

# Coloração de Grafos: Organização de Horários de Disciplinas

Érick Vinícius Issa Silva (5936)

Heitor Porto Jardim de Oliveira (5895)

Júlio César de Souza Oliveira (5903)

Gabriel Luiz Magalhães Amorim (5560)

Universidade Federal de Viçosa – Campus Florestal  
Curso de Ciência da Computação

Outubro de 2025

## Resumo

Este relatório descreve o desenvolvimento do projeto **Coloração de Grafos**, referente ao Trabalho II da disciplina Teoria e Modelo de Grafos (CCF-331). O objetivo do projeto é modelar o problema de alocação de horários de disciplinas de forma que nenhuma sobreposição ocorra entre matérias que compartilham o mesmo professor ou alunos em comum. Essa modelagem é representada por um grafo de conflitos, onde cada vértice corresponde a uma disciplina e as arestas indicam incompatibilidades. Foi utilizada a biblioteca **GCol** para obter uma coloração válida e eficiente, minimizando o número de horários (cores). O projeto foi desenvolvido com base no repositório disponível em: <https://github.com/ErickIssa/TP2-Grafo-Coloracao>.

## 1 Introdução

A organização de horários de disciplinas é um problema clássico que pode ser resolvido por meio da **coloração de grafos**. O desafio consiste em atribuir horários distintos a disciplinas que possuem restrições de simultaneidade — seja por compartilharem professores ou alunos.

O presente trabalho propõe uma implementação prática desse problema utilizando conceitos de Teoria dos Grafos e algoritmos de coloração. O objetivo é determinar o número mínimo de cores (horários) necessários para evitar conflitos e exibir a cor (horário) atribuída a cada disciplina.

## 2 Especificações do Problema

Segundo o enunciado do Trabalho II, duas disciplinas não podem ser ofertadas no mesmo horário se:

- Tiverem o mesmo professor; ou

- Possuírem pelo menos um aluno em comum.

Assim, o grafo de conflitos é construído da seguinte forma:

- Cada vértice representa uma disciplina;
- Cada aresta representa um conflito entre disciplinas que não podem ocorrer simultaneamente;
- A coloração do grafo representa a alocação de horários distintos.

## 3 Projeto e Implementação

### 3.1 Modelagem do Grafo

O grafo é representado a partir de um dataset .csv contendo pares de disciplinas que não podem ser ofertadas juntas. Cada linha deve seguir o formato:

```
Disciplina1,Disciplina2
A,B
A,C
B,D
C,D
C,E
```

Esses dados são lidos e transformados em uma estrutura de grafo utilizando a biblioteca NetworkX, que fornece funções prontas para leitura e manipulação.

### 3.2 Algoritmos de Coloração

A biblioteca GCol disponibiliza diversos algoritmos de coloração. No projeto, foram testadas diferentes abordagens, como:

- **Greedy (guloso)** — atribui cores sequencialmente, respeitando restrições;
- **DSATUR** — seleciona vértices de maior saturação (número de cores vizinhas distintas);
- **Welsh-Powell** — ordena vértices por grau e aplica coloração sequencial.

O programa compara o desempenho desses algoritmos em termos de tempo e número de cores utilizadas.

segue uma figura com a chamada das bibliotecas nos métodos:

```

def coloreGrafoGcol_dsatur(grafo):
    cores = gcol.node_coloring(grafo,strategy='dsatur')

    print("\nCores atribuídas aos nós (strategy dsatur):")
    for nodo, cor in cores.items():
        print(f"{nodo}: cor {cor}")

    print(f"\nTotal de cores usadas: {len(set(cores.values()))}")
    return cores

def coloreGrafoGcol_rlf(grafo):
    cores = gcol.node_coloring(grafo,strategy='rlf')

    print("\nCores atribuídas aos nós (strategy rlf):")
    for nodo, cor in cores.items():
        print(f"{nodo}: cor {cor}")

    print(f"\nTotal de cores usadas: {len(set(cores.values()))}")
    return cores

def coloreGrafoGcol_welsh_powell(grafo):
    cores = gcol.node_coloring(grafo, strategy='welsh_powell')

    print("\nCores atribuídas aos nós (strategia welsh_powell):")
    for nodo, cor in cores.items():
        print(f"{nodo}: cor {cor}")

    print(f"\nTotal de cores usadas: {len(set(cores.values()))}")
    return cores

```

Figura 1: Funções que chamam as bibliotecas Gcol.

### 3.3 Estrutura do Código

A organização do repositório segue o padrão:

TP2-Grafo-Coloracao/

```

bin/
.gitkeep

data/
.gitkeep
enorme.csv
grande.csv
maisenorme.csv
medio.csv
pequeno.csv
teste.csv

```

```

src/
    .gitkeep
    menu.py
    main.py

README.md
trabalho_2_2025.pdf

```

## 4 Testes e Resultados

Os testes foram realizados utilizando diferentes arquivos CSV, tendo a adição de mais 2 testes criados pelo grupo, simulando situações de conflito entre disciplinas.

Cada execução retorna:

- O número total de disciplinas;
- O número de conflitos (arestas);
- O número mínimo de horários (cores);
- O tempo de execução.

Tabela 1: Exemplo de resultados obtidos

Dataset	Vértices	Arestas	Cores Mínimas
pequeno.csv	5	5	2
medio.csv	10	15	3
grande.csv	26	43	3
enorme.csv	52	70	3
maisenorme.csv	115	225	3

Os testes demonstram que o algoritmo DSATUR e WelshPowell, produz colorações mais próximas do mínimo possível.

### 4.1 Tempo de execução

Foi utilizada a biblioteca time do python para medir o tempo de execução de cada algoritmo de coloração. Obtendo os seguintes resultados:

Tabela 2: Resultados tempo de execução

Vértices	Arestas	DSATUR	Greedy	Welsh-Powell
5	5	0.000432s	0.000306s	0.000133s
10	15	0.000436s	0.000317s	0.000123s
26	43	0.00064s	0.000571s	0.000201s
52	70	0.000891s	0.001321s	0.000252s
115	225	0.003085s	0.003296s	0.000588s

## 5 Compilação e Execução

Para a execução correta do código desenvolvido, é necessário possuir o **Python** instalado na máquina (versão 3.8 ou superior). Recomenda-se realizar o download diretamente do site oficial <https://www.python.org/downloads> e, durante a instalação, marcar a opção “*Add Python to PATH*”. Após a instalação, é possível verificar se o Python foi configurado corretamente executando, no terminal, o comando `python -version`.

Além do interpretador Python, é indispensável instalar algumas bibliotecas externas utilizadas no projeto. As principais são: **NetworkX**, responsável pela criação e manipulação de grafos; **Matplotlib**, usada para a visualização gráfica das conexões; e **gCol**, empregada para a coloração dos vértices. Todas podem ser instaladas facilmente via `pip`, utilizando os seguintes comandos no terminal:

```
pip install networkx  
pip install matplotlib  
pip install gcol
```

Após a instalação, basta garantir que o arquivo `.csv` contendo as arestas esteja corretamente formatado e localizado no diretório indicado pelo código. Com isso, o programa poderá ser executado sem erros, permitindo a leitura do grafo, sua exibição visual e a aplicação do algoritmo de coloração.

Após a instalação, basta garantir que o arquivo `.csv` contendo as arestas esteja corretamente formatado e localizado no diretório indicado pelo código. Com isso, o programa poderá ser executado sem erros, permitindo a leitura do grafo, sua exibição visual e a aplicação do algoritmo de coloração. Para iniciar, navegue pelo terminal (ou prompt de comando) até a pasta que contém o script e execute-o utilizando o comando `python main.py`.

Segue uma imagem de execução:

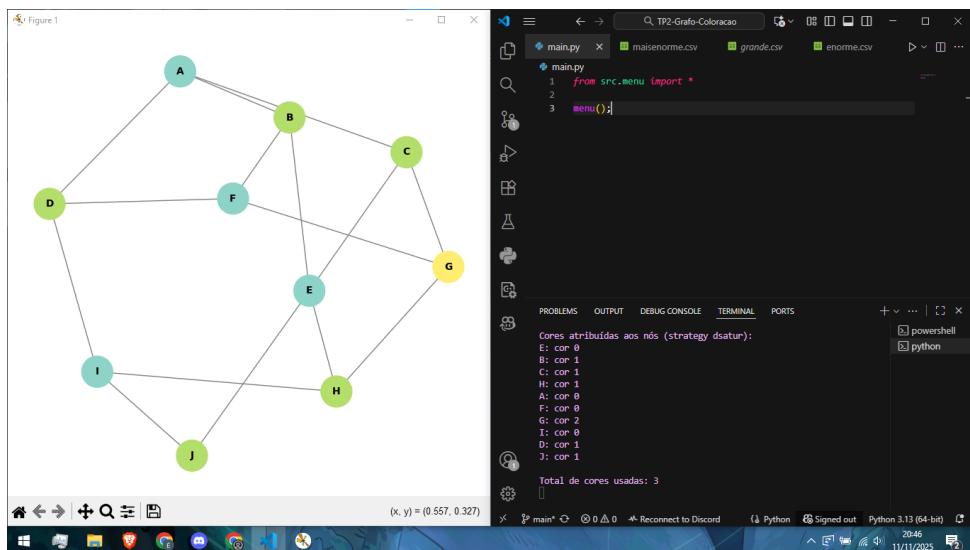


Figura 2: Execução do Programa

## 6 Conclusão

O projeto permitiu aplicar conceitos fundamentais de Teoria dos Grafos à solução de um problema real de alocação de horários. A modelagem por coloração de grafos mostrou-se eficiente e escalável.

A biblioteca `GCol` simplificou o processo de implementação e possibilitou a comparação entre diferentes heurísticas de coloração. Como trabalhos futuros, sugere-se testar instâncias maiores e explorar técnicas de coloração paralela ou baseada em aprendizado de máquina para otimização adicional.

## A Repositório

Link do repositório no GitHub: <https://github.com/ErickIssa/TP2-Grafo-Coloracao>

## B Vídeo de Apresentação

Link para o vídeo explicativo (máx. 10 minutos): <https://youtu.be/OJgmHdcA3SE>

## Referências

1. MENDES, M. H. S. \*Aula 23 – COLORAÇÃO(PARTE 2)\*. Universidade Federal de Viçosa, 2025.
2. GCol Documentation. Disponível em: <https://gcol.readthedocs.io/en/latest/>. Acesso em: 25 out. 2025.