Coloração de Grafos: Um Estudo de Caso de Problema NP-Completo

Arthur Rabelo de Carvalho[1], Allyson Kawa Santos da Silva[1], Raimundo Nonato Gomes Neto[1]
[1]Departamento de Computação,
Universidade Federal do Piauí – Teresina, PI, Brasil

rabeloarthur@ufpi.edu.br, allyson.silva@ufpi.edu.br, netoxd66@gmail.com

Resumo: Problemas NP-completos são desafios centrais em ciência da computação, sendo a Coloração de Grafos um dos mais clássicos e com diversas aplicações práticas. Este trabalho apresenta a formulação do problema, um cenário de aplicação real e uma implementação exata baseada em força bruta para servir de linha de base (baseline) para futuras comparações com heurísticas. A coloração consiste em atribuir cores aos vértices de um grafo, garantindo que vértices adjacentes não compartilhem a mesma cor e minimizando o número total de cores utilizadas. Também é explorada a execução real do algoritmo sobre uma instância com 18 disciplinas, discutindo seus resultados e desempenho computacional.
Palavras-chave: Coloração de grafos, NP-completo, algoritmos, força bruta, Python.

Abstract: NP-complete problems are central challenges in computer science, with Graph Coloring being one of the most classic and practically applicable. This paper presents the problem's formulation, a real-world application scenario, and a brute-force implementation as a baseline for future heuristic comparisons. Graph coloring involves assigning colors to graph vertices so that adjacent vertices do not share the same color, while minimizing the total number of colors used.
Keywords: Graph coloring, NP-complete, algorithms, brute-force, Python.

1. Introdução

Problemas NP-completos constituem uma classe de problemas computacionalmente difíceis, cuja solução exata exige tempo exponencial em função do tamanho da entrada. A coloração de grafos se destaca nesse conjunto por suas aplicações em áreas como alocação de horários, frequências de rádio e coloração de mapas. Neste trabalho, o problema é abordado com foco na compreensão de sua formulação, complexidade e implementação inicial.

2. Formulação do Problema

Dado um grafo G = (V, E), o problema da coloração consiste em atribuir uma cor a cada vértice v ∈ V de modo que nenhum par de vértices adjacentes compartilhe a mesma cor. O objetivo é minimizar o número de cores utilizadas — o chamado número cromático. Este problema é NP-completo para grafos gerais, conforme demonstrado por Karp (1972).

A coloração de grafos tem aplicações diretas em escalonamento de tarefas, como a alocação de disciplinas em horários escolares, onde cada disciplina (vértice) deve ser atribuída a um horário (cor) sem que duas disciplinas conflitantes ocorram ao mesmo tempo (arestas). Esse modelo se encaixa diretamente no caso prático tratado neste trabalho, no qual buscamos organizar a grade de um curso universitário garantindo que disciplinas do mesmo período ou com o mesmo professor não sejam alocadas em horários coincidentes.

3. Aplicação Prática

Na organização de horários de um curso universitário, é necessário evitar conflitos para disciplinas do mesmo período ou ministradas pelo mesmo professor. Cada disciplina é modelada como um vértice, e uma aresta é criada entre duas disciplinas se elas não puderem ocorrer simultaneamente. O problema da coloração de grafos permite atribuir horários distintos às disciplinas adjacentes, resolvendo o problema de escalonamento com precisão formal.

4. Solução Ótima por Força Bruta

A abordagem exata adotada foi baseada em backtracking, testando recursivamente atribuições de cores e retrocedendo sempre que uma violação de restrição é encontrada. O algoritmo verifica, para cada vértice, todas as cores possíveis e continua o processo caso a atribuição seja válida.

O funcionamento do algoritmo pode ser descrito por três etapas principais:

- Construção do grafo com arestas entre disciplinas que compartilham professor ou período.
- Definição de um conjunto de cores (horários reais) que serão testados.
- Execução do algoritmo de *backtracking*, que tenta atribuir o menor número possível de cores sem violar os conflitos definidos pelas arestas.

A complexidade do algoritmo é exponencial: O(k^n), onde n é o número de vértices e k o número de cores consideradas. Apesar de ineficiente para grandes grafos, essa solução é útil como referência para medir a qualidade de heurísticas.

5. Implementação em Python

```python
def eh_atribuicao_valida(vertice, cor, grafo, atribuicoes_atuais):
    for vizinho in grafo[vertice]:
        if atribuicoes_atuais.get(vizinho) == cor:
            return False
    return True

def resolver_coloracao_backtracking(grafo, k_cores, vertices, indice_vertice, atribuicoes):
    if indice_vertice == len(vertices):
        return True
    vertice_atual = vertices[indice_vertice]
    for cor_candidata in range(1, k_cores + 1):
        if eh_atribuicao_valida(vertice_atual, cor_candidata, grafo, atribuicoes):
            atribuicoes[vertice_atual] = cor_candidata
            if resolver_coloracao_backtracking(
                grafo, k_cores, vertices, indice_vertice + 1, atribuicoes
            ):
                return True
            del atribuicoes[vertice_atual]
    return False
```

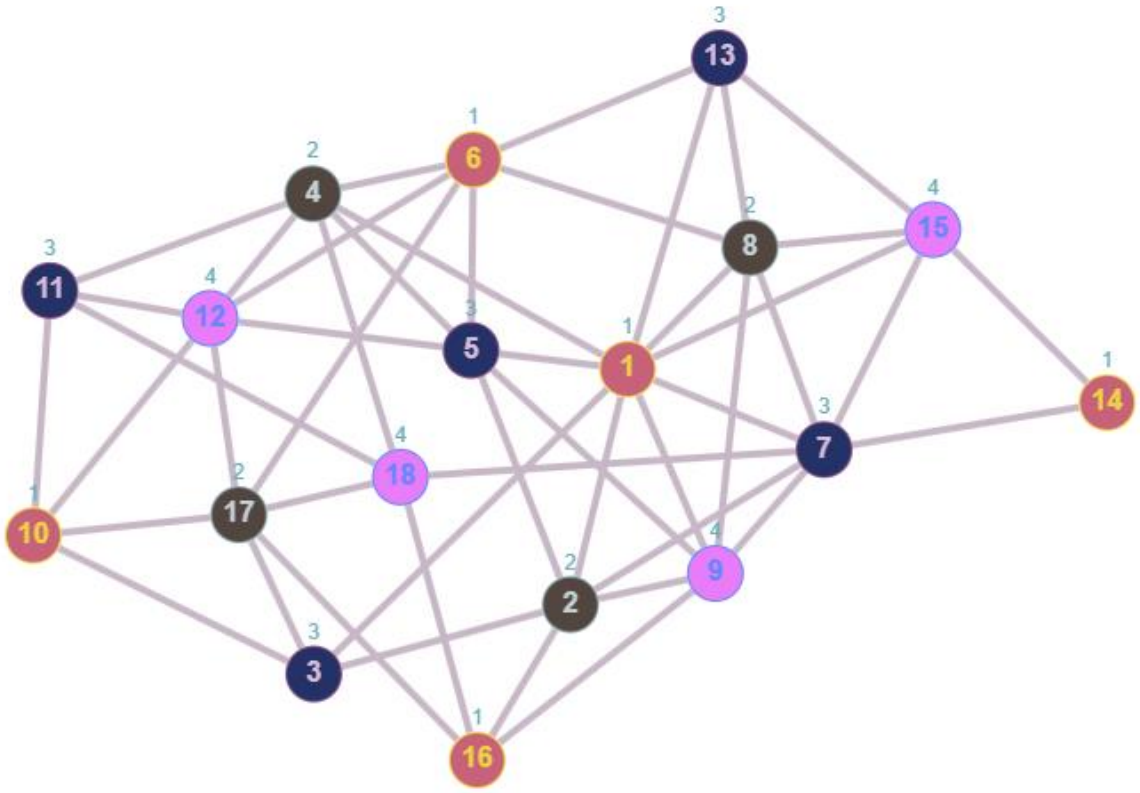
7. Execução em instância com 18 disciplinas

Para testar a viabilidade do algoritmo, foi utilizada uma entrada com 18 disciplinas, respeitando as regras de conflitos por período e professor. O grafo gerado a partir desses dados apresentou um total de 33 arestas, representando os conflitos entre as disciplinas.

A execução identificou corretamente todos os conflitos e encontrou uma coloração ótima utilizando 3 horários diferentes, sem sobreposição entre disciplinas conflitantes. A complexidade da execução foi influenciada diretamente pela densidade do grafo: como havia diversos conflitos cruzados, o algoritmo precisou explorar múltiplas combinações de cores até encontrar a distribuição mínima possível.

O algoritmo percorreu todas as combinações possíveis até encontrar a menor alocação viável (número cromático = 3), respeitando os horários disponíveis de segunda a sábado em quatro blocos diários.

Grafo de Conflitos para 18 Disciplinas



Fonte: Graph Online

8. Ambiente de Execução

A implementação foi realizada com os seguintes recursos computacionais:

- Linguagem: Python 3.11
- Sistema Operacional: Windows 11
- Processador: Intel Core i5 2,90 GHz x 6
- Memória RAM: 16 GB

9. Conclusão

A coloração de grafos, embora simples em sua definição, é um problema computacionalmente difícil. A implementação de uma solução exata baseada em backtracking oferece um ponto de partida sólido para estudos de heurísticas futuras.

O próximo passo será propor e avaliar métodos aproximados mais eficientes, capazes de lidar com instâncias maiores.

10. Referências

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. *Algoritmos: teoria e prática.* 3. ed. Rio de Janeiro: Elsevier, 2013.