um dos primeiros algoritmos exatos de *Branch-and-Bound* para coloração de grafos. Em 1972, Brown [5] introduziu uma nova proposta de algoritmo, desenvolvendo melhores limitantes para reduzir o espaço de busca de forma mais eficaz, embora a complexidade de tempo tenha permanecido exponencial.

Em 1979, Daniel Brélaz [6] propôs um algoritmo exato que incorpora o conceito de grau de saturação, utilizado no algoritmo DSATUR, apresentado na Seção 3. Ao longo do tempo, sua abordagem passou por diversas adaptações, incluindo a versão desenvolvida por San Segundo [21], que se destaca como um dos algoritmos mais eficientes baseados em *Branch-and-Bound* para o problema da coloração.

4.2 Algoritmo de Lawler

O algoritmo proposto por Lawler [17] utiliza um resultado que garante que existe uma coloração ótima de um grafo G onde uma das cores é um conjunto independente maximal de $V(G)$.

Teorema 4.1 (Wang [22])**.** *Dado um grafo G, para todo vértice $v \in V(G)$, sejam $I_1, \ldots, I_\ell$ os conjuntos independentes maximais que contêm v. Existe uma coloração ótima de G tal que uma das suas cores é I_i, para algum $1 \leq i \leq \ell$.*

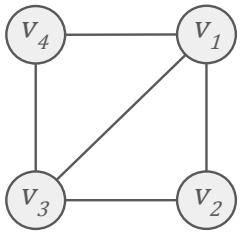
Demonstração. Seja $\{P_1, \ldots, P_k\}$ uma coloração ótima de G. Sem perda de generalidade, seja P_1 um conjunto não maximal contido em I_i, para algum $i \in \{1, \ldots, \ell\}$. Sejam $\{P'_2, \ldots, P'_k\}$ os conjuntos obtidos da remoção dos vértices de I_i dos conjuntos $\{P_2, \ldots, P_k\}$. Dessa forma, $\{I_i, P_2, \ldots, P_k\}$ é uma coloração ótima de G que usa um conjunto independente maximal. ■

Assim, se I é tal conjunto independente maximal garantido pelo Teorema 4.1, o número cromático de um grafo G é igual a $1 + \chi(G - I)$. A ideia do algoritmo é percorrer todos os conjuntos independentes maximais do grafo para encontrar tal conjunto I. Ele devolve a solução que utilizar a menor quantidade de cores. Como todas as combinações possíveis são avaliadas e a configuração de menor valor é priorizada, garante-se que o algoritmo devolve uma solução ótima.

Observamos que essa solução recursiva, isto é, em que para calcular $\chi(G)$ precisa-se ter calculado $\chi(G - I)$ para diferentes conjuntos $I \subseteq V(G)$, pode ser melhorada com auxílio de programação dinâmica. Como existem no máximo 2^n subconjuntos de vértices, onde $n = |V(G)|$, é possível utilizar-se de um vetor de tamanho 2^n tal que cada posição armazena o número cromático do subgrafo induzido por algum dos subconjuntos.

Seja $V(G) = \{v_1, v_2, \ldots, v_n\}$. Vamos denotar cada um dos 2^n subconjuntos possíveis por $\mathbb{S}_i$, onde $i \in \{0, \ldots, 2^n - 1\}$. A representação binária do índice i de um conjunto $\mathbb{S}_i$ indicará quais vértices estão em $\mathbb{S}_i$. Por exemplo, quando $i = 5$, a representação é 0101, o que significa $\mathbb{S}_5 = \{v_1, v_3\}$. Formalmente, se a representação binária de i é a sequência $s_n s_{n-1} \ldots s_1$, com cada $s_j \in \{0, 1\}$, então $\mathbb{S}_i = \{v_j \in$

$V(G)\colon s_j = 1\}$. Na Figura 7 enumeramos todos os subconjuntos de vértices de um grafo, como exemplo.



$\mathbb{S}_0 = \emptyset$ $\quad \mathbb{S}_5 = \{v_1, v_3\}$ $\quad \mathbb{S}_{11} = \{v_1, v_2, v_4\}$
$\mathbb{S}_1 = \{v_1\}$ $\quad \mathbb{S}_6 = \{v_2, v_3\}$ $\quad \mathbb{S}_{12} = \{v_3, v_4\}$
$\mathbb{S}_2 = \{v_2\}$ $\quad \mathbb{S}_7 = \{v_1, v_2, v_3\}$ $\quad \mathbb{S}_{13} = \{v_1, v_3, v_4\}$
$\mathbb{S}_3 = \{v_1, v_2\}$ $\quad \mathbb{S}_8 = \{v_4\}$ $\quad \mathbb{S}_{14} = \{v_2, v_3, v_4\}$
$\mathbb{S}_4 = \{v_3\}$ $\quad \mathbb{S}_9 = \{v_1, v_4\}$ $\quad \mathbb{S}_{15} = \{v_1, v_2, v_3, v_4\}$
$\mathbb{S}_{10} = \{v_2, v_4\}$

Figura 7: Grafo H e seus 16 subconjuntos possíveis de vértices.

Finalmente, com essa notação, podemos definir o vetor auxiliar do algoritmo de Lawler. Usaremos um vetor X de tamanho 2^n cuja i-ésima posição armazena o número cromático do subgrafo de G induzido pelo conjunto $\mathbb{S}_i$, isto é, $X[i] = \chi(G[\mathbb{S}_i])$. Claramente, se $\mathbb{S}_i$ já é um conjunto independente, teremos $X[i] = 1$. O caso base é o subconjunto vazio de vértices, de forma que $X[0] = 0$. O Algoritmo 4.3 formaliza a ideia da programação dinâmica.

Algoritmo 4.3 LAWLER

Entrada: Um grafo G.

Saída: O número cromático de G.

1: $n \leftarrow |V(G)|$
2: $X \leftarrow$ vetor indexado entre 0 e $2^n - 1$
3: $X[0] \leftarrow 0$
4: **Para** $i \leftarrow 1$ até $2^n - 1$ **faça**
5: $\quad X[i] \leftarrow \infty$
6: $\quad$ **Para cada** conjunto independente maximal $I \subseteq V(G[\mathbb{S}_i])$ **faça**
7: $\quad\quad \mathbb{S}_j \leftarrow \mathbb{S}_i \setminus I$
8: $\quad\quad X[i] \leftarrow \min\{X[i], X[j] + 1\}$
9: **Devolve** $X[2^n - 1]$

O algoritmo começa com a criação do vetor X e inicialização, na linha 3, do caso base. O laço da linha 4 percorre cada $\mathbb{S}_i$ e avalia, no laço da linha 6, todos os conjuntos independentes maximais I do grafo $G[\mathbb{S}_i]$. Para cada conjunto I, o algoritmo busca a posição j de X correspondente ao subconjunto $\mathbb{S}_i \setminus I$ e, na linha 8, o valor de $X[i]$ é atualizado para ser o mínimo entre seu valor atual e $X[j] + 1$. Isso reflete a ideia de que, para colorir $G[\mathbb{S}_i]$, é necessário uma cor a mais do que $G[\mathbb{S}_i \setminus \mathbb{I}]$.

Como essa atualização é feita para todos os conjuntos independentes maximais $\mathbb{I}$, é garantido que $X[i]$ armazene o número mínimo de cores necessário para colorir $\mathbb{S}_i$.

Como a última posição de X representa o conjunto de vértices equivalente a $V(G)$, fica armazenado nesta posição o resultado de $\chi(G)$.

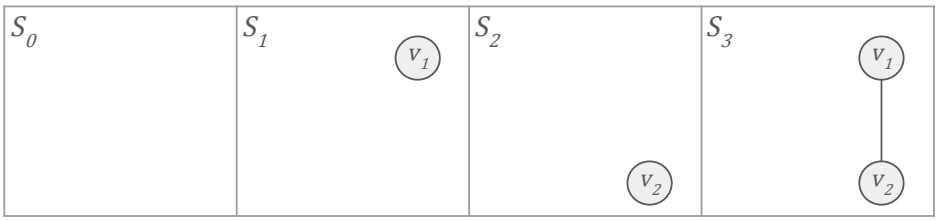
O Algoritmo 4.3 executa em tempo $O(2.4423^n mn)$ em um grafo com n vértices e m arestas. Para demonstrar esse resultado, Lawler partiu de um estudo realizado por Moon e Moser [20], de que a quantidade máxima de conjuntos independentes de um grafo é $3^{n/3}$. Esse algoritmo foi proposto em 1976 e permaneceu como o de melhor desempenho no pior caso por 25 anos. Em 2001, Eppstein [10] propôs uma melhoria desse algoritmo, onde limitava superiormente o tamanho dos conjuntos independentes maximais visitados pelo algoritmo, obtendo uma complexidade de tempo de $O(2.4150^n)$.

Uma segunda melhoria foi apresentada por Byskov [7] em 2002, com complexidade de tempo de $O(2.4023^n)$. Byskov obteve um melhor resultado ao considerar em seu algoritmo uma sugestão apresentada por Eppstein, de incluir uma marcação dos subgrafos 4-coloríveis antes da inicialização do vetor X, que o permitia estabelecer como limitante superior os conjuntos independentes maximais de tamanhos ainda menores, como visto no trabalho de Eppstein.

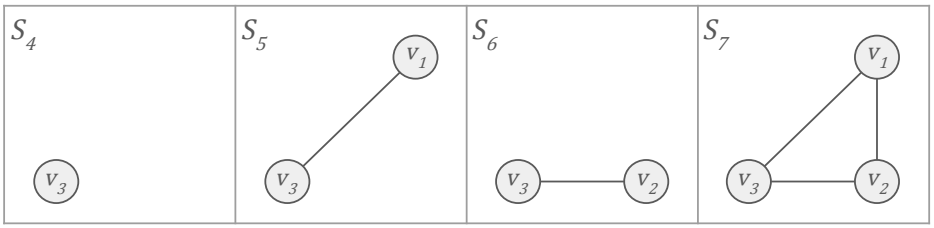
Por fim, um outro trabalho interessante, que se diferenciou dos demais algoritmos já conhecidos, foi o proposto por Bodlaender e Kratsch [3] em 2006. Este é o primeiro algoritmo de programação dinâmica que resolve o problema da coloração de vértices utilizando espaço polinomial, com complexidade de tempo de $O(5.283^n)$ no pior caso.

Na Figura 8, está representada uma execução do Algoritmo 4.3 para o grafo da Figura 7. No vetor X, encontra-se o resultado do número cromático de cada subgrafo de H. Na primeira posição de X associa-se o valor zero, como descrito na linha 3 do algoritmo. Em seguida, inicia-se a iteração da linha 4 com a avaliação do subgrafo $\mathbb{S}_1$. Então $X[1]$ é inicializado com ∞ (linha 5) e como $\mathbb{S}_1$ tem apenas um vértice, o único conjunto independente maximal é o próprio $\mathbb{S}_1$, portanto, o índice de $\mathbb{S}_1 \setminus \mathbb{S}_1$ é zero, e $X[1] \leftarrow \min\{\infty, X[0] + 1\} = 1$. De modo análogo, o valor 1 é associado a todas as posições do vetor X que representam subgrafos de um único vértice. No fim das iterações, em $X[15]$, encontra-se $\chi(H)$.

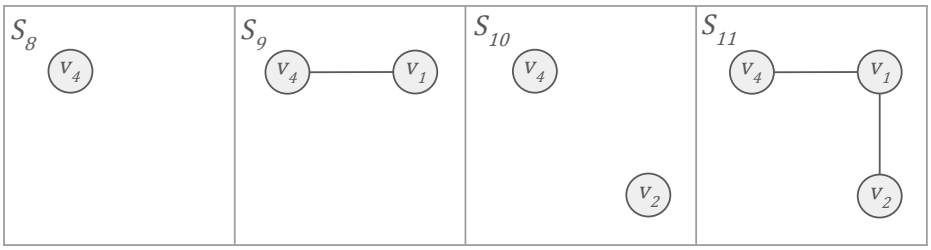
Note que, a cada iteração i, são resgatados valores de posições anteriores a i, característica padrão dos algoritmos de programação dinâmica.



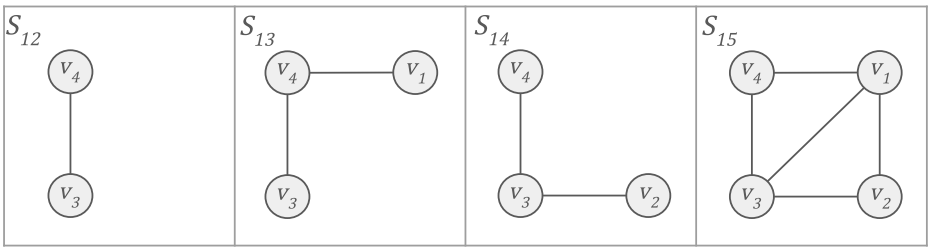
X[0] = 0 | X[1] = X[0] + 1 **X[1] = 1** | X[2] = X[0] + 1 **X[2] = 1** | X[3] = X[1] + 1 **X[3] = 2**



X[4] = X[0] + 1 **X[4] = 1** | X[5] = X[1] + 1 **X[5] = 2** | X[6] = X[2] + 1 **X[6] = 2** | X[7] = X[6] + 1 **X[7] = 3**



X[8] = X[0] + 1 **X[8] = 1** | X[9] = X[1] + 1 **X[9] = 2** | X[10] = X[0] + 1 **X[10] = 1** | X[11] = X[1] + 1 **X[11] = 2**



X[12] = X[4] + 1 **X[12] = 2** | X[13] = X[8] + 1 **X[13] = 2** | X[14] = X[4] + 1 **X[14] = 2** | X[15] = X[5] + 1 **X[15] = 3**

Figura 8: Aplicação do Algoritmo de Lawler no grafo da Figura 7.

4.3 Algoritmo baseado em PLI

O Algoritmo 4.4 utiliza a formulação do problema de coloração apresentada na Seção 2.3, como um Programa Linear Inteiro (PLI). Primeiramente, ele define uma função objetivo que busca minimizar o número de cores utilizadas, criando variáveis binárias $x[i][c]$ para indicar se um vértice i recebe a cor c. Em seguida, impõe restrições que garantem que cada vértice receba exatamente uma cor e que vértices adjacentes não compartilhem a mesma cor.

Algoritmo 4.4 Algoritmo de Coloração via PLI

Entrada: Um grafo G com n vértices e uma lista de adjacência.

Saída: O número cromático de G.

Criar um problema de programação linear inteira (PLI)
Definir a função objetivo para minimizar o número de cores
Criar variáveis binárias $x[i][c]$ indicando se o vértice i recebe a cor c
Para cada vértice i em $V(G)$ **faça**
    Adicionar restrição: $\sum_{c=1}^{n} x[i][c] = 1$
Para cada aresta $uv \in E(G)$ **faça**
    Para cada cor $c \in \{1, \ldots, n\}$ **faça**
        Adicionar restrição: $x[u][c] + x[v][c] \leq 1$
Adicionar restrição: $\sum_{c=1}^{n} x[c][c] \geq 1$
Resolver o problema de PLI para obter uma solução ótima em x.
$k \leftarrow 0$
Para cada cor $c \in \{1, \ldots, n\}$ **faça**
    Para cada vértice $i \in V(G)$ **faça**
        Se $x[i][c] = 1$ **então**
            $k \leftarrow \max\{k, c\}$
            break
Devolve k

Por conta da dificuldade em definir a complexidade dos algoritmos de programação linear, a análise de desempenho é, em grande parte, feita de forma experimental. A complexidade do Algoritmo 4.4 é dominada pela resolução do problema de PLI, que pertence à classe NP-difícil. O tempo de execução pode ser exponencial nos piores casos, pois depende do desempenho do *solver* e do tamanho do grafo.

Na formulação usada pelo Algoritmo 4.4, o número de variáveis e restrições cresce de forma polinomial com o tamanho da entrada, mas a presença de simetrias entre soluções pode dificultar seu uso prático. Dizemos que uma solução é **simétrica** quando pode ser representada de diferentes formas ao permutar os valores atribuídos

às variáveis, sem alterar seu significado. No contexto do problema, isso significa que diferentes atribuições numéricas podem corresponder à mesma solução, gerando $P(n, k)$ permutações equivalentes, onde n representa o numero de vértices e k a quantidade de cores usadas na coloração. Essa redundância aumenta o tamanho da árvore de busca do *branch-and-bound* empregado no *solver* do PLI, prejudicando seu desempenho.

A formulação baseada em conjuntos independentes maximais, vista na Seção 2.3, contorna o problema da simetria. Porém, o número de variáveis utilizadas nessa formulação pode ser muito maior em relação à formulação utilizada pelo Algoritmo 4.4, dado que o número de conjuntos independentes maximais de um grafo G pode ser exponencial em relação à $|V(G)|$.

Mehrotra e Trick [19] propuseram, em 2001, um algoritmo de coloração baseado na formulação do PLI por conjuntos independentes maximais. Esse algoritmo utiliza o método *branch-and-price*, obtido pela combinação da técnica de *branch-and-bound* com o método de **geração de colunas**, que consiste em resolver uma versão reduzida do problema inicial e inserir novas variáveis iterativamente, conforme necessário, para melhorar a solução atual do problema reduzido. O método *branch-and-price* supera as limitações impostas ao tamanho da entrada, permitindo que o algoritmo proposto por Methotra e Trick obtenha os melhores resultados experimentais para o problema de coloração de grafos, como observado no trabalho de Held [16].

5 Resultados

Os algoritmos apresentados nas Seções 3 e 4 foram implementados na linguagem de programação C++ e executados em um computador com processador Intel Core i5-8265U, 8 GB de memória RAM e Linux 64 bits. O código-fonte e os arquivos das instâncias utilizadas no testes encontram-se disponíveis no repositório GitHub[1].

O primeiro grupo de casos de teste é composto por 12 grafos gerados de forma aleatória usando o modelo $G(n, p)$, em que cada um deles tem n vértices e probabilidade p de existir uma aresta entre dois vértices quaisquer. Os valores de n e p para cada instância encontram-se na Tabela 1.

Tabela 1: Primeiro grupo casos de teste - Grafos aleatórios.

G	n	p	G	n	p
G_1	10	25%	G_7	1000	25%
G_2	10	50%	G_8	1000	50%
G_3	10	75%	G_9	1000	75%
G_4	100	25%	G_{10}	10000	25%
G_5	100	50%	G_{11}	10000	50%
G_6	100	75%	G_{12}	10000	75%

No segundo grupo de casos de teste foram utilizadas 26 instâncias do DIMACS [9], um conjunto de grafos amplamente utilizado pela comunidade científica que contempla instâncias de diferentes dimensões e aplicações. Cada grafo G possui n vértices e m arestas, e seu número cromático $\chi(G)$ também é conhecido. As instâncias selecionadas estão descritas na Tabela 2.

Tabela 2: Segundo grupo de casos de teste - DIMACS.

G	$\chi(G)$	n	m	G	$\chi(G)$	n	m
queen5_5	5	25	160	le450_15b	15	450	8169
queen6_6	7	36	290	le450_15c	15	450	16680
queen7_7	7	49	476	le450_15d	15	450	16750
queen8_8	9	64	728	le450_25a	25	450	8260
queen8_12	12	96	1368	le450_25b	25	450	8263
queen9_9	10	81	2112	le450_25c	25	450	17343
queen11_11	11	121	3960	le450_25d	25	450	17425
queen13_13	13	169	6656	miles250	8	128	387
le450_5a	5	450	5714	miles500	20	128	1170
le450_5b	5	450	5734	miles750	31	128	2113
le450_5c	5	450	9803	miles1000	42	128	3216
le450_5d	5	450	9757	miles1500	73	128	5198
le450_15a	15	450	8168	myciel5	6	47	236

[1] https://github.com/raphaelramos97/graphColoring_pgc

No primeiro grupo de teste, o objetivo é avaliar o desempenho de cada heurística em instâncias de diferentes tamanhos, verificando também como o resultado se comporta em diferentes valores de densidade. No segundo grupo de teste, como o valor do número cromático é conhecido, também é possível verificar o quão próxima a resposta fornecida pela heurística está da solução exata.

A Tabela 3 mostra os resultados dos testes dos quatro algoritmos heurísticos de coloração (Sequencial, Welsh-Powell, DSATUR e RFL) para cada instância do primeiro grupo de teste. Para cada algoritmo foi obtido o tempo de execução **T(s)** em segundos e a quantidade **k** de cores utilizadas pela realizar a coloração. Em cada instância, a menor quantidade de cores obtida está sinalizada em negrito.

Tabela 3: Desempenho das heurísticas em grafos aleatórios.

G	Sequencial		Welsh-Powell		DSATUR		RFL	
	T(s)	k	T(s)	k	T(s)	k	T(s)	k
G_1	$2 \cdot 10^{-5}$	**3**	$2 \cdot 10^{-5}$	**3**	$3 \cdot 10^{-5}$	**3**	$9 \cdot 10^{-5}$	**3**
G_2	$2 \cdot 10^{-5}$	4	$2 \cdot 10^{-5}$	**3**	$3 \cdot 10^{-5}$	**3**	$1 \cdot 10^{-4}$	4
G_3	$3 \cdot 10^{-5}$	**6**	$3 \cdot 10^{-5}$	**6**	$3 \cdot 10^{-5}$	**6**	$2 \cdot 10^{-4}$	**6**
G_4	$2 \cdot 10^{-4}$	13	$2 \cdot 10^{-4}$	**11**	$3 \cdot 10^{-4}$	**11**	$2 \cdot 10^{-3}$	13
G_5	$3 \cdot 10^{-4}$	21	$4 \cdot 10^{-4}$	**20**	$5 \cdot 10^{-4}$	**20**	$4 \cdot 10^{-3}$	21
G_6	$4 \cdot 10^{-4}$	35	$8 \cdot 10^{-4}$	**33**	$5 \cdot 10^{-4}$	**33**	$7 \cdot 10^{-3}$	35
G_7	$1 \cdot 10^{-2}$	65	$2 \cdot 10^{-2}$	**61**	$2 \cdot 10^{-2}$	64	$3 \cdot 10^{-1}$	65
G_8	$2 \cdot 10^{-2}$	127	$7 \cdot 10^{-2}$	**122**	$2 \cdot 10^{-2}$	125	$9 \cdot 10^{-1}$	127
G_9	$3 \cdot 10^{-2}$	218	$2 \cdot 10^{-1}$	**208**	$3 \cdot 10^{-2}$	215	2	218
G_{10}	1	420	9	**414**	2	419	$2 \cdot 10^{2}$	420
G_{11}	2	884	$4 \cdot 10^{1}$	**875**	3	878	$7 \cdot 10^{2}$	884
G_{12}	3	1586	$1 \cdot 10^{2}$	**1571**	4	1581	$1 \cdot 10^{3}$	1586

O Algoritmo Sequencial apresentou o menor tempo de execução em todas as instâncias, seguido pelo DSATUR e Welsh-Powell. Em contrapartida, o Algoritmo de Welsh-Powell obteve melhores resultados, devolvendo o menor número de cores na maioria dos grafos. No geral, dentre todos os algoritmos, o DSATUR obteve o melhor equilíbrio entre tempo de execução e qualidade da resposta, e o Algoritmo RFL apresentou os piores resultados, principalmente em grafos de maior densidade e maior quantidade de vértices.

A Tabela 4 mostra os resultados do mesmo teste aplicados em instâncias do segundo grupo. Em algumas instâncias, os algoritmos devolveram uma solução exata, que encontra-se **sublinhada** na tabela, e nos casos de empate, a solução obtida em menor tempo de processamento foi sinalizada com um asterisco **(*)**. Nesse segundo teste, o Algoritmo DSATUR foi o que obteve respostas mais próximas da solução ótima, e novamente o RFL apresentou os piores resultados, tanto em tempo de execução quanto em proximidade da solução ótima.

Tabela 4: Desempenho das heurísticas em instâncias DIMACS.

G	$\chi(G)$	Sequencial		Welsh-Powell		DSATUR		RLF	
		T(s)	k	T(s)	k	T(s)	k	T(s)	k
queen5_5	5	$3 \cdot 10^{-5}$	8	$2 \cdot 10^{-5}$	**7**	$6 \cdot 10^{-5}$	**7**	$3 \cdot 10^{-4}$	8
queen6_6	7	$5 \cdot 10^{-5}$	11	$4 \cdot 10^{-5}$	**9**	$1 \cdot 10^{-4}$	10	$5 \cdot 10^{-4}$	11
queen7_7	7	$8 \cdot 10^{-5}$	**10**	$6 \cdot 10^{-5}$	13	$1 \cdot 10^{-4}$	12	$7 \cdot 10^{-4}$	**10**
queen8_8	9	$1 \cdot 10^{-4}$	**13**	$9 \cdot 10^{-5}$	**13**	$2 \cdot 10^{-4}$	15	$1 \cdot 10^{-3}$	**13**
queen8_12	12	$3 \cdot 10^{-4}$	**15**	$2 \cdot 10^{-4}$	16	$3 \cdot 10^{-4}$	**15**	$3 \cdot 10^{-3}$	**15**
queen9_9	10	$2 \cdot 10^{-4}$	16	$2 \cdot 10^{-4}$	16	$3 \cdot 10^{-4}$	**15**	$2 \cdot 10^{-3}$	16
queen11_11	11	$3 \cdot 10^{-4}$	**17**	$3 \cdot 10^{-4}$	18	$4 \cdot 10^{-4}$	18	$4 \cdot 10^{-3}$	**17**
queen13_13	13	$6 \cdot 10^{-4}$	21	$7 \cdot 10^{-4}$	**20**	$7 \cdot 10^{-4}$	22	$7 \cdot 10^{-3}$	21
le450_5a	5	$5 \cdot 10^{-4}$	14	$6 \cdot 10^{-4}$	**12**	$2 \cdot 10^{-3}$	**12**	$9 \cdot 10^{-3}$	14
le450_5b	5	$5 \cdot 10^{-4}$	13	$6 \cdot 10^{-4}$	**11**	$2 \cdot 10^{-3}$	**11**	$9 \cdot 10^{-3}$	13
le450_5c	5	$8 \cdot 10^{-4}$	17	$7 \cdot 10^{-4}$	13	$2 \cdot 10^{-3}$	**12**	$1 \cdot 10^{-2}$	17
le450_5d	5	$8 \cdot 10^{-4}$	18	$8 \cdot 10^{-4}$	**13**	$2 \cdot 10^{-3}$	**13**	$1 \cdot 10^{-2}$	18
le450_15a	15	$7 \cdot 10^{-4}$	22	$1 \cdot 10^{-3}$	**18**	$2 \cdot 10^{-3}$	19	$1 \cdot 10^{-2}$	22
le450_15b	15	$7 \cdot 10^{-4}$	22	$1 \cdot 10^{-3}$	**17**	$2 \cdot 10^{-3}$	18	$1 \cdot 10^{-2}$	22
le450_15c	15	$2 \cdot 10^{-3}$	30	$2 \cdot 10^{-3}$	**27**	$3 \cdot 10^{-3}$	**27**	$3 \cdot 10^{-2}$	30
le450_15d	15	$1 \cdot 10^{-3}$	31	$2 \cdot 10^{-3}$	27	$3 \cdot 10^{-3}$	**26**	$3 \cdot 10^{-2}$	31
le450_25a	25	$7 \cdot 10^{-4}$	28	$2 \cdot 10^{-3}$	26	$2 \cdot 10^{-3}$	**25**	$2 \cdot 10^{-2}$	28
le450_25b	25	$7 \cdot 10^{-4}$	27	$1 \cdot 10^{-3}$	26	$2 \cdot 10^{-3}$	**25**	$1 \cdot 10^{-2}$	27
le450_25c	25	$1 \cdot 10^{-3}$	37	$3 \cdot 10^{-3}$	**31**	$3 \cdot 10^{-3}$	**31**	$3 \cdot 10^{-2}$	37
le450_25d	25	$1 \cdot 10^{-3}$	35	$3 \cdot 10^{-3}$	30	$3 \cdot 10^{-3}$	**29**	$3 \cdot 10^{-2}$	35
miles250	8	$8 \cdot 10^{-5}$	9	$7 \cdot 10^{-5}$	**8***	$2 \cdot 10^{-4}$	**8**	$9 \cdot 10^{-4}$	9
miles500	20	$2 \cdot 10^{-4}$	22	$2 \cdot 10^{-4}$	**20***	$3 \cdot 10^{-4}$	**20**	$2 \cdot 10^{-3}$	22
miles750	31	$5 \cdot 10^{-4}$	34	$5 \cdot 10^{-4}$	32	$5 \cdot 10^{-4}$	**31**	$5 \cdot 10^{-3}$	34
miles1000	42	$5 \cdot 10^{-4}$	44	$9 \cdot 10^{-4}$	**43**	$6 \cdot 10^{-4}$	**43**	$8 \cdot 10^{-3}$	44
miles1500	73	$8 \cdot 10^{-4}$	76	$2 \cdot 10^{-3}$	**73**	$1 \cdot 10^{-3}$	**73***	$2 \cdot 10^{-2}$	76
myciel5	6	$3 \cdot 10^{-5}$	**6***	$3 \cdot 10^{-5}$	**6***	$5 \cdot 10^{-5}$	7	$2 \cdot 10^{-4}$	**6**

A Tabela 5 apresenta os resultados dos testes realizados com os três algoritmos exatos de coloração (Zykov, Lawler e PLI). Para garantir que o processamento ocorresse dentro de um intervalo de tempo satisfatório, foi necessário limitar a quantidade de vértices devido ao aumento exponencial do tempo de execução. Os casos de teste também foram gerados de forma aleatória, para instâncias com 5, 10 e 15 vértices, e probabilidade p variando de 20% a 80%.

O Algoritmo de Lawler apresentou o menor tempo de execução em grande parte das instâncias, seguido pelo Algoritmo de Zykov e o Algoritmo de PLI, nessa ordem. Os resultados obtidos reforçam o motivo pelo qual a formulação utilizada no Algoritmo de PLI não é viável em termos práticos, dado que o problema de simetria gerado nessa formulação impacta diretamente no tempo de execução do algoritmo.

Nota-se também que, conforme o valor de p aumenta, o desempenho do Algoritmo de Zykov melhora em relação aos demais algoritmos. Isso acontece pois quanto maior a densidade do grafo, menor é a quantidade de operações de contração de vértices e

Tabela 5: Desempenho dos algoritmos exatos em grafos aleatórios.

n	p	Zykov		Lawler		PLI	
		T(s)	k	T(s)	k	T(s)	k
5	20%	$4 \cdot 10^{-4}$	2	$\mathbf{7 \cdot 10^{-5}}$	2	$2 \cdot 10^{-3}$	2
	40%	$7 \cdot 10^{-5}$	3	$\mathbf{3 \cdot 10^{-5}}$	3	$3 \cdot 10^{-3}$	3
	60%	$6 \cdot 10^{-5}$	3	$\mathbf{2 \cdot 10^{-5}}$	3	$4 \cdot 10^{-3}$	3
	80%	$3 \cdot 10^{-5}$	4	$\mathbf{2 \cdot 10^{-5}}$	4	$1 \cdot 10^{-2}$	4
10	20%	$2 \cdot 10^{-1}$	3	$\mathbf{5 \cdot 10^{-3}}$	3	$4 \cdot 10^{-2}$	3
	40%	$4 \cdot 10^{-2}$	4	$\mathbf{5 \cdot 10^{-3}}$	4	1	4
	60%	$8 \cdot 10^{-3}$	5	$\mathbf{4 \cdot 10^{-3}}$	5	8	5
	80%	$\mathbf{2 \cdot 10^{-3}}$	6	$4 \cdot 10^{-3}$	6	$3 \cdot 10^{1}$	6
15	20%	$2 \cdot 10^{4}$	3	1	3	$\mathbf{4 \cdot 10^{-1}}$	3
	40%	$1 \cdot 10^{2}$	4	**1**	4	$1 \cdot 10^{1}$	4
	60%	7	5	**1**	5	$3 \cdot 10^{2}$	5
	80%	$\mathbf{1 \cdot 10^{-2}}$	6	$1 \cdot 10^{-1}$	6	$1 \cdot 10^{3}$	6

adição de arestas necessárias para transformá-lo em um grafo completo, o que limita o tamanho da árvore de Zykov.

O Algoritmo de Lawler foi o que apresentou menor variação de tempo de execução para variações de densidade p em instâncias com uma mesma quantidade de n vértices. Esse comportamento pode ser explicado pela característica da programação dinâmica do algoritmo, que exige o cálculo dos 2^n subconjuntos de vértices para definição do número cromático.

Por fim, a estratégia de cada algoritmo impacta diretamente seu desempenho para diferentes instâncias, dependendo dos valores de n e p. A escolha do melhor algoritmo varia conforme a aplicação: o Algoritmo de Zykov pode ser mais eficiente em grafos densos, o de Algoritmo de Lawler pode ser mais adequado para grafos com um grande número de vértices, e o Algoritmo de PLI pode ter sua formulação modificada para ser eficaz em aplicações reais.

6 Conclusão

A coloração de vértices de um grafo é um problema clássico na Teoria dos Grafos, com aplicações em diversas áreas, como alocação de frequências, escalonamento de tarefas e otimização de recursos. Ao longo deste trabalho, foram apresentados conceitos fundamentais e abordagens computacionais para resolver o problema, desde heurísticas rápidas e eficientes até algoritmos exatos que garantem a solução ótima. A análise dessas abordagens permitiu compreender os desafios envolvidos e as estratégias disponíveis para minimizar o número de cores utilizadas.

Foram analisadas quatro heurísticas: o algoritmo Sequencial, Welsh-Powell, DSATUR e RLF. Cada um desses métodos possui características distintas, sendo que alguns priorizam simplicidade e velocidade, enquanto outros buscam uma melhor distribuição das cores com base em critérios específicos, como o grau de saturação. Apesar de não garantirem a solução ótima, essas heurísticas apresentam bom desempenho prático, sendo adequadas para grafos de grande porte.

Além disso, foram estudados três algoritmos exatos: Zykov, Lawler e Programação Linear Inteira (PLI). Esses métodos garantem a obtenção do número cromático mínimo, mas apresentam um custo computacional elevado, principalmente em grafos com uma grande quantidade de vértices. Esse desafio foi bastante perceptível na execução dos testes, onde foi necessário reduzir o tamanho das instâncias para viabilizar o processamento dos algoritmos.

A implementação dos algoritmos foi uma das etapas mais desafiadoras do trabalho. O desenvolvimento das heurísticas permitiu uma maior familiarização com a linguagem de programação C++, além de exigir a criação de funções auxiliares para medir o tempo de execução das soluções em diferentes instâncias. Esse processo reforçou a importância de otimizações no código e da escolha eficiente de estruturas de dados para lidar tanto com os grafos gerados de forma aleatória quanto com as instâncias extraídas do DIMACS.

Por fim, a comparação entre heurísticas e algoritmos exatos evidenciou a relação entre qualidade da solução e tempo de processamento. Enquanto as heurísticas são capazes de fornecer respostas rápidas e razoavelmente boas, os algoritmos exatos se mostram indispensáveis quando a solução ótima é necessária. Dessa forma, a escolha do método mais adequado depende do contexto e das restrições do problema analisado, equilibrando precisão e viabilidade computacional.

Referências

[1] Appel, A.W.: Modern compiler implementation in C. Cambridge university press (2004)

[2] Bellman, R.: Dynamic programming. Science **153**(3731), 34–37 (1966)

[3] Bodlaender, H.L., Kratsch, D.: An exact algorithm for graph coloring with polynomial memory. UU-CS **2006** (2006)

[4] Bondy, J.A., Murty, U.S.R.: Graph theory. Springer Publishing Company, Incorporated (2008)

[5] Brown, J.R.: Chromatic scheduling and the chromatic number problem. Management science **19**(4-part-1), 456–463 (1972)

[6] Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM **22**(4), 251–256 (1979)

[7] Byskov, J.M.: Chromatic Number in Time O (2.4023↑n): Using Maximal Independent Sets. BRICS (2002)

[8] Dantzig, G.B.: Linear programming and extensions. In: Linear programming and extensions. Princeton university press (2016)

[9] DIMACS: Dimacs implementation challenges (2025), https://www.dimacs.rutgers.edu/, acessado em: 14 abr. 2025

[10] Eppstein, D.: Small maximal independent sets and faster exact graph coloring. arXiv preprint cs/0011009 (2000)

[11] Feofiloff, P., Kohayakawa, Y., Wakabayashi, Y.: Uma introduçao sucinta à teoria dos grafos (2011)

[12] Fritsch, R., Fritsch, R., Fritsch, G., Fritsch, G.: Four-Color Theorem. Springer (1998)

[13] Garey, M.R., Johnson, D.S.: Computers and intractability, vol. 174. freeman San Francisco (1979)

[14] Goldbarg, M.C., Luna, H.P.L.: Otimização combinatória e programação linear. Editora CAMPUS, Rio de Janeiro (2000)

[15] Guthrie, F.: 9. note on the colouring of maps. Proceedings of the Royal Society of Edinburgh **10**, 727–728 (1880)

[16] Held, S., Cook, W., Sewell, E.C.: Safe lower bounds for graph coloring. In: Integer Programming and Combinatoral Optimization. pp. 261–273. Springer (2011)

[17] Lawler, E.L.: A note on the complexity of the chromatic number problem. Information Processing Letters **5**(3), 66–67 (1976)

[18] Leighton, F.T.: A graph coloring algorithm for large scheduling problems. Journal of research of the national bureau of standards **84**(6), 489 (1979)

[19] Mehrotra, A., Trick, M.A.: A column generation approach for graph coloring. informs Journal on Computing **8**(4), 344–354 (1996)

[20] Moon, J.W., Moser, L.: On cliques in graphs. Israel journal of Mathematics **3**, 23–28 (1965)

[21] San Segundo, P.: A new dsatur-based algorithm for exact vertex coloring. Computers & Operations Research **39**(7), 1724–1733 (2012)

[22] Wang, C.C.: An algorithm for the chromatic number of a graph. Journal of the ACM (JACM) **21**(3), 385–391 (1974)

[23] Welsh, D.J., Powell, M.B.: An upper bound for the chromatic number of a graph and its application to timetabling problems. The Computer Journal **10**(1), 85–86 (1967)

[24] Zykov, A.A.: On some properties of linear complexes. Matematicheskii sbornik **66**(2), 163–188 (1949)