Análise do Algoritmo de Coloração de Grafos

Caio Arruda[1], Kauã Dapper[2]

[1]Faculdades Integradas de Taquara (FACCAT) Taquara – RS – Brasil

caioarruda@sou.faccat.br, kauarafael@sou.faccat.br

Resumo. *Este artigo apresenta um estudo sobre o algoritmo de coloração de grafos, uma análise profunda e detalhada acerca de seu funcionamento, utilidade e complexidade. Inicialmente serão discutidos os conceitos fundamentais do problema, em sequência será apresentado um algoritmo baseado em heurística gulosa. Este artigo visa contribuir para a compreensão desse problema clássico da ciência da computação e reforça sua relevância em diversas aplicações reais.*

1. Coloração de grafos

A coloração de grafos é um dos principais problemas da teoria de grafos, e consiste na atribuição de cores aos vértices, de modo que vértices adjacentes tenham cores distintas. Logo, vértices que são ligados pela mesma aresta (linha), não podem possuir a mesma coloração.

O problema de coloração de grafos surgiu em 1852, quando um aluno da Universidade de Londres, Frederick Guthrie, apresentou ao seu professor, o matemático Augustus De Morgan, uma hipótese que daria origem a esse tão famoso problema de teoria de grafos.

Alfred Kemp, no ano de 1879 publicou um artigo apresentando o Teorema das Quatro Cores, o qual afirma que não são necessárias mais do que quatro cores para colorir as regiões de qualquer mapa. Porém este teorema foi questionado por Percy John Heawood, no ano de 1890. Ele apresentou o Teorema das Cinco Cores, afirmando que ao desenhar um mapa e dividindo ele em regiões, se pode usar até cinco cores diferentes para pintar cada região, desta forma garantindo que duas regiões que fazem fronteira não possuam a mesma cor. E somente no ano de 1976, utilizando um computador, Kenneth Appel e Wolfgang Haken mostraram que o Teorema das Quatro Cores proposto por Alfred Kemp estava correto.

A partir do Teorema das Quatro Cores acaba surgindo outro problema. Abordando como seria possível estabelecer um valor mínimo de cores para colorir um grafo qualquer, problema este conhecido como Problema de Coloração de Grafos.

Esse problema pode ser formulado de duas maneiras. A primeira é como um problema de decisão, cujo objetivo é verificar se um grafo pode ser colorido com “k” cores, respeitando a condição de que vértices adjacentes não compartilhem a mesma cor. Já a segunda formulação, busca determinar o menor valor de “k” possível para colorir um grafo, também conhecido como número cromático.

A Coloração de Grafos é considerado um clássico da Ciência da Computação por pertencer à classe dos problemas NP-Completo, cujo todos os problemas de busca

podem ser resolvidos por ele, porém estes problemas não possuem algoritmos tão eficientes para resolvê-los em tempo polinomial.

Do ponto de vista prático, o estudo do Problema de Coloração de Grafos se mostra extremamente relevante, pois é utilizado em diversas áreas do conhecimento. Entre as principais aplicações estão, escalonamento de tarefas (scheduling), elaboração de calendários acadêmicos e de exames (timetabling), otimização na alocação de registradores em compiladores, organização de plataformas ferroviárias, alocação de frequências em sistemas de comunicação e projetos de redes de comunicação eficientes.

2. Código fonte

2.1. Construção Inicial da Coloração:

```python
def coloracao_grafo_construcao(grafo):
    n = len(grafo)              # Número de vértices
    cores = [-1] * n            # cor de cada vértice (-1 = sem cor)
    cor_atual = 0               # índice da cor atual

    nao_coloridos = set(range(n))  # Conjunto U de vértices ainda não coloridos

    while nao_coloridos:
        bloqueados = set()     # Conjunto B de vértices adjacentes (bloqueados)
        for v in list(nao_coloridos):
            if v not in bloqueados:
                cores[v] = cor_atual
                nao_coloridos.remove(v)
                bloqueados.update(grafo[v])  # Adiciona todos os vizinhos de v a B
        cor_atual += 1

    return cores
```

Figura 1. Imagem da função (def coloracao_grafo_construcao)

Essa função aplica uma heurística gulosa para colorir o grafo pela primeira vez, onde *n* é o número de vértices (nós), e *cores* guarda a cor de cada vértice, sendo que inicialmente todos estão sem cor definida. Começamos usando a cor *0*, com *nao_coloridos* guardando os vértices sem cor. Após as definições das variáveis, o código entra em um loop while que só se encerra quando todos os vértices tiverem cores e retorna uma lista com com as cores atribuídas em cada vértice.

2.2. Busca local

```python
import random

def busca_local(grafo, cores):
    k = max(cores) + 1  # número atual de cores
    while k > 1:
        cor_removida = random.randint(0, k - 1)
        vertices_removidos = [v for v in range(len(cores)) if cores[v] == cor_removida]
        for v in vertices_removidos:
            cores[v] = -1  # Remove a cor

        sucesso = True
        for v in vertices_removidos:
            vizinhos = grafo[v]
            cores_vizinhos = set(cores[u] for u in vizinhos if cores[u] != -1)
            nova_cor = next((c for c in range(k) if c != cor_removida and c not in cores_vizinhos), None)
            if nova_cor is not None:
                cores[v] = nova_cor
            else:
                sucesso = False
                break

        if sucesso:
            k -= 1  # Conseguimos reduzir o número de cores
        else:
            for v in vertices_removidos:
                if cores[v] == -1:
                    cores[v] = cor_removida  # Reverte

    return cores
```

Figura 2. Imagem da função (def busca_local)

Essa função busca reduzir o número de cores utilizadas. Sua ideia principal é remover uma cor aleatória e tentar recolorir os vértices dela com outras cores existentes,

se der certo, a cor é excluída. Essa função funciona com 2 loops for, onde o primeiro remove a cor das vértices (deixando elas sem cor), e o segundo tenta colorir cada vértice *v*, pegando as cores do "vizinho" (para evitar usar a mesma) e procurando uma nova cor. Se achar, a cor é aplicada, se não, o processo é revertido e a cor é restaurada. Por fim, a função retorna uma versão otimizada da coloração inicial.

2.3. Exemplo de uso

```
51  # Exemplo de grafo como lista de adjacência
52  grafo = [
53      [1, 2],    # Vértice 0 é vizinho de 1 e 2
54      [0, 2],    # Vértice 1 é vizinho de 0 e 2
55      [0, 1, 3], # Vértice 2 é vizinho de 0, 1 e 3
56      [2]        # Vértice 3 é vizinho de 2
57  ]
58
59  cores_iniciais = coloracao_grafo_construcao(grafo)
60  print("Cores iniciais:", cores_iniciais)
61
62  cores_otimizadas = busca_local(grafo, cores_iniciais)
63  print("Cores após busca local:", cores_otimizadas)
```

Figura 3. Imagem do exemplo de uso do algoritmo

O grafo possui 4 vértices. Primeiro, aplicamos a heurística gulosa com a primeira função, depois tentamos melhorar o número de cores com a segunda função. Se for possível, o código pode reduzir para menos cores, mas nem sempre é possível dependendo da estrutura do grafo.

3. Complexidade

def coloracao_grafo_construcao(grafo)

- Complexidade Temporal:
  - $O(n^2)$ - Para cada vértice ainda não colorido (n vértices), pode percorrer até todos os vértices (n) na lista *nao_coloridos*.
- Complexidade Espacial:
  - $O(n)$ - Armazena: lista de cores (n), conjunto de não coloridos (n), bloqueados ($\leq n$).

def busca_local(grafo, cores)

- Complexidade Temporal:
  - $O(k \cdot n \cdot d)$ - k = número de cores. | n = número de vértices. | d = grau médio dos vértices (para buscar as cores dos vizinhos). ***Obs.:*** Como há um while que tenta reduzir k repetidamente, pode ser custoso em grafos grandes.
- Complexidade Espacial:
  - $O(n)$ - Lista de cores, vizinhos, conjuntos temporários.

4. Referências

REGO, Marcelo Ferreira. Algoritmos para o problema de coloração de grafos. 2011. 22 f. Trabalho acadêmico (PCC104 – Projeto e Análise de Algoritmos) – Departamento

de Computação, UFOP, Ouro Preto, 2011. Disponível em: http://www.decom.ufop.br/menotti/paa111/files/PCC104-111-ars-11.1-MarceloFerreiraRego.pdf. Acesso em: 3 jul. 2025.

FIGUEIREDO, Jorge César Abrantes de. Coloração. 2005. [S.l.]: Departamento de Sistemas e Computação, Universidade Federal de Campina Grande, 10 out. 2005. Disponível em: http://www.dsc.ufcg.edu.br/~abrantes/CursosAnteriores/TG051/coloracao.pdf. Acesso em: 3 jul. 2025.

LIMA, Alane Marie de; HERNANDES, Fábio (orientador). Uma aplicação para o problema de coloração de grafos. In: XIX Semana de Iniciação Científica, 2014, Guarapuava, PR. Anais... Guarapuava: UNICENTRO, Departamento de Ciência da Computação, 25 e 26 set. 2014. ISSN 2238-7358. Disponível em: https://anais.unicentro.br/proic/pdf/xixv2n1/285.pdf. Acesso em: 4 jul. 2025.

RANGEL, Socorro; OLIVEIRA, Valeriano A. de; ARAUJO, Silvio A. de. Coloração de vértices. In: Departamento de Matemática Aplicada, UNESP/IBILCE. Teoria dos Grafos: notas de aula. São José do Rio Preto: IBILCE–UNESP, 2002–2013. Disponível em: https://www.ibilce.unesp.br/Home/Departamentos/MatematicaAplicada/docentes/socorro/coloracaovertices.pdf. Acesso em: 4 jul. 2025.