Universidade Federal do ABC
Centro de Matemática, Computação e Cognição
Relatório parcial de pesquisa – Iniciação Científica
Fundação de Amparo à Pesquisa do Estado de São Paulo
Processo 2020/09330-7

# Algoritmos para Coloração de Grafos

*Aluno:*
Wesley Lima de Araujo

*Supervisor:*
Prof. Dra. Carla Negri Lintzmayer

Abril/2020

**Resumo**

Problemas de coloração de grafos são muito importantes pois modelam situações em que há conflitos entre objetos e deseja-se separá-los em grupos de acordo com algum critério, como é o caso em problemas de alocação, por exemplo. Esse relatório parcial, destinado à FAPESP, é critério de avaliação do projeto de iniciação científica 2020/09330-7 e visa informar à FAPESP sobre o estado do projeto. O principal objetivo desse projeto é expandir os conhecimentos do aluno sobre projeto de algoritmos e otimização combinatória através do estudo de diversos tópicos relacionados à coloração de grafos, especialmente algoritmos que visam resolver o problema da coloração mínima.

**Assuntos:** Algoritmos; Otimização Combinatória; Grafos; Análise de Algoritmos.

# Sumário

# 1 Introdução

Ao longo desse relatório usaremos amplamente definições e conceitos de teoria dos grafos. Para qualquer conceito que não for aqui definido, estará sendo considerada a definição apresentada no livro *"Graph Theory"* de Bondy e Murty [1], e o mesmo vale para notações. Também vale ressaltar que quando usamos o termo grafo estaremos nos referindo a grafos simples.

O **problema de coloração de vértices** (também chamado de coloração de grafos), de maneira intuitiva, visa rotular os vértices de um grafo com cores, de maneira que não haja conflito entre vértices vizinhos. Um conflito ocorre quando dois vértices vizinhos possuem a mesma cor. Dessa forma, cada aresta representa a possibilidade de um conflito. Esse é um problema aplicável em diversas situações do mundo real onde desejamos modelar situações com conflitos, e para tal basta que representemos os objetos modelados como vértices e os conflitos entre eles como arestas. Problemas de alocação são um bom exemplo de problema que pode ser modelado e resolvido por coloração de vértices.

Por exemplo, imagine um problema de construção de calendário em que temos algum conjunto de atividades (palestras, aulas, jogos, etc) e um conjunto de horários em que essas atividades podem ocorrer. Desejamos alocar as atividades nesses horários mas somos limitados por um conjunto de restrições (pessoas, salas, etc). Veja que nesse caso podemos representar cada atividade como um vértice e cada restrição como uma aresta, sendo que nossa busca por marcar um horário compatível se resume a colorir o grafo que construímos, onde cada cor é um horário.

Dado um grafo $G$, chamamos de **coloração própria** de $G$ uma coloração dos vértices em $V(G)$ que respeite as restrições impostas, ou seja, uma coloração é própria se dois vértices vizinhos em $G$ nunca têm a mesma cor. Normalmente, quando falamos de uma coloração qualquer estará implícito que estamos falando de uma coloração própria. Por isso, ao longo desse trabalho, a não ser que seja especificado o contrário, quando usarmos o termo coloração estaremos nos referindo a uma coloração própria. Se um grafo $G$ aceita uma coloração que usa $k$ cores, falamos que trata-se de uma **$k$-coloração**, além disso, falamos que $G$ é um grafo **$k$-colorível** ou **$k$-colorável**.

Muitas vezes, não nos interessa apenas encontrar uma resolução qualquer do conflito proposto, mas também encontrar uma solução com o menor número de cores possível. Então, no nosso exemplo de calendário seria como se, além de querer construir o calendário, quiséssemos usar o menor número de horários possíveis. É evidente que caso quiséssemos usar o maior número de cores possível bastaria associar à cada atividade (vértice) um horário (cor).

Assim sendo, o problema de coloração de vértices passa a ser um problema de otimização

combinatória quando desejamos não só encontrar uma coloração, mas sim uma coloração com o menor número possível de cores. Chamamos essa quantidade mínima de cores possível para uma coloração de um grafo $G$ de **número cromático do grafo** $G$, denotado por $\chi(G)$. Chamamos uma coloração de um grafo $G$ que use $\chi(G)$ cores de **coloração ótima de** $G$. Além disso, se para um grafo $G$ temos que $\chi(G) = k$, falamos que $G$ é um **grafo $k$-cromático**.

Segue a definição formal do problema de Coloração Mínima de Vértices, em que buscamos encontrar uma coloração ótima para um grafo.

**Problema 1.1** (Coloração Mínima de Vértices)**.** *Dado um grafo $G$, sendo que $n = |V(G)|$, no problema da Coloração Mínima de Vértices (CMV), desejamos associar a cada vértice $v \in V(G)$ um inteiro $c(v) \in \{1, \ldots, k\}$ de forma que:*

- *$c(v) \neq c(u)$, caso $\{v, u\} \in E(G)$;*
- *$k$ é mínimo.*

*Nessa definição do problema cada valor inteiro em $\{1, \ldots, k\}$ é chamado de **cor**.*

A definição para o CMV que acabamos de dar não é a única possível. Ela foi retirada do livro *"A Guide to Graph Colouring"* de Lewis [5], que também dá outras definições, inclusive uma que usa o conceito de conjuntos independentes. Dizemos que um subconjunto $S$ dos vértices de um grafo $G$ é um **conjunto independente** se quaisquer dois vértices $u, v \in S$ não são vizinhos. Portanto, toda coloração viável (coloração que respeite os conflitos) nada mais é que uma coleção de conjuntos independentes de $G$ onde os vértices do grafo pertencem a um, e somente um, conjunto da coleção. Segue essa definição alternativa para o CMV.

**Problema 1.2** (Coloração Mínima de Vértices)**.** *Dado um grafo $G$, desejamos encontrar uma coleção $\mathcal{S}$ de $k$ conjuntos independentes $S_i$ em $G$ tal que*

$$\bigcup_{i=1}^{k} S_i = V(G) \tag{1}$$

$$S_i \cap S_j = \emptyset, \quad 1 \leq i \neq j \leq k \tag{2}$$

$$\forall u, v \in S_i, \{u, v\} \notin E(G), \quad 1 \leq i \leq k \tag{3}$$

*sendo que $k$ deve ser mínimo.*

*Nessa definição do problema, cada conjunto independente $S_i$ é chamado de **cor**.*

Observe que a definição alternativa define exatamente o mesmo problema, sendo a grande diferença a forma como o problema é formalmente representado.

A partir daqui, sempre que falarmos "problema da coloração de grafos" ou "problema da coloração de vértices", estaremos nos referindo ao CMV.

Para resolvermos o problema da coloração de grafos podemos usar algoritmos que recebem uma instância do problema (grafo $G$ qualquer) e devolvem uma solução para essa instância (coloração ótima em $G$). Infelizmente, o problema da coloração de grafos possivelmente não pode ser resolvido em tempo polinomial, afinal, o problema de decisão que decide se um número natural $k$ é número cromático para um grafo $G$ é $\mathcal{NP}$-completo [4], e, por consequência, encontrar uma coloração ótima de um grafo $G$ é $\mathcal{NP}$-difícil. Dada essa realidade, não temos esperança em desenvolver um algoritmo para coloração de vértices que:

1. seja computacionalmente eficiente; e
2. devolva uma solução ótima para qualquer instância de entrada.

Porém, isso não impede os projetistas de algoritmos de desenvolverem algoritmos para o problema, a questão é que é necessário sacrificar ou tempo de execução eficiente ou otimalidade da solução para toda instância. Baseando-se nessa ideia de sacrifício, existem técnicas de projeto de algoritmos que seguem três abordagens distintas com relação aos dois itens citados acima:

- **Abordagem exata:** Nessa abordagem sempre garantimos que a solução devolvida pelo nosso algoritmo e ótima, porém, para problemas pertencentes a $\mathcal{NP}$-difícil esse tipo de algoritmo é ineficiente para instâncias que sejam muito grandes;
- **Abordagem heurística:** Um algoritmo é considerado heurístico se ele possuir tempo de execução polinomial para toda instância e devolve uma solução viável;
- **Abordagem por aproximação:** Dizemos que um determinado algoritmo é de aproximação se para qualquer instância o tempo de execução é polinomial e existe certa garantia sobre a qualidade da solução, mas nem sempre a solução devolvida é ótima. Especificamente, um algoritmo é uma $\alpha$-aproximação se para qualquer instância recebida ele devolve uma solução cujo custo está a um fator de no máximo $\alpha$ do custo da solução ótima.

O objetivo desse projeto é ampliar a experiência do candidato com pesquisa científica na área de Ciência da Computação, bem como a complementar a sua formação, ampliando seu conhecimento na área de otimização combinatória e, principalmente, no projeto de algoritmos por meio do estudo de algoritmos e problemas relacionados a coloração de grafos.

Nesse relatório parcial pretendemos expor os assuntos que já foram estudados pelo aluno de maneira resumida, para que fique bem claro quais tópicos foram vistos sem que haja

necessidade do avaliador entrar em todos os pormenores dos assuntos estudados. Lembramos que a metodologia adotada para realização do projeto consiste no aluno estudar o assunto em questão entendendo as demonstrações e conceitos envolvidos e, em reuniões de frequência semanal, discutir com a orientadora sobre o assunto e sanar dúvidas que possam ter surgido.

O relatório está dividido em diversas seções. Além dessa introdução teremos seções dedicadas a assuntos específicos. A Seção 2 introduz os problemas de coloração de aresta e coloração total. A Seção 3 apresenta alguns limitantes para o número cromático de um grafo. A Seção 4 fala sobre a complexidade do problema da coloração de vértices. A Seção 5 apresenta alguns algoritmos para o problema da coloração de vértices baseados em heurísticas. A Seção 6 apresenta alguns algoritmos exatos para o problema da coloração de vértices. Já a Seção 7 é dedicada a analisar o desenvolvimento do projeto durante o período que esse relatório abrange (de 01/11/2020 até 10/04/2021) e apresentar as expectativas para os próximos meses do projeto.

Além das seções que compõem o corpo do relatório também existem três apêndices, todos eles terão conteúdos complementares ao que foi escrito no corpo principal do trabalho mas que fogem do escopo do que seria apresentado num relatório de pesquisa. O Apêndice A apresentará alguns conceitos e notações em teoria dos grafos que foram utilizados ao longo do relatório. O Apêndice B apresentará algumas demonstrações de teoremas selecionados apresentados no relatório. O Apêndice C apresentará alguns dos algoritmos citados no relatório.

# 2 Da Coloração Total à Coloração de Vértices

Nessa seção apresentamos alguns problemas de coloração associados ao problema da coloração de Grafos. Mais especificamente, apresentaremos o Problema de Coloração de Arestas, o Problema de Coloração Total de grafos e alguns resultados associados. Essa seção tem como principal referência a tese de Campos [2], que trata sobre o problema da Coloração Total de Vértices para classes específicas de grafos.

No problema da **Coloração de Arestas**, que nada mais é que uma rotulação das arestas de um grafo de forma que duas arestas que incidam em um mesmo vértice não possuam o mesmo rótulo, no caso a mesma cor. Segue a definição formal do problema da Coloração Mínima de Arestas.

**Problema 2.1** (Coloração Mínima de Arestas)**.** *Dado um grafo $G$, sendo que $m = |E(G)|$, desejamos associar a cada aresta $e \in E(G)$ um inteiro $c(e) \in \{1, \ldots, k\}$ de forma que:*

- *Dadas $e_i, e_j \in E(G)$ onde $e_i = \{v, u\}$, $e_j = \{v, w\}$ e $e_i \neq e_j$, vale que $c(e_i) \neq c(e_j)$;*

- *$k$ é mínimo.*

Na coloração de vértices denotamos o número cromático de um grafo $G$ por $\chi(G)$ e o definimos como a menor quantidade de cores possível que uma coloração de vértices pode assumir para $G$. Analogamente, para a coloração de arestas temos o **índice cromático do grafo**, representado por $\chi'(G)$, que é a menor quantidade de cores que qualquer coloração de arestas pode ter.

Podemos dizer que o principal resultado relacionado ao problema da Coloração Mínima de Arestas (a partir de agora chamaremos só de coloração de arestas) é o Teorema de Vizing [1], enunciado a seguir.

**Teorema 2.1** (Teorema de Vizing)**.** *Dado grafo um $G$ e seja $\Delta(G)$ o grau máximo de $G$, então vale que:*

- $\chi'(G) = \Delta(G)$, *ou*
- $\chi'(G) = \Delta(G) + 1$.

Baseando-se no Teorema de Vizing os grafos são divididos em duas classes. A classe 1 é a classe dos grafos $G$ em que $\chi'(G) = \Delta(G)$. Já a classe 2 de grafos é a classe de grafos $G$ em que $\chi'(G) = \Delta(G) + 1$.

Da mesma forma que falamos de colorações só de arestas ou só vértices, também podemos falar de colorações que visem colorir ambos, arestas e vértices. Uma **Coloração Total** de um grafo $G$ é uma rotulação de cores às arestas e aos vértices de $G$, de maneira que cada par de vértices adjacentes e cada par de arestas adjacentes tenham sempre cores distintas e, além disso, cada vértice deve ter cor distinta à das arestas que nele incidem. Formalmente, dados um grafo $G$ e um valor $k \in \mathbb{N}$ chamamos de coloração total de $G$ uma função $\Phi : V(G) \cup E(G) \to \{1, \ldots, k\}$ tal que:

- $\forall v \in V(G)$ e $\forall u \in N_G(v)$ vale que $\Phi(v) \neq \Phi(u)$;
- $\forall v \in V(G)$ e $\forall \{v, x\} \in E(G)$ vale que $\Phi(v) \neq \Phi(\{v, x\})$;
- $\forall \{v, x\}, \{v, y\} \in E(G)$, onde $x \neq y$, vale que $\Phi(\{v, x\}) \neq \Phi(\{v, y\})$.

Chamamos de **número cromático total** do grafo $G$, o menor valor $k$ para o qual $G$ assume uma coloração total. Denotamos o número cromático total como $\chi_T(G)$.

Qualquer coloração total de um grafo $G$ precisa de ao menos $\Delta(G)+1$ cores, pois o vértice de grau máximo tem $\Delta(G)$ arestas vizinhas e, além disso, também precisamos de mais uma cor para o próprio vértice. Esse fato estabelece um limitante inferior para $\chi_T(G)$.

Behzad e Vizing [2] apresentaram a **Conjectura da Coloração Total** que afirma que para todo grafo $G$ vale que $\chi_T(G) \leq \Delta(G)+2$. Então, em conjunto com o que discutimos no último parágrafo, essa conjectura dá base para um resultado na coloração total semelhante ao Teorema de Vizing na coloração de arestas, que seria: dado grafo $G$, vale que $\chi_T(G) = \Delta(G)+1$ ou $\chi_T(G) = \Delta(G)+2$. É importante destacar que a Conjectura da Coloração Total já foi provada para diversas classes de grafos [2].

Para encerrar a seção, na Figura 2 há um exemplo onde temos (da esquerda para a direita) uma coloração de vértices, uma coloração de arestas e uma coloração total de um mesmo grafo.

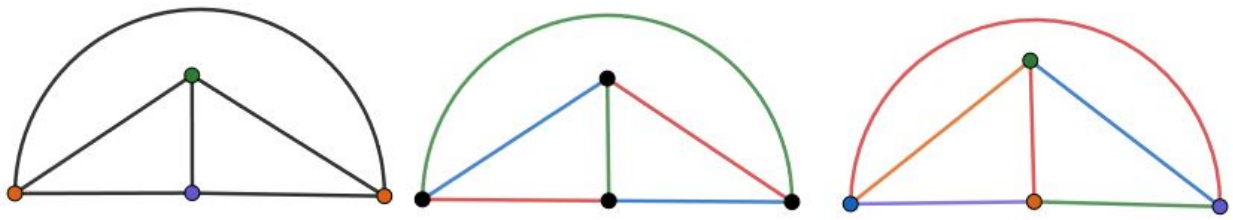

Figura 1: Coloração de vértices, coloração de arestas e coloração total de um grafo completo com 4 vértices

# 3 Limitantes do Número Cromático

É fácil perceber que $|V(G)|$ é limitante superior para o valor de $\chi(G)$, afinal, no máximo podemos ter uma cor para cada vértice de um grafo $G$. Nessa seção apresentaremos alguns resultados clássicos referentes à coloração de vértices e que fornecem limitantes para o valor de $\chi(G)$. A principal referência dessa seção foi o livro *"Graph Theory"* de Bondy e Murty [1]. Esses resultados estão listados na Proposição 3.1.

**Proposição 3.1.** *Dado um grafo $G$, onde $n = |V(G)|$, vale que:*

- $\chi(G) = 1 \Leftrightarrow G$ *é um grafo nulo;*
- $\chi(G) = 2 \Leftrightarrow G$ *é um grafo bipartido;*
- *Se $G$ é um ciclo de tamanho par, então $\chi(G) = 2$ (bipartido);*
- *Se $G$ é um ciclo de tamanho ímpar, então $\chi(G) = 3$;*
- *Se $G$ é um grafo completo, então $\chi(G) = n$;*

- $\chi(G) \geq \omega(G)$, *onde* $\omega(G)$ *é a cardinalidade da maior clique em* $G$*; e*
- $\chi(G) \geq \frac{n}{\alpha(G)}$, *onde* $\alpha(G)$ *é tamanho do maior conjunto independente em* $G$*.*

Os resultados acima são interessantes, mas podem ser considerados intuitivos. Em contrapartida, existem teoremas que estabelecem outros três limitantes para o valor de $\chi(G)$. O primeiro está associado à quantidade de arestas do grafo, o segundo está relacionado ao conceito de grafo k-crítico, e o terceiro é conhecido como Teorema de Brooks.

**Teorema 3.1.** *Dado um grafo* $G$*, vale que* $\chi(G) \leq \frac{1}{2} + \sqrt{2|E(G)| + \frac{1}{4}}$.

Para o segundo limitante que iremos apresentar, vamos primeiro definir os conceitos de **grafo crítico** e **grafo** $k$**-crítico**, que serão usados na demonstração do Teorema 3.2, que por sua vez implicará em dois corolários, sendo que é um desse corolários que realmente apresentará o limitante para $\chi(G)$.

**Definição 3.1** (Grafo Crítico e Grafo $k$-crítico)**.** *Dado um grafo* $G$*, falamos que* $G$ *é um* ***grafo crítico*** *se todo subgrafo* $H$ *próprio de* $G$ *respeita que* $\chi(H) < \chi(G)$*. Além disso, sendo* $k = \chi(G)$*, dizemos que* $G$ *é um* ***grafo k-crítico*** *se ele é crítico e* $k$*-cromático.*

Um resultado interessante que usaremos, e que relaciona grafos $k$-cromáticos com grafos $k$-críticos, diz que todo grafo $G$ $k$-cromático possui um subgrafo $H$ que é $k$-crítico [1].

**Teorema 3.2.** *Dado um grafo* $G$ $k$*-crítico, é verdade que* $\delta(G) \geq k - 1$.

**Corolário 3.1.** *Todo grafo* $k$*-cromático possui ao menos* $k$ *vértices com grau maior ou igual a* $k - 1$.

**Corolário 3.2.** *Para todo grafo* $G$ *vale que* $\chi(G) \leq \Delta(G) + 1$.

Por mais que o Corolário 3.2 nos dê um limitante superior para o valor de $\chi(G)$, esse limitante ainda pode ser muito distante do valor real de $\chi(G)$, sendo o exemplo mais fácil de visualizar isso um grafo bipartido completo. O Teorema de Brooks prova um limitante um pouco mais justo, porém, que ainda pode estar muito longe do valor real de $\chi(G)$.

**Teorema 3.3** (Teorema de Brooks)**.** *Se* $G$ *é conexo e não é um ciclo ímpar ou um grafo completo, vale que* $\chi(G) \leq \Delta(G)$.

# 4 Complexidade do Problema

Conforme já dizemos na Seção 1, o problema de decisão da coloração de vértices é $\mathcal{NP}$-completo, enquanto o problema de otimização da coloração de vértices é $\mathcal{NP}$-difícil. Nessa

seção, pretendemos apresentar a demonstração da $\mathcal{NP}$-completude do problema de decisão de coloração de vértices para 3 cores, problema conhecido como **3-coloração** (Problema 4.1). Esse resultado implica na $\mathcal{NP}$-completude do problema de decisão da coloração de vértices para qualquer $k$, problema conhecido como **k-coloração** (Problema 4.2). As principais referências dessa seção são Karp [4] e Cormen e Leiserson [3], inclusive, quaisquer termos usados nessa seção que não forem definidos aqui podem ter sua definição encontrada nesses trabalhos.

**Problema 4.1** (3-Coloração)**.** *Problema de decisão onde, dado um grafo $G$, devolvemos:*

- Sim*, caso exista 3-coloração própria para $G$;*
- Não*, caso não exista 3-coloração própria para $G$.*

**Problema 4.2** (k-Coloração)**.** *Problema de decisão onde, dado um grafo $G$ e um número natural $k$, devolvemos:*

- Sim*, caso exista k-coloração própria para $G$;*
- Não*, caso não exista k-coloração própria para $G$.*

Para demonstrar que um problema é $\mathcal{NP}$-completo é necessário realizar uma redução de um outro problema de decisão que seja $\mathcal{NP}$-completo para ele. Então, para demonstrar que o 3-coloração é $\mathcal{NP}$-completo temos que realizar uma redução de algum outro problema $\mathcal{NP}$-completo para o 3-coloração. Isso significa tomar uma instância desse outro problema, adaptar ela para uma instância do 3-coloração, provar que a resposta do 3-coloração é sim para essa instância adaptada se e somente se a resposta do problema para a instância original também for sim. A ideia por trás disso é provar que a dificuldade em resolver ambos os problemas é igual.

**Definição 4.1** (Redução)**.** *Sejam $P_1$ e $P_2$ dois problemas de decisão. Se existe uma forma de adaptar uma instância qualquer $\alpha$ de $P_1$ para uma instância $\beta$ de $P_2$, de forma que a resposta de $P_2$ para instância $\beta$ é "sim" se e somente se a resposta de $P_1$ para a instância $\alpha$ é "sim", então chamamos o processo de adaptação de instância em conjunto com a resolução de $P_1$ para a instância $\beta$ de* ***redução****.*

O primeiro problema provado $\mathcal{NP}$-completo foi o **Satisfatibilidade Booliana** (Problema 4.3), conhecido como SAT. Aqueles que descobriram esse resultado foram Stephen Cook e Leonid Levin, fazendo com que essa descoberta fosse conhecida como Teorema de Cook-Levin (Teorema 4.1).

**Problema 4.3** (Satisfatibilidade Booliana)**.** *Problema de decisão onde, dada uma expressão booleana $\phi$ em sua forma normal conjuntiva, devolvemos:*

- SIM*, caso exista valoração dos literais de $\phi$ que faça seu valor ser verdade;*
- NÃO*, caso não exista valoração dos literais de $\phi$ que faça seu valor ser verdade.*

*Caso a resposta seja* SIM*, dizemos que $\phi$ é* ***satisfazível****, caso contrário dizemos que $\phi$* ***não é satisfazível****.*

**Teorema 4.1** (Teorema de Cook-Levin)**.** *O problema da Satisfatibilidade Booliana é $\mathcal{NP}$-completo, além disso, qualquer outro problema $\mathcal{NP}$-completo pode ser reduzido em tempo polinomial para a Satisfatibilidade Booliana.*

Um outro problema $\mathcal{NP}$-completo, que é relacionado ao SAT, é o **3-Satisfatibilidade Booliana** (Problema 4.4), conhecido como 3-SAT. O 3-SAT é conhecidamente $\mathcal{NP}$-completo, sendo que esse resultado foi obtido por Karp [4] através de uma redução entre SAT e 3-SAT. A grande diferença entre o SAT e o 3-SAT está relacionada à quantidade de literais em cada cláusula da forma normal conjuntiva de uma expressão booliana que seja instância do problema.

**Problema 4.4** (3-Satisfatibilidade Booliana)**.** *Problema de decisão onde, dada uma expressão booleana $\phi$ em sua forma normal conjuntiva onde cada cláusula possui exatamente 3 literais, devolvemos:*

- SIM*, caso exista valoração dos literais de $\phi$ que faça seu valor ser verdade;*
- NÃO*, caso não exista valoração dos literais de $\phi$ que faça seu valor ser verdade.*

Para provar que o 3-coloração é $\mathcal{NP}$-completo usa-se uma redução do 3-SAT para a 3-coloração, isso significa que vamos pegar uma instância do 3-SAT ($\phi$) e transformá-la numa instância do 3-coloração ($G$), e então provar que $G$ é 3-colorável se e somente se $\phi$ é satisfazível. Esse resultado é demonstrado com detalhes no Teorema

**Teorema 4.2.** *3-coloração é $\mathcal{NP}$-completo.*

Voltando a algo que foi dito no início dessa seção, é fácil perceber que qualquer instância do 3-coloração pode ser reduzida para a k-coloração, sendo que a resposta devolvida pela k-coloração é sempre a mesma que a da 3-coloração, dessa forma k-coloração também é $\mathcal{NP}$-completo. O fato da k-coloração ser um problema $\mathcal{NP}$-completo implica diretamente que o problema da coloração mínima de vértices seja $\mathcal{NP}$-difícil.

# 5 Algoritmos Básicos

Como discutimos na seção anterior, o problema da coloração de vértices é $\mathcal{NP}$-difícil, isso faz com que ele seja considerado, até hoje, intratável. Para tratar problemas com essa característica uma alternativa viável é o uso de heurísticas, ou seja, algoritmos com tempo de execução polinomial mas sem garantias sobre a qualidade da solução. Nessa seção apresentaremos três algoritmos heurísticos considerados básicos devido à sua ideia de funcionamento ser simples, também vale destacar que os três são algoritmos gulosos. A principal referência dessa seção foi o livro *A Guide to Graph Colouring* [5], nesse livro os 3 algoritmos são chamados de algoritmos construtivos. Ao longo da seção vamos ver que, por mais que não existam garantias sobre a qualidade da solução devolvida por esses algoritmos para **todos** os grafos, existem classes de grafos quais podemos usar esses algoritmos para encontrar a solução ótima.

## 5.1 Algoritmo Guloso

Vamos começar com aquela que, provavelmente, é a heurística mais simples e fundamental para o problema da coloração de vértices, a heurística gulosa. O algoritmo funciona da seguinte forma, dada uma ordem, pré-definida, dos vértices do grafo instância o algoritmo passa por cada vértice seguindo essa ordem, e então ele atribui a $1^{\underline{a}}$ cor disponível para esse vértice. Aqui, o algoritmo guloso está representado no Algoritmo 2.

O algoritmo guloso produz soluções viáveis bem rapidamente, e dependendo da ordem em que os vértices são passados o algoritmo guloso pode produzir uma solução ótima. Por outro lado, a solução encontrada poder ser muito ruim. Tome como exemplo um grafo bipartido, se a ordem de vértices dada ao algoritmo tem todos os vértices de uma das duas partições em sequência a quantidade de cores usada será muito maior do que o real valor de $\chi(G)$. Isso só ressalta que ordem de vértices é muito importante. Existem algoritmos próprios para tentar encontrar ordens de vértices que levariam a uma solução ótima.

Vamos agora fazer uma análise do tempo de execução do algoritmo Guloso (Algoritmo 2), considere que $n = |V(G)|$ e $m = |E(G)|$. Veja que o primeiro laço **para** do algoritmo executa um número $O(n)$ de vezes, pois temos até uma iteração para cada vértice da sequência. Já no segundo laço **para** temos que verificar se um conjunto é independente, uma operação $O(nm)$, além disso, o próprio laço é $O(n)$, pois podemos ter uma cor $c_i$ para cada vértice. Dessa forma, o tempo do algoritmo é dado por $O(n(n^2m)) = O(n^3m) = O(n^3)$.

Dada a simplicidade e rapidez do algoritmo, uma estratégia comum ao usá-lo envolve executá-lo repetidas vezes, porém, a ordem $\pi$ dos vértices da segunda execução em diante depende da coloração atribuída na execução anterior. A estratégia consiste em, na ordem $\pi$ da próxima execução, colocar os vértices que foram coloridos com a mesma cor em sequência.

Inclusive, existem teoremas que dão certas garantias sobre essa estratégia, aqui vamos apresentar dois desses teoremas, o Teorema 5.1 e o Teorema 5.2.

**Teorema 5.1.** *Seja $S$ uma coloração viável do grafo $G$ e seja $1 \leq i \leq |S|$. Se todos os vértices de cada classe $c_i \in S$ encontram-se em sequência na ordenação $\pi$, então, quando a instância $(G, \pi)$ é dada ao* Guloso*, a solução devolvida $S'$ também é viável, além disso $|S'| \leq |S|$.*

**Teorema 5.2.** *Seja $G$ um grafo onde $S = \{c_1, c_2, \ldots, c_k\}$ é uma coloração de vértices ótima. Então, existem, no mínimo, $k! \prod_{i=1}^{k} |S_i|!$ permutações de vértices que, quando passadas ao* Guloso *como $\pi$, levam o algoritmo a devolver uma solução ótima.*

## 5.2 Algoritmo DSatur

DSatur significa *Degree of Saturation*, o que é traduzido como Grau de Saturação. O Algoritmo Dsatur tem funcionamento semelhante ao Guloso, sendo que a grande diferença entre esses dois reside na forma em que a ordem dos vértices a serem coloridos é determinada. Diferente do Guloso que já recebe na entrada a ordem na qual os vértices vão ser lidos, o DSatur usa o conceito de grau de saturação do vértice para escolher qual vai ser o próximo vértice a ser colorido.

**Definição 5.1** (Grau de Saturação)**.** *Dado um grafo $G$ que está tendo uma coloração construída por um algoritmo. Dado $v \in V(G)$, dizemos que o* ***grau de saturação*** *de $v$ naquele momento é igual à quantidade de cores diferentes atribuídas aos vizinhos de $v$. Considere que vértices ainda não coloridos pelo algoritmo não são contabilizados no grau de saturação. Denotamos o grau de saturação de $v$ por $sat(v)$.*

Então, como podemos ver no Algoritmo 3, o DSatur nada mais é que um algoritmo heurístico para o problema da coloração. A ideia heurística por trás desse algoritmo é dar preferência aos vértices que já estão mais restritos, sendo que o critério para avaliar isso é o grau de saturação.

Sobre tempo de execução,

Como o procedimento de coloração do DSatur é consideravelmente menos aleatório que do Guloso, em algumas topologias de grafos o DSatur devolve soluções exatas. Seguem dois teoremas que apresentam topologias onde o DSatur é algoritmo exato.

**Teorema 5.3.** *Seja $G$ um grafo bipartido, então* DSatur*$(G)$ devolve uma solução exata.*

**Teorema 5.4.** *Seja $G$ um ciclo ou uma roda, então* DSatur*$(G)$ devolve uma solução exata.*

### 5.3 Algoritmo Recursive Largest First (RLF)

## 6 Algoritmos Exatos

### 6.1 Algoritmos de Programação Dinâmica

#### 6.1.1 Algoritmo de Lawler

#### 6.1.2 Algoritmo de Beigel e Eppstein

### 6.2 Algoritmos de Branch-and-Bound e Backtracking

#### 6.2.1 Algoritmo de Brown

#### 6.2.2 Algoritmo DSatur

## 7 Próximos Passos

## Referências


[1] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 1846289696.

[2] C. N. Campos. *O problema da coloração total em classes de grafos*. PhD thesis, Instituto de Computação - Universidade Estadual de Campinas, 2006.

[3] T. H. Cormen, C. E. Leiserson, R. Rivest, and S. C. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009. ISBN 0262033844.

[4] R. M. Karp. Reducibility Among Combinatorial Problems. 1972. URL https://people.eecs.berkeley.edu/~luca/cs172/karp.pdf.

[5] R. M. R. Lewis. *A Guide to Graph Colouring*. Springer International Publishing, 1st edition, 2016. ISBN 978-3-319-25730-3.

# A Grafos

Considere que $\delta(G)$ é o grau mínimo de $G$ e que $\Delta(G)$ é o grau máximo de $G$.

Ao longo desse trabalho sempre nos referiremos ao conjunto de vértices de um grafo $H$ como $V(H)$ e ao conjunto de arestas do mesmo grafo como $E(H)$.

Um grafo é nulo se ele não possui arestas.

# B Demonstrações

**Demonstração do Teorema 3.1**

*Demonstração.* Seja $\{S_1, S_2, \ldots, S_k\}$ uma coloração mínima para o grafo $G$. Para todo par de cores $S_i$ e $S_j$, onde $1 \leq i \leq j \leq k$, existe ao menos uma aresta com um extremo em $S_i$ e outro em $S_j$, pois, caso contrário, poderíamos unir as duas cores, o que seria contraditório, já que essa é uma coloração mínima.

Dessa forma, $|E(G)| \geq k$, não somente isso, mas $|E(G)| \geq \binom{k}{2} = \frac{k(k-1)}{2} = \frac{k^2-k}{2}$. Como $k \geq 1$, vale que

$$|E(G)| \geq \frac{k^2-k}{2} \geq \frac{k^2-1}{2} = 2\left[\left(k-\frac{1}{2}\right)^2 - \frac{1}{4}\right]$$

Desenvolvendo a desigualdade acima chegamos em $k = \chi(G) \leq \frac{1}{2} + \sqrt{2|E(G)| + \frac{1}{4}}$. □

**Demonstração do Teorema 3.2**

*Demonstração.* Vamos provar por contradição. Para isso, vamos assumir que $G$ é k-crítico, mas que $\delta(G) < k-1$. Seja $v$ um vértice com grau $\delta(G)$. Como $G$ é k-crítico, então $\chi(G-v)$[1] $\leq k-1$.

Seja $c_1, c_2, \ldots, c_{k-1}$ as cores de uma $(k-1)$-coloração do grafo $G-v$. Como $\delta(G) \leq k-2 < k-1$, sabemos que $v$ é adjacente a no máximo $k-2$ vértices. Então, com certeza, $v$ não é adjacente a nenhum vértice de pelo menos uma das $k-1$ cores dessa coloração. Suponha que $c_i$ seja uma cor que não possui vértice adjacente a $v$ no grafo $G$. Então, ao adicionarmos $v$ a $c_i$ encontramos uma $k-1$ coloração para $G$, o que é uma contradição, afinal, $G$ é grafo $k$-crítico. □

**Demonstração do Corolário 3.1**

[1]Seja $G$ um grafo e $v \in V(G)$, representamos por $G-v$ o subgrafo de $G$ tal que $V(G-v) = V(G) - v$ e $E(G-v) = E(G) - \{$conjunto de arestas em E(G) com um dos extremos em $v\}$.

*Demonstração.* Seja $G$ um grafo $k$-cromático, então, sabemos que existe um subgrafo $H$ de $G$ tal que $H$ é $k$-crítico. Pelo Teorema 3.2 sabemos que $\delta(H) \geq k-1$, isso implica que $\delta(G) \geq k-1$.

Como $H$ é $k$-crítico, então, por definição, $H$ é $k$-cromático e $H$ tem ao menos $k$-vértices. Dessa forma, como $H \subseteq G$ vale que $G$ tem ao menos $k$ vértices com grau maior ou igual a $k-1$. □

**Demonstração do Corolário 3.2**

*Demonstração.* Vamos provar por contradição. Para isso, vamos assumir que $\chi(G) > \Delta(G)+1$. Isso implica que $\chi(G) - 1 > \Delta(G)$, mas veja que isso não pode ser verdade, pois, pelo Corolário 3.1, sabemos que existem ao menos $k$ vértices com grau maior ou igual a $k-1$, sendo que $k = \chi(G)$. □

**Demonstração do Teorema 3.3**

*Demonstração.* Vamos realizar essa demonstração por partes. Na primeira parte vamos considerar que $G$ não é regular, e depois vamos provar para o caso em que $G$ é regular. Sendo que no caso em que $G$ é regular vamos separar a demonstração em uma parte em que $G$ tem vértice de corte e em outra onde $G$ não tem.

Vamos começar com o caso em que $G$ não é regular. Seja $v \in V(G)$, onde $d(v) = \delta(G)$. Seja $T$ uma árvore de busca com raiz em $v$. Vamos colorir cada vértice de $G$ usando a heurística gulosa dado no Algoritmo 1, mas vamos fazer a seleção do vértice a ser colorido escolhendo uma folha qualquer de $T' \subseteq T$, onde $T'$ é a subárvore induzida pelos vértices ainda não coloridos, isso a cada passo do processo de coloração. Perceba que o último vértice a ser colorido é a raiz $v$.

Dado $w \in V(G)$, sendo que $w \neq v$, quando $w$ for colorido temos certeza que ele é vizinho de ao menos um vértice ainda não colorido (nó pai). Então ele é vizinho de no máximo $\Delta(G) - 1$ vértices já coloridos. Logo, $w$ será atribuído com uma cor entre $1, \ldots, \Delta(G)$.

No momento em que $v$ for colorido, será atribuída uma cor entre $1, \ldots, \Delta(G)$, afinal, ele possui $\delta(G) \leq \Delta(G) - 1$ vizinhos, pois o grafo não é regular, ou seja, $\delta(G) \neq \Delta(G)$. Isso prova o teorema para o caso em que $G$ não é regular.

Agora, suponha que $G$ é um grafo regular que possui um vértice de corte $v$[2]. Então, considere os dois grafos $G_1, G_2 \subseteq G$ tais que $G = G_1 \cup G_2$ e $\{v\} = G_1 \cap G_2$. Vamos denotar por $d_{G_i}(v)$ o grau do vértice $v$ no grafo $G_i$. Veja que $d_{G_i}(v) < \Delta(G)$, então nenhum dos grafos $G_i$ é regular, logo, pelo caso que acabamos de provar, $\chi(G_i) \leq \Delta(G_i) = \Delta(G)$. Como $G_1 - v$ não possui intersecção com $G_2$, vale que $max\{\chi(G_1), \chi(G_2)\} \leq \Delta(G)$.

[2]Um vértice $v \in V(G)$ é dito vértice de corte em $G$, se sua remoção do grafo aumenta o número de componentes.

Por fim, suponha que $G$ é um grafo regular que não possui vértice de corte.

Se toda busca em profundidade em um grafo $H$ é um ciclo hamiltoniano, onde o nó raiz é uma das pontas, então $H$ ou é ciclo, ou é grafo completo, ou é grafo bipartido completo [1].

Suponha que toda busca em profundidade em $G$ resulte num caminho hamiltoniano. Como, por hipótese, $G$ não é ciclo ímpar nem grafo completo, então $G$ é grafo bipartido completo, onde $\chi(G) = 2 \leq \Delta(G)$.

Agora, caso nem toda busca em profundidade em $G$ resulte num caminho hamiltoniano, como descrito. Dessa forma, seja $T$ uma árvore de busca em $G$ que não é caminho hamiltoniano. Seja $v$ um vértice com ao menos 2 filhos em $T$, digamos $x$ e $y$. Como $G$ é 2-conexo, $G - x$ e $G - y$ também são conexos. Então, devido às características do grafo e à forma como uma busca em profundidade é realizada, podemos dizer que $x$ e $y$ são ambos folhas ou possuem descendentes conectados a $v$ no grafo $G$. Então $G' = G - x, y$ é um grafo conexo. Considere uma árvore de busca em largura $T'$ no grafo $G'$ e com raiz em $v$.

Para finalizar, vamos atribuir a $x$ e $y$ a cor 1 e vamos colorir os vértices da árvore $T'$ usando heurística gulosa. Ao fim, obtemos uma $\Delta(G)$ coloração.

□

**Demonstração do Teorema 4.2**

*Demonstração.* Considere uma instância $\phi$ do 3-SAT, onde $c_1, c_2, \ldots, c_m$ são as cláusulas definidas sobre as variáveis $\{x_1, x_2, \ldots, x_n\}$. Com base nessa instância do 3-SAT vamos construir um grafo $G$ instância de 3-coloração, para isso, a forma como vamos construir $G$ vai considerar os seguintes aspectos:

1. A necessidade de estabelecer um paralelo entre os valores verdade de cada variável em $\phi$ com as cores de $G$.

2. A necessidade de capturar a satisfatibilidade de cada cláusula em $\phi$.

Para que $G$ considere os dois aspectos levantados vamos primeiro criar um triângulo em $G$ com os vértices $t, f, b$, onde $t$ significa verdadeiro (*true*), $f$ significa falso (*false*) e $b$ significa *base* (*base*). Cada um desses três vértices representa uma cor que vai ser usada para colorir $G$, afinal, como esse é um triângulo já sabemos que cada um terá uma cor diferente.

O próximo passo é adicionar dois vértices $v_i$ e $v_i'$ para cada variável $x_i$, sendo que cada par desses vértices formará um triângulo com o vértice base e representará os literais da variável. Portanto, impomos ao grafo $G$ que se $v_i$ tem a mesma cor que $t$ então $v_i'$ tem mesma cor que $f$, e vice versa. Essa construção de $G$ permite um paralelo preciso entre os valores verdade de cada literal $x_i$ em $\phi$ e a cor dos vértices $v_i$ e $v_i'$ em $G$. Dessa forma, o primeiro aspecto

listado já está sendo considerado nessa construção de instância. Veja a construção de $G$ até o momento na Figura B.

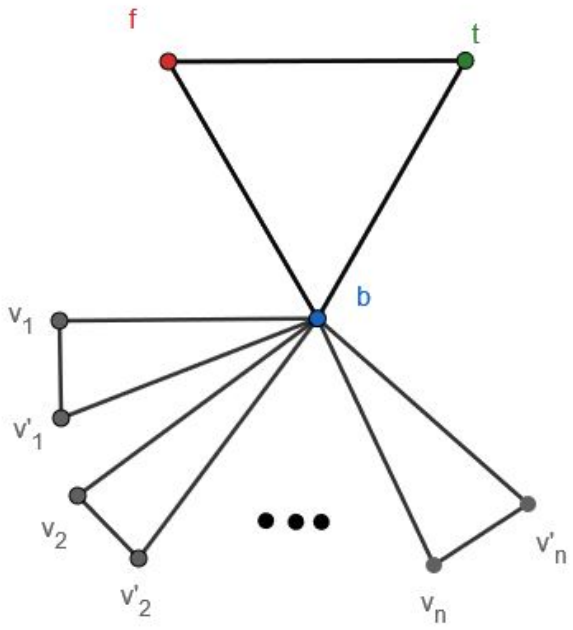


Figura 2: Construção parcial da instância $G$

Agora, vamos adicionar restrições a $G$ de forma que o segundo aspecto citado também seja considerado. Para isso vamos usar o dispositivo de satisfatibilidade da cláusula, sendo que esse dispositivo nada mais é do que uma forma de expressar a operação lógica OR entre os três literais de cada cláusula em $\phi$ no grafo $G$. Para obter o resultados dessas operações usamos as cores já comentadas.

Para cada cláusula $c_j = (a \vee b \vee c)$, definimos o dispositivo de satisfatibilidade dessa cláusula como um grafo capaz de representar a operação OR entre os vértices que representam tais literais. Na figura B temos um exemplo de construção do dispositivo descrito.

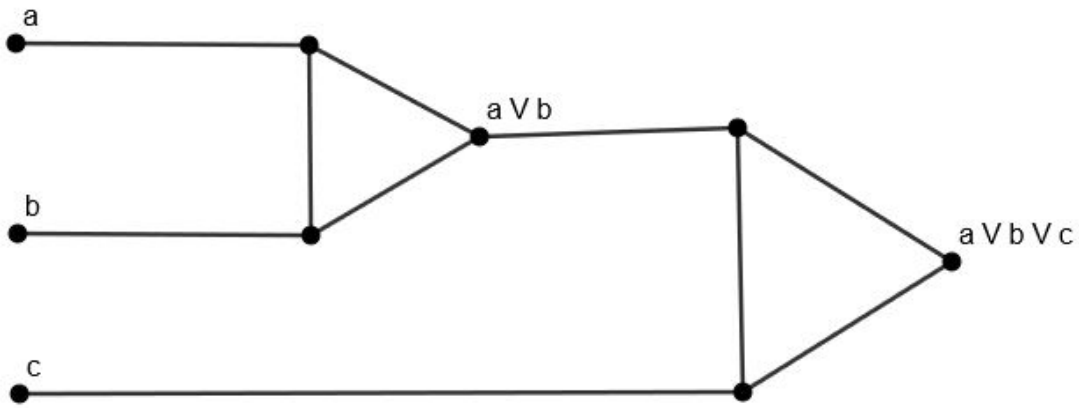


Figura 3: Dispositivo de satisfatibilidade da cláusula

Perceba que essa construção baseada no dispositivo de satisfatibilidade tenta repetir duas vezes a mesma operação, primeiramente fazendo $(a \vee b)$ e depois $((a \vee b) \vee c)$.

É possível perceber que o dispositivo possui as seguintes propriedades:

- Se $a, b, c$ possuem todos cor $f$, então $((a \vee b) \vee c)$ também tem que ser colorido com $f$. Dessa forma, o dispositivo nos indica quando alguma cláusula não é satisfazível.
- Se algum dos vértices $a, b, c$ possui cor $t$, então existe 3-coloração própria para o dispositivo, o que indica que a cláusula considerada possui valoração que a satisfaça.

Perceba então que os dispositivos capturam muito bem o segundo aspecto citado, porém, veja que eles indicam o resultado da operação OR entre os literais considerados. Para considerar a satisfatibilidade expressão booliana $\phi$ é necessário adicionar mais uma estrutura à cada dispositivo conforme podemos ver na Figura B . Com a adição dessa nova estrutura, qualquer outra cor atribuída à operação final do dispositivo que não $t$ indicará que não existe 3-coloração válida para o grafo e portanto $\phi$ não é satisfazível.

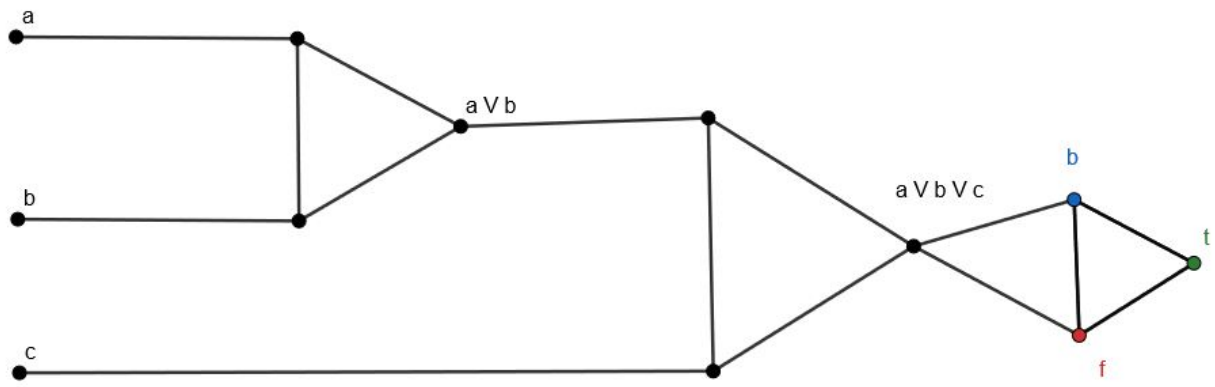


Figura 4: Dispositivo de satisfatibilidade da cláusula em sua versão final

Finalizamos nossa adaptação entre instâncias de forma que agora podemos demonstrar que $\phi$ é satisfazível se e somente se $G$ é 3-colorável.

Vamos começar com a ida. Se $\phi$ é satisfazível então $G$ é 3-colorável. Suponha que $\{y_1, y_2, \ldots, y_n\}$ seja uma valoração das variáveis $\{x_1, x_2, \ldots, x_n\}$ que satisfaça $\phi$. Se $y_i = V$, então $v_i$ é colorido com $t$ e $v_i'$ com $f$, sendo que caso $y_i = F$ o que acontece é análogo. Como $\phi$ é satisfazível, toda cláusula $c_j$ também deve ser satisfazível, isso nos diz que $a, b$ ou $c$ é colorido com $t$. Pela construção de $G$, em especial do dispositivo de satisfatibilidade, sabemos que o dispositivo correspondente a cada $c_j$ é 3-colorível e o vértice de saída do dispositivo é colorido com $t$, isso nos garante que a o grafo $G$ possui 3-coloração própria.

Agora a volta. Suponha que $G$ seja 3-colorível, vamos construir uma valoração das variáveis $\{x_1, x_2, \ldots, x_n\}$ da seguinte forma: se $v_i$ é colorido com $t$, então $x_i$ é valorada com $V$, caso seja colorido com $f$ então $x_i$ é valorada $F$. Por contradição, assuma que $\phi$ não seja satisfazível com a valoração construída, isso significa que ao menos uma cláusula não foi satisfeita com tal valoração. Então, pela construção de $G$, o vértice de saída dos dispositivos que representam as cláusulas não satisfeitas devem assumir valor $f$, porém, também pela construção de $G$, isso cria um conflito na coloração que faz com que $G$ não seja 3-colorável, contradição com o que já assumimos. □

# C Algoritmos

**Algoritmo 1:** GulosoBásico($G$)

**Entrada:** Grafo $G$
**Saída:** Coloração do grafo $G$ na forma de conjuntos independentes

Nomeie cada vértice de $G$ como $v_1, v_2, \ldots, v_n$, onde $n = |V(G)|$
**para** $i = 1$ *até* $n$ **faça**
    Colorir $v_i$ com o menor valor ainda não atribuído a um de seus vizinhos.
**retorna** *Coloração dos vértices de G no formato de coleção de conjuntos independentes*

**Algoritmo 2:** Guloso($G$, $\pi$)

**Entrada:** Grafo $G$ e sequência $\pi = (v_1, \ldots, v_n)$ dos vértices desse grafo, tal que $n = |V(G)|$
**Saída:** Coloração $\mathcal{S}$

Seja $S_1 = \emptyset$
Seja $\mathcal{S} = \{S_1\}$
**para** *todo* $k = 1$ *até* $n$ **faça**
    $adicionou = False$
    $j = 1$
    **enquanto** $j \leq |\mathcal{S}|$ *e* $adicionou == False$ **faça**
        **se** $S_j \cup \{v_k\}$ *é conjunto independente* **então**
            $S_j = S_j \cup \{v_k\}$
            $adicionou = True$
        $j = j + 1$
    **se** $adicionou == False$ **então**
        Crie $S_j = \{v_k\}$
        $\mathcal{S} = \mathcal{S} \cup \{S_j\}$
**retorna** $\mathcal{S}$

**Algoritmo 3:** DSatur($G$)

**Entrada:** Grafo $G$
**Saída:** Coloração $\mathcal{S}$

```
1  Seja S_1 = ∅
2  Seja 𝒮 = {S_1}
3  enquanto |𝒮| ≠ |V(G)| faça
4  |  adicionou = False
5  |  Escolha v ∈ V(G) tal que sat(v) é mínimo
6  |  j = 1
7  |  enquanto j ≤ |𝒮| e adicionou == False faça
8  |  |  se S_j ∪ {v} é conjunto independente então
9  |  |  |  S_j = S_j ∪ {v}
10 |  |  |  adicionou = True
11 |  |  j = j + 1
12 |  se adicionou == False então
13 |  |  Crie S_j = {v}
14 |  |  𝒮 = 𝒮 ∪ {S_j}
15 retorna 𝒮
```

Universidade Federal do ABC
Centro de Matemática, Computação e Cognição
Relatório final de pesquisa – Iniciação Científica
Universidade Federal do ABC
Processo 2020/09330-7

# Algoritmos para Coloração de Grafos

Relatório de pesquisa referente ao segundo período, entre 10/04/2021 até 31/05/2021

*Aluno:*
Wesley Lima de Araujo

*Supervisor:*
Prof. Dra. Carla Negri Lintzmayer

Agosto/2021

**Resumo**

Problemas de coloração de grafos são muito importantes pois modelam situações em que há conflitos entre objetos e deseja-se separá-los em grupos de acordo com algum critério, como é o caso em problemas de alocação, por exemplo. Esse relatório final, destinado à FAPESP, e critério de avaliação do projeto de iniciação científica 2020/09330-7, visa informar à FAPESP sobre como a segunda etapa do projeto foi realizada. O principal objetivo desse projeto foi expandir os conhecimentos do aluno sobre desenvolvimento de algoritmos e otimização combinatória através do estudo de diversos tópicos relacionados à coloração de grafos, especialmente algoritmos que visam resolver o problema da coloração mínima.

**Assuntos:** Coloração de Grafos; Algoritmos; Otimização Combinatória.

# Sumário

# 1 Introdução

Ao longo desse relatório usaremos amplamente definições e conceitos de teoria dos grafos. Para qualquer conceito que não for aqui definido, estará sendo considerada a definição apresentada no livro *"Graph Theory"* de Bondy e Murty [5], e o mesmo vale para notações. Também vale ressaltar que quando usamos o termo grafo estaremos nos referindo a grafos simples.

O **problema de coloração de vértices** (também chamado de coloração de grafos), de maneira intuitiva, visa rotular os vértices de um grafo com cores, de maneira que não haja conflito entre vértices vizinhos. Um conflito ocorre quando dois vértices vizinhos possuem a mesma cor. Dessa forma, cada aresta representa a possibilidade de um conflito. Esse é um problema aplicável em diversas situações do mundo real onde desejamos modelar situações com conflitos, e para tal basta que representemos os objetos modelados como vértices e os conflitos entre eles como arestas. Problemas de alocação são um bom exemplo de problema que pode ser modelado e resolvido por coloração de vértices.

Por exemplo, imagine um problema de construção de calendário em que temos algum conjunto de atividades (palestras, aulas, jogos, etc) e um conjunto de horários em que essas atividades podem ocorrer. Desejamos alocar as atividades nesses horários mas somos limitados por um conjunto de restrições (pessoas, salas, etc). Veja que nesse caso podemos representar cada atividade como um vértice e cada restrição como uma aresta, sendo que nossa busca por marcar um horário compatível se resume a colorir o grafo que construímos, onde cada cor é um horário.

Dado um grafo $G$, chamamos de **coloração própria** de $G$ uma coloração dos vértices em $V(G)$ que respeite as restrições impostas, ou seja, uma coloração é própria se dois vértices vizinhos em $G$ nunca têm a mesma cor. Normalmente, quando falamos de uma coloração qualquer estará implícito que estamos falando de uma coloração própria. Por isso, ao longo desse texto, a não ser que seja especificado o contrário, quando usarmos o termo coloração estaremos nos referindo a uma coloração própria. Se um grafo $G$ aceita uma coloração que usa $k$ cores, falamos que trata-se de uma **$k$-coloração**, além disso, falamos que $G$ é um grafo **$k$-colorível** ou **$k$-colorável**.

Na grande maioria das vezes, não nos interessa apenas encontrar uma resolução qualquer do conflito proposto, mas também encontrar uma solução com o menor número de cores possível. Então, no nosso exemplo de calendário seria como se, além de querer construir o calendário, quiséssemos usar o menor número de horários possíveis. Também evidente que caso quiséssemos usar o maior número de cores possível bastaria associar à cada atividade (vértice) um horário (cor).

Algumas considerações sobre notações e conceitos usados nesse relatório são as seguintes. Sempre nos referiremos ao conjunto de vértices de um grafo $H$ como $V(H)$ e ao conjunto de arestas do mesmo grafo como $E(H)$. Também considere que $\delta(G)$ é o grau mínimo de $G$ e que $\Delta(G)$ é o grau máximo de $G$. Um grafo é nulo se ele não possui arestas. Por fim, se $S \in V(G)$, representamos por $G[S]$ o subgrafo de $G$ induzido pelos vértices em $S$.

Assim sendo, o problema de coloração de vértices passa a ser um problema de otimização combinatória quando desejamos não só encontrar uma coloração, mas sim uma coloração com o menor número possível de cores. Chamamos essa quantidade mínima de cores possível para uma coloração de um grafo $G$ de **número cromático do grafo** $G$, denotado por $\chi(G)$. Chamamos uma coloração de um grafo $G$ que use $\chi(G)$ cores de **coloração ótima de** $G$. Além disso, se para um grafo $G$ temos que $\chi(G) = k$, falamos que $G$ é um **grafo $k$-cromático**.

Segue a definição formal do problema de Coloração Mínima de Vértices, em que buscamos encontrar uma coloração ótima para um grafo.

**Problema 1.1** (Coloração Mínima de Vértices)**.** *Dado um grafo $G$, sendo que $n = |V(G)|$, no problema da Coloração Mínima de Vértices (CMV), desejamos associar a cada vértice $v \in V(G)$ um inteiro $c(v) \in \{1, \ldots, k\}$ de forma que:*

- *$c(v) \neq c(u)$, caso $\{v, u\} \in E(G)$;*
- *$k$ é mínimo.*

*Nessa definição do problema cada valor inteiro em $\{1, \ldots, k\}$ é chamado de **cor**.*

A definição para o CMV que acabamos de dar não é a única possível. Ela foi retirada do livro *"A Guide to Graph Colouring"* de Lewis [11], que também dá outras definições, inclusive uma que usa o conceito de conjuntos independentes. Dizemos que um subconjunto $S$ dos vértices de um grafo $G$ é um **conjunto independente** se quaisquer dois vértices $u, v \in S$ não são vizinhos. Portanto, toda coloração viável (coloração que respeite os conflitos) nada mais é que uma coleção de conjuntos independentes de $G$ onde os vértices do grafo pertencem a um, e somente um, conjunto da coleção. Segue essa definição alternativa para o CMV.

**Problema 1.2** (Coloração Mínima de Vértices)**.** *Dado um grafo $G$, desejamos encontrar uma coleção $\mathcal{S} = \{S_1, \ldots, S_k\}$, onde cada $S_i \in \mathcal{S}$ é um conjunto independente em $G$ tal que*

$$\bigcup_{i=1}^{k} S_i = V(G) \tag{1}$$

$$S_i \cap S_j = \emptyset, \quad 1 \leq i \neq j \leq k \tag{2}$$

$$\forall u, v \in S_i, \{u, v\} \notin E(G), \quad 1 \leq i \leq k \tag{3}$$

*sendo que $k$ deve ser mínimo.*

*Nessa definição do problema, cada conjunto independente $S_i$ é chamado de* ***cor****.*

Observe que a definição alternativa define exatamente o mesmo problema, sendo a grande diferença a forma como o problema é formalmente representado.

A partir daqui, sempre que falarmos "problema da coloração de grafos" ou "problema da coloração de vértices", estaremos nos referindo ao CMV.

Para resolvermos o problema da coloração de grafos podemos usar algoritmos que recebem uma instância do problema (grafo $G$ qualquer) e devolvem uma solução para essa instância (coloração ótima em $G$). Infelizmente, o problema da coloração de grafos possivelmente não pode ser resolvido em tempo polinomial, afinal, o problema de decisão que decide se um número natural $k$ é número cromático para um grafo $G$ é $\mathcal{NP}$-completo [10], e, por consequência, encontrar uma coloração ótima de um grafo $G$ é $\mathcal{NP}$-difícil. Dada essa realidade, não temos esperança em desenvolver um algoritmo para coloração de vértices que:

1. seja computacionalmente eficiente; e
2. devolva uma solução ótima para qualquer instância de entrada.

Porém, isso não impede os projetistas de algoritmos de desenvolverem algoritmos para o problema, a questão é que é necessário sacrificar ou tempo de execução eficiente ou otimalidade da solução para toda instância. Baseando-se nessa ideia de sacrifício, existem técnicas de projeto de algoritmos que seguem três abordagens distintas com relação aos dois itens citados acima:

- **Abordagem exata:** Nessa abordagem sempre garantimos que a solução devolvida pelo nosso algoritmo e ótima, porém, para problemas pertencentes a $\mathcal{NP}$-difícil esse tipo de algoritmo é ineficiente para instâncias que sejam muito grandes;
- **Abordagem heurística:** Um algoritmo é considerado heurístico se ele possuir tempo de execução polinomial para toda instância e devolve uma solução viável;
- **Abordagem por aproximação:** Dizemos que um determinado algoritmo é de aproximação se para qualquer instância o tempo de execução é polinomial e existe certa garantia sobre a qualidade da solução, mas nem sempre a solução devolvida é ótima. Especificamente, um algoritmo é uma $\alpha$-aproximação se para qualquer instância recebida ele devolve uma solução cujo custo está a um fator de no máximo $\alpha$ do custo da solução ótima.

O objetivo desse projeto foi ampliar a experiência do candidato com pesquisa científica na área de Ciência da Computação, bem como a complementar a sua formação, ampliando seu conhecimento na área de otimização combinatória e, principalmente, no projeto de algoritmos por meio do estudo de algoritmos e problemas relacionados a coloração de grafos.

Nesse relatório pretendemos expor os assuntos que foram estudados pelo aluno, mas de maneira resumida, para que fique bem claro quais tópicos foram vistos sem que haja necessidade do avaliador entrar em todos os pormenores dos assuntos estudados. Lembramos que a metodologia adotada para realização do projeto consistiu no aluno estudar o assunto em questão entendendo as demonstrações e conceitos envolvidos e, em reuniões de frequência semanal, discutir com a orientadora sobre o assunto e sanar dúvidas que possam ter surgido. Algo importante a se destacar é que as demonstrações de vários teoremas citados nesse texto foram estudados detalhadamente, além de algumas demonstrações de teoremas sobre assuntos não cobertos por esse relatório.

Esse relatório está dividido em seções. A Seção 2 apresenta o conceito de meta-heurísticas, alguns conceitos de otimização relacionados a meta-heurísticas e, por fim, algumas meta-heurísticas aplicadas à coloração de grafos. A Seção 3 apresenta um resultado sobre a inaproximabilidade do problema de coloração de vértices e um resumo de alguns artigos lidos que apresentam uma espécie de evolução dos resultados em algoritmos de aproximação para coloração de grafos. A Seção 4 traz alguns tópicos mais atuais em pesquisas de algoritmos para coloração de grafos, tendo como principais referências artigos recentes (após 2019). Por fim, a Seção 5 traz as considerações finais sobre a realização do projeto. Lembrando que esse é o relatório final do projeto, então os tópicos relatados aqui foram abordados durante a segunda etapa do projeto, entre 10/04/2021 e 10/08/2021.

# 2 Algoritmos Meta-Heurísticos

Nessa seção falaremos sobre alguns algoritmos heurísticos para o problema da coloração de grafos, porém, diferentemente dos algoritmos heurísticos apresentados no relatório anterior, aqui focaremos em apresentar meta-heurísticas e em como elas são usadas para encontrar soluções do problema de coloração de grafos. A principal referência usada nessa seção é o livro de Lewis [11], pois nele existe uma seção dedicada às meta-heurísticas aplicadas na coloração de grafos.

Vamos começar relembrando que um algoritmo é dito heurístico se ele possuir tempo de execução polinomial para toda instância e se devolve sempre uma solução viável. Segundo Lewis [11], o campo de pesquisa em algoritmos heurísticos pode ser considerado o mais frutífero para o problema da coloração dos grafos. Geralmente, heurísticas devolvem soluções

não exatas, podendo até não reconhecer uma solução ótima. Porém, algumas heurísticas podem devolver excelentes resultados quando aplicadas a grafos específicos, sendo inclusive mais recomendadas que algoritmos exatos em alguns casos onde o tamanho da instância é muito grande.

Não existe definição única de meta-heurística, porém, sua ideia é ser um padrão algorítmico desenvolvido para ser aplicável a diversos problemas de otimização sendo necessário apenas algumas alterações em sua implementação e projeto. A grande vantagem das meta-heurísticas é justamente essa adaptabilidade a diversos problemas. Alguns exemplos de meta-heurísticas são: busca tabu, *Simulated Annealing*, algoritmos evolucionários e algoritmos de colônias de formigas [8] .

Geralmente, meta-heurísticas aprendem e exploram sobre o espaço de solução do problema enquanto executam. Dessa forma, podem ir melhorando a tomada de decisão até devolverem a melhor solução encontrada. Dito isso, algoritmos meta-heurísticos possuem duas características mais marcantes: a primeira é que em sua execução eles costumam ser feitos para explorar o maior número de soluções viáveis para o problema sem perder o controle do tempo de execução, e a segunda é que sempre temos uma solução momentaneamente tomada, ou seja, o local onde o algoritmo se encontra no momento influencia nos próximos passos que ele vai tomar.

Nessa seção começaremos apresentando a questão dos espaços de soluções, pois, segundo Lewis [11], é útil dividir as meta-heurísticas dessa forma. Depois de apresentar os possíveis espaços de soluções falaremos sobre o conceito de vizinhança e sua importância para meta-heurísticas. Ainda depois apresentaremos especificamente sobre como algumas meta-heurísticas são aplicadas ao problema de coloração de grafos: TabuCol e o Algoritmos Evolucionários. Vale ressaltar que outros algoritmos meta-heurísticos vistos ao longo do projeto foram PartialCol, Hill-Climbing, Colônia de Formigas e Backtracking.

## 2.1 Os Espaços de Soluções

Quando falamos de espaços de solução nos quais meta-heurísticas podem trabalhar, temos que entender a ideia de como essas meta-heurísticas operam quanto a otimização. Uma meta-heurística sempre começa com alguma solução do problema e então ela vai explorando algum espaço de soluções possíveis em busca de uma melhor solução viável. É nessa ação que entra a ideia de otimização. Por exemplo, no caso da coloração imagine que uma meta-heurística qualquer começa com uma solução inviável e então ela vai alterando as cores dos vértices, seguindo alguma regra, até obter uma solução viável, e que depois disso, ela busca minimizar a quantidade de cores com o mesmo procedimento. Essa atividade caracteriza uma exploração do espaço de soluções.

Podemos dividir os espaços de soluções em três tipos usando como critério a viabilidade das soluções contidas nele. Segue uma explicação de cada um desses tipos e como isso aplica-se ao problema da coloração de grafos:

- **Espaço de Soluções Viáveis -** Nesse tipo de espaço trabalha-se apenas com soluções viáveis. Os algoritmos de coloração de grafos que trabalham nesse espaço geralmente usam de alguma forma um algoritmo Guloso para encontrar permutações de vértices que possam reduzir o número de cores. Veja que com esse procedimento o algoritmo nunca sai do espaço de soluções viáveis.

- **Espaço de Soluções Inviáveis -** É o espaço onde a maioria das meta-heurísticas trabalha [11] e contém soluções viáveis e também inviáveis. Geralmente, os algoritmos de coloração que trabalham nesse espaço começam estabelecendo um número $k$ de cores e então procuram uma coloração viável para o grafo instância com $k$ cores. Caso encontrem tal coloração eles diminuem o valor de $k$ e repetem o processo, caso não encontrem tal coloração podem aumentar o valor de $k$. Uma coisa a se destacar sobre essa abordagem é que a cada iteração estamos tentando resolver um problema $\mathcal{NP}$-difícil.

- **Espaço de Soluções Parciais -** Essa abordagem recebe menos atenção que as outras duas [11] e tem como principal característica conter soluções parciais do problema em questão. Quando aplicada ao problema da coloração de grafos, geralmente, segue a estratégia de construir um conjunto de vértices ainda não coloridos $U$. Quando ocorre um conflito, descolorimos algum vértice e o adicionamos a esse conjunto $U$.

Em uma mesma estratégia de resolução podemos ainda combinar espaços de solução, sendo que diferentes espaços de solução são usados em diferentes etapas da resolução. Por trás dessas combinações temos o objetivo de diversificar nossa busca, já que estratégias trabalhando em espaços de solução diferentes dificilmente terão o mesmo ótimo local.

## 2.2 Vizinhança

Dada uma solução $S$ para o problema da coloração de grafos, ou seja, uma coloração dos vértices do grafo instância, falamos que um **operador de vizinhança** é uma operação que quando recebe a solução $S$ modifica ela seguindo alguma regra e devolve uma outra solução para o problema, digamos $S'$. Toda solução $S'$ gerada dessa forma é chamada de **solução vizinha** de $S$, e o conjunto de todas as $S'$ é chamado **vizinhança** de $S$.

No caso do problema de coloração de vértices o operador mais comum é a troca de cor de um dos vértices. Dessa forma, quaisquer duas soluções que tenham somente a cor de

um vértice como diferença são chamadas de vizinhas. Para mensurar a distância de uma solução a outra no espaço de soluções geralmente usamos a quantidade de vértices com cores diferentes.

## 2.3 Algoritmo TabuCol

Para começar a falar do Algoritmo TabuCol vamos falar sobre dois algoritmos de exploração de espaço de soluções, a Descida Íngreme e a Busca Tabu:

- **Descida Íngreme -** Dada uma solução $S$ calculamos o custo de todas as soluções vizinhas de $S$. Vamos considerar que $S'$ é o vizinho de $S$ com menor custo. Assim, o algoritmo se movimenta para $S'$ e repete o processo de calcular todos os vizinhos de $S'$. Caso não haja um vizinho com custo menor o algoritmo para na solução do momento. Esse algoritmo tem o claro problema de cair em ótimos locais. Caso estivéssemos falando de um problema de maximização usaríamos a **Subida Íngreme**, que é um algoritmo análogo ao da descida.

- **Busca Tabu -** Trata-se de uma generalização do algoritmo de descida íngreme, apresentando um mecanismo para escapar de ótimos locais. Basicamente, aplicamos a descida íngreme de forma que quando identificamos que o algoritmo caiu em ótimo local (não há melhora no custo da solução) permite-se que ocorram movimentos para soluções piores. Para evitar rodar em círculos é mantida uma lista de soluções chamadas lista tabu. O algoritmo é impedido de voltar a alguma solução nessa lista durante algumas iterações. A forma como a lista tabu é gerenciada (tempo de uma solução na lista em iterações) é algo parametrizável.

Apresentados esses dois algoritmos de exploração no espaço de soluções vamos falar do TabuCol (Coloração Tabu). O TabuCol opera em um espaço de soluções impróprias e sempre opera em função de diminuir a quantidade de colisões. Consideramos que uma **colisão** ocorre quando dois vértices vizinhos possuem a mesma cor. O TabuCol parte com um número $k$ de cores, podendo a solução de partida ser própria ou imprópria.

Dada uma solução $S = \{S_1, S_2, \ldots, S_k\}$ e a lista de vértices $V(G) = \{v_1, v_2, \ldots, v_n\}$, onde cada $S_i$ representa uma cor, um movimento no espaço de soluções a partir de $S$ é feito da seguinte forma: escolhemos um vértice $v \in S_i$ que esteja causando um conflito e movemos $v$ para $S_j \neq S_i$.

Como o nome indica, o TabuCol utiliza um método de busca tabu para explorar o espaço de soluções impróprias. A lista do TabuCol é mantida em uma matriz $T$, de dimensões $n \times k$. Digamos que em uma iteração $\ell$ o algoritmo movimente um vértice $v_a$ de $S_i$ para uma

outra cor $S_j \neq S_i$. Então, na posição $T_{ai}$ colocamos o valor $\ell + t$, onde $t$ é um parâmetro que indica quantas interações o movimento $v_a$ para $S_i$ ficará com *status* tabu. Podemos configurar o TabuCol para mover-se para uma solução tabu se certas condições forem atendidas.

Dada uma solução $S$ e escolhido um vértice $v_a$, devemos calcular cada uma das $k-1$ soluções vizinhas, que geralmente é o processo mais demorado do algoritmo [11]. Após avaliar as soluções vizinhas o movimento é realizado para o vizinho com maior decréscimo de conflitos ou com menor acréscimo. Empates são resolvidos aleatoriamente.

A ideia para escolher um bom valor de $t$ é fazê-lo inversamente proporcional à qualidade da solução, onde entende-se qualidade como menor número de conflitos possíveis. Para certas famílias de grafos específicas pode haver melhores maneiras de determinar o valor de $t$.

## 2.4 Algoritmos Evolutivos

Algoritmos Evolutivos é uma nomenclatura usada para nos referirmos a uma meta-heurística bio-inspirada que baseia-se em alguns mecanismos evolutivos, em especial, recombinação, mutação e seleção. É uma estratégia que se baseia fortemente em aleatoriedade, especialmente nas fases relacionadas à mutação. A ideia dessa meta-heurística é começar com um conjunto de soluções inicial, onde as soluções são chamadas de indivíduos e o conjunto delas de população, e ir evoluindo essa população através da recombinação entre soluções usando operadores específicos, para mutação aleatória e seleção da próxima geração. Para comparar soluções e mensurar a evolução usamos o valor de *fitness*, que nada mais é que a função que dá valor a um indivíduo. Assim, evoluir uma população trata-se de buscar melhores soluções para o problema em questão através de certas operações. Algoritmos Evolutivos trabalham no espaço de soluções impróprias.

Já citamos que as três principais etapas dos Algoritmos Evolutivos são recombinação, mutação e seleção. Agora, vamos explicar detalhadamente a ideia por trás de cada uma dessas etapas:

1. **Recombinação (*Crossover*):** Dados dois indivíduos da população, sendo que cada um deles representa de alguma forma uma solução (viável ou inviável), misturamos os elementos que representam essas soluções através de um operador específico e obtemos uma, ou até mais de uma, nova solução que carrega características das soluções que a geraram. Nessa situação os dois indivíduos originais são chamados de soluções mães, enquanto o indivíduo gerado é a solução filha.

2. **Mutação:** Após obtermos uma solução filha usamos aleatoriedade para fazermos alterações (mutações) aleatórias nela, assim trazendo um "fato novo" para as soluções que temos para o problema, evitando possíveis vieses que podem levar a ótimos locais.

3. **Seleção:** Realizada a mutação, temos agora, além da população original, um conjunto de novos indivíduos e temos que selecionar dentre todas as soluções que temos aquelas que continuarão no processo evolutivo, o que caracteriza uma pressão evolutiva.

Os operadores de recombinação são, basicamente, o principal diferencial entre diferentes Algoritmos Evolutivos, pois são eles que determinam como a hereditariedade é herdada por soluções filhas.

Explicados esses conceitos sobre Algoritmos Evolutivos vamos focar no problema da coloração de grafos. Para o problema de coloração de grafos uma solução nos Algoritmo Evolutivos é representada por uma coleção de conjuntos $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$, onde cada conjunto $S_i$ é uma cor. A função de custo (*fitness*) usada é o número de colisões da solução, assim soluções mais adaptadas são aquelas com menos colisões.

Seguem um exemplo de operador de recombinação citado no livro de Lewis [11]:

- **Greedy Partition Crossover (GPC)** – A ideia desse operador é que o filho herde cores com muitos vértices de suas mães, ou seja, os filhos herdam cores inteiras das mães. A primeira cor a ser herdada é aquela com mais vértices, considerando duas mães $X$ e $Y$. Sem perda de generalidade, suponha que a mãe que possui a cor selecionada seja $X$. Para evitar duplicação de vértices, os vértices pertencentes à primeira cor adicionada são eliminados de ambas as mães. Em seguida, escolhemos a maior cor de $Y$ e adicionamos à solução filha. Então, o processo de atualização de vértices e escolha da cor é repetido, alternando-se entre as mães, até que a solução filha contenha todos os vértices do grafo. Para o GPC podemos escolher um parâmetro $k$ que determina quantas cores queremos que a solução filha tenha. Após obtermos a solução filha, usamos o TabuCol para melhorar localmente a solução.

Para construir a população original usamos um algoritmo heurístico de construção de soluções que seja capaz de gerar soluções impróprias, como o Dsatur. É interessante que a população original seja a mais diversa possível. Para melhorar a qualidade da resposta fornecida por Algoritmos Evolutivos podemos mesclar eles com procedimentos de busca mais agressivos e determinísticos, como a busca tabu.

# 3 Algoritmos de Aproximação

Em problemas de otimização combinatória desejamos encontrar uma melhor solução dentro de um domínio de soluções, de acordo com algum critério, sendo que as variáveis do problema possuem domínios discretos. Diversos problemas de otimização combinatória são $\mathcal{NP}$-difíceis,

ou seja, não há esperança em encontrar algoritmos exatos eficientes para qualquer instância. Um desses problemas, conforme discutido nesse relatório, é o problema da coloração de grafos. Nesse cenário, os algoritmos de aproximação são uma ferramenta útil, pois são executáveis em tempo polinomial e sempre devolvem uma solução cujo custo está garantidamente a um determinado fator de distância do custo da melhor solução.

Nessa seção, vamos falar sobre algoritmos de aproximação para o problema da coloração de grafos. Em especial, vamos falar da inaproximabilidade do problema e de um algoritmo de aproximação para o problema que foi publicados na década de 1980. Todas as referências bibliográficas dessa parte do projeto foram artigos científicos.

## 3.1 Inaproximabilidade

O principal resultado estudado nessa etapa do projeto foi justamente o resultado que iremos apresentar nessa subseção, que trata da inaproximabilidade do problema. O resultado que iremos apresentar foi desenvolvido por Garey e Johnson [7] e, de maneira resumida, o resultado afirma que chegar a uma aproximação de fator 2 para o problema da coloração de grafos é tão difícil quanto resolver um problema $\mathcal{NP}$-completo. Ao longo de seu artigo, Garey e Johnson [7] trabalharam com dois teoremas para provar o resultado dado a seguir, o que resultou em um terceiro teorema que nada mais é do que uma consequência direta dos dois teoremas anteriores. Dados um algoritmo $A$ para coloração de grafos e um grafo $G$, considere que $A(G)$ é a quantidade de cores da solução devolvida pelo algoritmo $A$ quando recebe o grafo $G$.

**Teorema 3.1.** *[13] Se existe um algoritmo polinomial que determina, para todo grafo $G$, se $\chi(G) \leq 3$ ou não, então existe um algoritmo $A$ de coloração de grafos, com tempo polinomial, que garante que $A(G) = \chi(G)$.*

**Teorema 3.2.** *[7] Dadas uma constante $r < 2$ e uma constante $d$, se existe um algoritmo polinomial $A$ para coloração de grafos que garante que $A(G) \leq r.\chi(G) + d$, então, existe um algoritmo de tempo polinomial que determina, para qualquer grafo $G$, se $\chi(G) \leq 3$.*

Enquanto o primeiro teorema já havia sido provado por Stockmeyer [13], o segundo teorema foi provado no próprio artigo, sendo que boa parte do artigo é dedicado a estruturar sua demonstração, apresentando conceitos e lemas usados na demonstração. Com esses dois teoremas chegamos no resultado almejado do artigo.

**Teorema 3.3.** *[7] Dadas constantes $r < 2$ e $d$, se existe um algoritmo $A$ de coloração de grafos com a garantia de que $A(G) \leq r.\chi(G) + d$, então, existe um algoritmo polinomial $A'$ que garante que $A'(G) = \chi(G)$.*

Veja que o Teorema 3.2 nos diz que se existe um algoritmo $A$ que respeita $A(G) \leq r.\chi(G) + d$, então temos um algoritmo $A'$ que determina para qualquer grafo se $\chi(G) \leq 3$. Mas, veja que o Teorema 3.1 diz que se $A'$ existe, então existe um terceiro algoritmo $A''$ para coloração de grafos que devolve uma coloração com $\chi(G)$ cores. Essa cadeia lógica que amarra os Teoremas 3.1 e 3.2 é exatamente o que resulta no Teorema 3.3.

Também vale destacar que existe um resultado mais forte referente à inaproximabilidade do problema de coloração de grafos. Em 1995, Bellare, Goldreich e Sudan [2], mostraram que o problema de coloração de grafos não é aproximável para nenhum fator abaixo de $|V(G)|^{(1/7)-\varepsilon}$, isso para qualquer $\varepsilon > 0$.

## 3.2 Algoritmos de Aproximação para Coloração de Grafos

Nessa subseção falaremos sobre um dos algoritmos de aproximação vistos em artigos. O algoritmo que mostraremos vem do artigo de Wigderson [14] e tem como fator de aproximação $O(n(\log\log n)^2/(\log n)^2)$, o que representa uma melhora do que era o melhor fator de aproximação até aquele momento para o problema de coloração, $O(n/\log n)$.

Ainda no começo de seu artigo, Wigderson [14] cita o grande *gap* que existia na época entre o que sabia-se ser $\mathcal{NP}$-completo no problema de coloração de grafos (fator de aproximação menor que 2) e o fator de aproximação do melhor algoritmo existente até aquele momento ($O(n/\log n)$). Como já dito, o principal objetivo de Wigderson é mostrar um algoritmo de aproximação com fator $O(n(\log\log n)^2/(\log n)^2)$. Para chegar a esse resultado ele mostra cinco resultados importantes em seu artigo artigo, resumimos cada um dos resultados demonstrados:

1. É mostrado um algoritmo de aproximação para coloração de grafos 3-coloráveis. É demonstrado que o algoritmo apresentado sempre devolve uma coloração com até $3\lceil\sqrt{n}\rceil$ cores. Para chegar a esse resultado são usadas algumas propriedades sobre coloração de grafos, em especial, propriedades de grafos 3-coloráveis e de grafos bipartidos.

2. É feita uma extensão das ideias aplicadas ao algoritmo do resultado anterior para qualquer grafo $k$-colorável, onde mostra-se um algoritmo que sempre devolve uma coloração com até $2k\lceil n^{1-(\frac{1}{k-1})}\rceil$ cores.

3. Nos dois algoritmos anteriores, assumia-se que o número cromático do grafo $G$ já era conhecido. Agora, vamos tentar contornar isso tentando encontrar o valor de $\chi(G) = k$. Para encontrar o valor de $k$ usa-se uma busca binária da seguinte forma: definimos $k = 2^\ell$, para $\ell = 1$ e usamos o algoritmo anterior; caso não seja encontrada coloração viável vamos aumentando o valor de $\ell$ de um em um até encontrar um valor de $\ell$ onde

a resposta do algoritmo é solução viável, quando encontramos tal valor de $\ell$ aí sim realizamos a busca binária de fato, entre 1 e o $\ell$ encontrado, para encontrar o menor valor de $k$ onde haveria coloração viável;

4. Por fim, é apresentado o algoritmo que melhora a performance de aproximação para a coloração de grafos.

Além do artigo de Wigderson os outros dois artigos que mostram algoritmos de aproximação para o problema da coloração e que foram estudados durante o projeto são o artigo de Berger e Rompel [3], que apresenta um algoritmo com fator de aproximação de $O(n((\log\log n)/\log n)^3)$, e o artigo de Halldórsson [9], que apresenta um algoritmo com fator de aproximação $O(n(\log\log n)^2/(\log n)^3)$.

# 4 Trabalhos Atuais em Coloração de Grafos

Como pode ser observado ao longo do relatório, a maioria dos resultados estudados foram publicados antes da década de 2010. Por isso, entendeu-se que era necessário que o aluno entrasse em contato com resultados mais atuais em algoritmos para coloração de grafos. Dito isso, foi realizada uma pesquisa bibliográfica sobre alguns tópicos em coloração de grafos que, atualmente (de 2019 até hoje), vêm sendo intensamente pesquisados.

Para determinar quais tópicos seriam discutidos nessa seção foi feita uma pequena busca bibliográfica no *Google Scholar*. Após a pesquisa foi determinado que dois tópicos seriam vistos com mais detalhes: **algoritmos de coloração para grafos dinâmicos** e **a relação entre computação paralela e coloração de grafos**. Determinados esses dois temas, foram selecionados alguns artigos relacionados que pudessem dar uma boa introdução aos assuntos.

É importante relembrar que o caráter dessa etapa da pesquisa era introdutório, então não estudamos com grande profundidade os resultados apresentados pelos artigos. Ao invés disso, focamos em entender como os assuntos escolhidos se relacionam com coloração de grafos e o conteúdo estudado até essa etapa.

## 4.1 Coloração de Grafos Dinâmicos

Para estudar esse assunto, escolhemos os artigos de Mertzios, Molter e Zamarev [12], e de Bossek, Neumann, Peng e Sudholt [6]. A contribuição do artigo de Mertzios, Molter e Zamarev para a área é a definição de um problema de otimização usando o conceito de grafo dinâmico (Definição 4.1) e a demonstração de alguns resultados sobre sua complexidade. Já

o artigo de Bossek, Neumann, Peng e Sudholt analisa experimentalmente o comportamento de heurísticas de busca já conhecidas para o problema de coloração dos grafos só que quando aplicadas a grafos dinâmicos.

A maioria dos estudos sobre coloração de grafos trata de grafos estáticos, isso é, grafos que não possuem alterações em sua topologia ao longo do tempo. Porém, a maioria das redes, que usam grafos para serem modeladas, são dinâmicas. Claramente, diversos problemas em grafos são dinâmicos, pois muitas vezes as topologias de redes modeladas estão sujeitas a mudanças ao longo do tempo. Dito isso, parece claro que é necessário analisar problemas com grafos dinâmicos de forma diferente. Recentemente, diversos problemas de otimização combinatória clássicos tem tido sua versão dinâmica estudada [6, 12].

Para deixar a ideia de grafo dinâmico mais clara, vamos definir o conceito de grafo dinâmico que formaliza a ideia de um grafo dinâmico ao longo do tempo.

**Definição 4.1** (Grafo Dinâmico [12])**.** *Um grafo dinâmico é um par* $(G, \lambda)$*, onde* $G$ *é um grafo e* $\lambda : E(G) \to 2^{|V(G)|}$ *é uma função de rótulos que associa a cada aresta do grafo um conjunto de valores discretos que representam etiquetas associadas a tempo.*

Assim, podemos dizer que dado um grafo dinâmico $(G, \lambda)$, a função de rótulos associa a cada aresta $e \in E$ um conjunto de momentos no tempo onde $e$ fica ativa no grafo.

O artigo de Mertzios, Molter e Zamarev [12] cita vários trabalhos, a partir de 2016, que fazem generalizações de conceitos em grafos estáticos para grafos temporais. Alguns conceitos citados são Clique e *Vertex Cover*.

Uma coisa interessante a se pensar sobre a resolução da coloração para grafos dinâmicos é a necessidade de realizar várias otimizações para vários grafos diferentes, que na verdade são o mesmo grafo dinâmico. Dessa forma, é necessário entender o comportamento do grafo ao longo dos rótulos. O artigo de Bossek, Neumann, Peng e Sudholt [6] se aprofunda nesse tipo de questão focando em algumas heurísticas de busca.

## 4.2 Paralelismo e Coloração de Grafos

Dois artigos foram usados como base para o desenvolvimento dessa subseção. O primeiro foi o artigo de Bogle, Boman, Devine, Rajamanickam e Slota [4], onde os autores propõem uma heurística para reduzir a necessidade de comunicação entre máquinas durante um processamento paralelo de um algoritmo para o problema de coloração de grafos. Seus resultados experimentais mostram que houve aumento de eficiência em grafos que usam diferentes quantidades de GPUs para determinar uma coloração. Já o segundo artigo, escrito por Alanbadi, Powers e Burtscher [1], mostra uma técnica para aumentar o paralelismo do problema de coloração de grafos sem afetar a qualidade da solução devolvida, sendo isso feito através de

*shortcuts* que adiantam a coloração de certas estruturas quando essas são identificadas nos grafos processados.

Um detalhe interessante sobre a relação da coloração de grafos com algoritmos paralelos é que, muitas vezes, usa-se a coloração de grafos como etapa de pré-processamento em algoritmos paralelos. Essa prática acontece, pois, uma coloração de grafos equivale a conjuntos independentes, que no contexto do paralelismo, seriam elementos que podem ser processados de maneira independente dos outros conjuntos, ou seja, de forma paralela [4].

O trabalho de Bogle, Boman, Devine, Rajamanickam e Slota usa essa aplicação à computação científica como motivação, afinal, pode não ser viável realizar essa determinação de conjuntos independentes em uma única máquina. Isso auxilia a aplicação do paralelismo em todas as pontas do processo, desde a etapa do pré-processamento[4].

Geralmente, heurísticas seguem a lógica de que execuções mais demoradas levam a soluções de maior qualidade, enquanto execuções mais rápidas devolvem soluções de menor qualidade. Claro que a escolha por qual heurística será usada depende do caso específico, porém, pode ser um objetivo nosso encontrar uma heurística equilibrada em relação a esses pontos [1].

O trabalho de Alanbadi, Powers e Burtscher [1] busca justamente um equilíbrio em suas heurísticas *shortcuts* que visam aumentar o paralelismo em coloração de grafos paralelos sem perder a qualidade da solução. Em seu trabalho são apresentadas também técnicas de implementação para que a otimização via *shortcuts* seja feita de forma eficiente.

# 5 Considerações Finais

Nesta seção apresentaremos nossas considerações sobre o que foi realizado durante o período abrangido pelo relatório. Começaremos com uma revisão do objetivo geral estabelecido no início do projeto, depois apresentaremos como esse objetivo foi trabalhado ao longo do período. Com essa estrutura, esperamos expor as considerações do aluno sobre cada parte do projeto e como o projeto influenciou em sua formação. Vamos começar relembrando o objetivo geral do projeto, que era ampliar a experiência do aluno em pesquisa nas áreas de projeto de algoritmos e otimização combinatória, através do estudo do problema da coloração mínima de vértices e de algoritmos para tal problema.

Pelo conteúdo desse relatório é possível perceber que o aluno foi exposto a uma grande variedade de técnicas de programação e conceitos da área de projeto de algoritmos. Dentre os comentados de maneira mais extensa nesse relatório podemos destacar as meta-heurísticas, a inaproximabilidade, o conceito de grafos dinâmicos e alguns conceitos de algoritmos paralelos. Ainda é importante destacar que houveram conteúdos estudados durante o período mas que não foram detalhados nas seções desse relatório, isso devido ao formato de relatório onde há

necessidade de uma certa brevidade na escrita. Nesses casos os assuntos são citados na seção que o engloba.

O processo de escolher quais assuntos fariam parte do relatório também toca o objetivo principal do projeto, pois força o aluno a redigir o texto de maneira coerente dentro das informações a serem passadas enquanto segue certas exigências de formato, sendo que a escolha de quais conteúdos seriam aprofundados e como eles seriam abordados é uma parte fundamental desse processo. Também é interessante ressaltar que a grande diversidade de fontes de informação que o aluno teve de consultar para realizar o projeto reforçaram sua familiaridade com diversos tipos de texto encontrados na literatura científica da área, indo de livros didáticos até artigos científicos.

Por último, gostaríamos de destacar que todos os resultados citados nesse relatório (teoremas, lemas, corolários), bem como outros não escritos aqui, foram estudados minuciosamente pelo aluno, o que significa que as demonstrações foram estudadas com atenção e discutidas com a professora orientadora.

# Referências


[1] G. Alabandi, E. Powers, and M. Burtscher. Increasing the parallelism of graph coloring via shortcutting. page 262–275, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368186. doi: 10.1145/3332466.3374519. URL https://doi.org/10.1145/3332466.3374519.

[2] M. Bellare, O. Goldreich, and M. Sudan. Free bits, pcps and non-approximability-towards tight results. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 422–431, 1995. doi: 10.1109/SFCS.1995.492573.

[3] B. Berger and J. Rompel. A better performance guarantee for approximate graph coloring. *Algorithmica*, 5:459–466, June 1990. ISSN 0004-5411. doi: 10.1007/BF01840398. URL https://doi.org/10.1007/BF01840398.

[4] I. Bogle, E. G. Boman, K. D. Devine, S. Rajamanickam, and G. M. Slota. Parallel graph coloring algorithms for distributed gpu environments, 2021.

[5] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 1846289696.

[6] J. Bossek, F. Neumann, P. Peng, and D. Sudholt. Time complexity analysis of randomized search heuristics for the dynamic graph coloring problem, 2021.

[7] M. R. Garey and D. S. Johnson. The complexity of near-optimal graph coloring. *J. ACM*, 23(1):43–49, Jan. 1976. ISSN 0004-5411. doi: 10.1145/321921.321926. URL https://doi.org/10.1145/321921.321926.

[8] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. Springer Publishing Company, Incorporated, 2nd edition, 2010. ISBN 1441916636.

[9] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45(1):19–23, 1993. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(93)90246-6. URL https://www.sciencedirect.com/science/article/pii/0020019093902466.

[10] R. M. Karp. Reducibility Among Combinatorial Problems. 1972. URL https://people.eecs.berkeley.edu/~luca/cs172/karp.pdf.

[11] R. M. R. Lewis. *A Guide to Graph Colouring*. Springer International Publishing, 1st edition, 2016. ISBN 978-3-319-25730-3.

[12] G. B. Mertzios, H. Molter, and V. Zamaraev. Sliding window temporal graph coloring, 2019.

[13] L. Stockmeyer. Planar 3-colorability is polynomial complete. *SIGACT News*, 5(3):19–25, July 1973. ISSN 0163-5700. doi: 10.1145/1008293.1008294. URL https://doi.org/10.1145/1008293.1008294.

[14] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *J. ACM*, 30(4):729–735, Oct. 1983. ISSN 0004-5411. doi: 10.1145/2157.2158. URL https://doi.org/10.1145/2157.2158.