Implementação do segundo trabalho de grafos

Gabiel V. T. Santos - 5921[1],
Matheus T. P. Coelho - 5909[2],
Vítor A. de S. Carvalho - 5906 [3]

[1]Universidade Federal de Viçosa - Florestal (UFV - Florestal)
35.690-000 – Florestal – MG – Brazil

gabriel.vitor.santos@ufv.br, matheus.coelho@ufv.br, vitor.souza.carvalho@ufv.br

Resumo. *O projeto tem como objetivo resolver o problema de alocação de horários de disciplinas utilizando o conceito de coloração de grafos, onde cada vértice representa uma disciplina e cada aresta indica um conflito causado por compartilhamento de professor ou alunos. O programa lê um arquivo .csv contendo pares de disciplinas em conflito, constrói o grafo com a biblioteca NetworkX e aplica algoritmos de coloração fornecidos pela GCol, buscando minimizar o número total de cores (horários) necessárias para evitar sobreposições. O sistema exibe o número mínimo de cores utilizadas, a cor atribuída a cada disciplina e o tempo aproximado de execução, além de gerar visualizações gráficas do grafo colorido.*

1. Introdução

A organização eficiente dos horários de disciplinas em uma universidade é um desafio recorrente, pois envolve a necessidade de evitar conflitos entre matérias que compartilham professores ou alunos. Esse problema pode ser formalmente representado por meio da teoria dos grafos , em que cada disciplina é modelada como um vértice e os conflitos entre elas são representados por arestas. A partir dessa modelagem, o objetivo é aplicar técnicas de coloração de grafos para atribuir um horário distinto (cor) a cada disciplina, garantindo que não ocorram sobreposições e que o número total de horários seja o menor possível. Neste contexto, o presente trabalho propõe a implementação de uma solução computacional em Python, utilizando as bibliotecas NetworkX e GCol , para construir o grafo de conflitos, realizar a coloração e apresentar os resultados de forma visual e analítica.

2. Metodologia

A metodologia adotada para o desenvolvimento deste trabalho baseou-se na aplicação prática dos conhecimentos adquiridos ao longo das aulas de Teoria dos Grafos , que serviram como fundamento para a compreensão do problema de coloração e sua modelagem computacional. Foram revisados diversas vezes os materiais e slides disponibilizados em sala , com o objetivo de reforçar os conceitos teóricos necessários para a implementação correta do grafo e dos algoritmos de coloração. Além disso, pesquisas complementares na internet tiveram papel essencial para compreender o funcionamento das bibliotecas NetworkX e GCol , bem como para esclarecer dúvidas sobre a manipulação de grafos em Python e otimização da estrutura do código. A partir dessa base teórica e prática, o projeto

foi estruturado em etapas: leitura dos datasets, construção do grafo de conflitos, aplicação dos algoritmos de coloração e apresentação dos resultados, garantindo uma abordagem sistemática e fundamentada para alcançar a solução proposta.

3. Execução do Problema

Para executar o programa desenvolvido em `Python`, é necessário que o ambiente esteja devidamente configurado com as dependências utilizadas no projeto. A seguir, são apresentadas as etapas e requisitos para execução em sistemas **Windows** e **Linux**.

3.1. Requisitos Gerais

- Ter o **Python 3.8** ou superior instalado no sistema.
- Instalar as bibliotecas utilizadas:
  - `pandas` — leitura e manipulação dos arquivos `.csv`;
  - `networkx` — criação e manipulação de grafos;
  - `gcol` — coloração de grafos e execução dos algoritmos de alocação;
  - `matplotlib` — geração das visualizações gráficas;
  - `time` — medição do tempo de execução.

3.2. Execução no Windows

1. Instalar o Python a partir do site oficial: `https://www.python.org/downloads/`.
2. Abrir o terminal `cmd` ou o `PowerShell`.
3. Acessar o diretório onde o projeto está localizado:
   ```
   cd caminho\para\o\projeto
   ```
4. Instalar as dependências necessárias:
   ```
   pip install pandas networkx gcol matplotlib
   ```
5. Executar o programa principal:
   ```
   python main.py
   ```

3.3. Execução no Linux

1. Verificar se o Python já está instalado utilizando:
   ```
   python3 --version
   ```
   Caso não esteja, instalar com:
   ```
   sudo apt install python3 python3-pip
   ```
2. Acessar o diretório do projeto:
   ```
   cd caminho/para/o/projeto
   ```
3. Instalar as dependências necessárias:
   ```
   pip3 install pandas networkx gcol matplotlib
   ```
4. Executar o programa principal:
   ```
   python3 main.py
   ```

4. Descrição do Código

O código desenvolvido foi estruturado em três módulos principais: `preencherGrafo.py`, `leituraArquivos.py` e `main.py`. Essa divisão modular foi adotada com o objetivo de garantir clareza, organização e facilidade de manutenção do sistema, permitindo que cada etapa do processo de coloração de grafos seja tratada de forma independente, desde a leitura dos dados até a visualização final dos resultados.

4.1. Módulo `preencherGrafo.py`

Este módulo é responsável pela leitura dos dados do dataset e pela construção do grafo de conflitos utilizando a biblioteca `NetworkX`. Ele contém duas funções principais:

- **`CriarMontarGrafo(datasetEscolhido)`**: realiza a leitura do arquivo `.csv` correspondente ao dataset selecionado, utilizando a biblioteca `pandas`. Cada linha do arquivo representa uma relação de conflito entre duas disciplinas, que são adicionadas como uma aresta no grafo. Dessa forma, o grafo é composto por vértices (disciplinas) e arestas (conflitos).
- **`imprimirDadosGrafo(Grafo)`**: exibe informações descritivas sobre o grafo, como o número de vértices, número de arestas, lista de disciplinas e suas respectivas conexões. Essa função é útil para verificar a consistência dos dados lidos e validar a estrutura do grafo antes da coloração.

4.2. Módulo `leituraArquivos.py`

O módulo `leituraArquivos.py` é responsável por gerenciar a interação com o usuário para a seleção do dataset que será analisado. O programa apresenta um menu interativo no terminal, oferecendo três opções de arquivos: `pequeno.csv`, `medio.csv` e `grande.csv`. Após a escolha, o módulo chama a função `CriarMontarGrafo()` do módulo anterior para construir o grafo correspondente e, em seguida, exibe suas informações iniciais. Essa abordagem torna o programa mais dinâmico e flexível, permitindo a análise de diferentes conjuntos de dados sem necessidade de alterações no código.

4.3. Módulo `main.py`

O arquivo principal `main.py` coordena a execução geral do programa, realizando a aplicação dos algoritmos de coloração e a geração dos resultados. Após a leitura e construção do grafo, o programa segue os seguintes passos:

1. **Seleção do algoritmo de coloração:** o sistema solicita ao usuário que escolha, por meio de um menu interativo, qual algoritmo da biblioteca `GCol` será utilizado. Estão disponíveis os métodos `DSATUR`, `Welsh-Powell`, `Random` e `RLF`, cada um com características específicas de desempenho e qualidade da coloração.
2. **Conversão e coloração do grafo:** o grafo gerado pelo `NetworkX` é processado pela biblioteca `GCol`, que aplica a estratégia escolhida através da função `node_coloring()`, garantindo que disciplinas em conflito não recebam a mesma cor (horário).
3. **Medição do tempo de execução:** o tempo total de processamento é calculado utilizando a biblioteca `time`, possibilitando avaliar o desempenho de cada algoritmo em diferentes conjuntos de dados.

4. **Apresentação dos resultados:** o programa exibe no terminal o número mínimo de cores utilizadas, a cor atribuída a cada disciplina e o tempo de execução aproximado. As cores são associadas a nomes legíveis (por exemplo, "Azul Escuro", "Laranja" etc.) para facilitar a compreensão dos resultados.
5. **Visualização gráfica:** o grafo colorido é exibido com a biblioteca `matplotlib`, utilizando três diferentes layouts (`spring_layout`, `coloring_layout` e `multipartite_layout`), permitindo uma análise visual da coloração e dos conflitos existentes.

4.4. Escolha do Algoritmo de Coloração

Nesta etapa, o programa solicita ao usuário que selecione o algoritmo de coloração de grafos a ser aplicado durante a execução. A interação é realizada diretamente no terminal, conforme mostrado abaixo:

```
Escolha o algoritmo de coloração que deseja usar:
==========================================
1 - DSATUR (boa qualidade, mais comum)
2 - Welsh-Powell (rápido)
3 - Random (aleatório)
4 - RLF (Recursive Largest First)
==========================================

Digite o número do algoritmo desejado:
```

O código associa a opção selecionada à variável `estrategia`, que é então passada como parâmetro para o método de coloração da biblioteca `GCol`. Caso o usuário insira um valor inválido, o algoritmo `DSATUR` é escolhido por padrão, assegurando a continuidade da execução. Os algoritmos disponíveis têm as seguintes características:

- **DSATUR (Degree of Saturation)** — busca minimizar o número total de cores utilizadas, priorizando vértices com maior saturação (maior número de cores distintas em vizinhos). É o método mais equilibrado entre desempenho e qualidade da solução.
- **Welsh-Powell** — ordena os vértices em função do grau e aplica uma coloração sequencial. É um método rápido, indicado para testes com grandes grafos.
- **Random** — realiza a coloração de forma aleatória, sendo útil para análises comparativas de desempenho e variação de resultados.
- **RLF (Recursive Largest First)** — heurística que seleciona, a cada iteração, o maior conjunto independente de vértices, atribuindo-lhes a mesma cor antes de continuar o processo.

Essa etapa confere ao sistema flexibilidade e capacidade de comparação entre diferentes estratégias de coloração, permitindo avaliar tanto o desempenho computacional quanto a eficiência na redução do número de cores.

4.5. Estrutura Geral do Programa

O fluxo de execução geral do programa pode ser descrito conforme os seguintes passos:

1. O usuário executa o arquivo `main.py`.

2. O sistema solicita a escolha do dataset a ser analisado.
3. O grafo de conflitos é construído a partir dos dados do arquivo `.csv`.
4. O usuário escolhe o algoritmo de coloração a ser aplicado.
5. O grafo é colorido conforme a estratégia selecionada.
6. Os resultados são exibidos no terminal e apresentados graficamente.

Essa estrutura modular, aliada ao uso das bibliotecas `pandas`, `NetworkX`, `GCol` e `matplotlib`, garante clareza, eficiência e precisão na análise dos resultados, além de proporcionar uma aplicação prática dos conceitos teóricos de coloração de grafos estudados em sala de aula.

5. Resultados e Análises

Nesta seção, são apresentados os resultados obtidos para três tamanhos distintos de datasets: pequeno, médio e grande. Cada conjunto contém três imagens que ilustram diferentes etapas do processamento e análise dos grafos.

5.1. Dataset Pequeno

```
===== RESULTADOS =====
Algoritmo utilizado: DSATUR
Número mínimo de cores utilizadas (Horários): 2
Cor atribuída a cada disciplina:
  C: Azul Escuro
  A: Azul Claro
  D: Azul Claro
  B: Azul Escuro
  E: Azul Claro
Tempo de execução aproximado: 0.000515 segundos
======================
```

Figure 1. Resultados Dataset Pequeno.

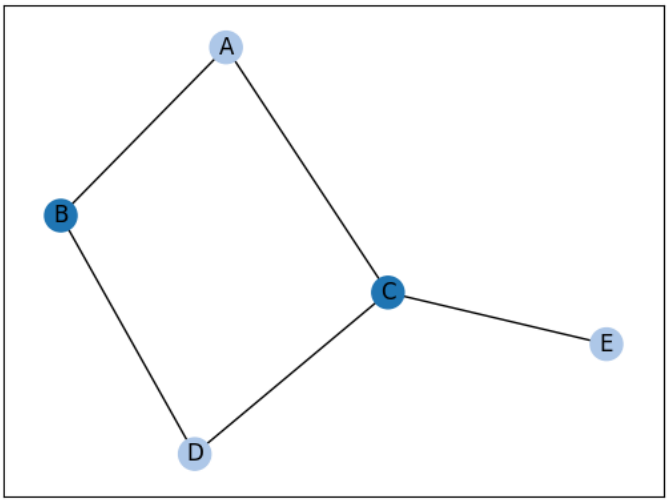

Figure 2. Visualização inicial do grafo - Dataset Pequeno.

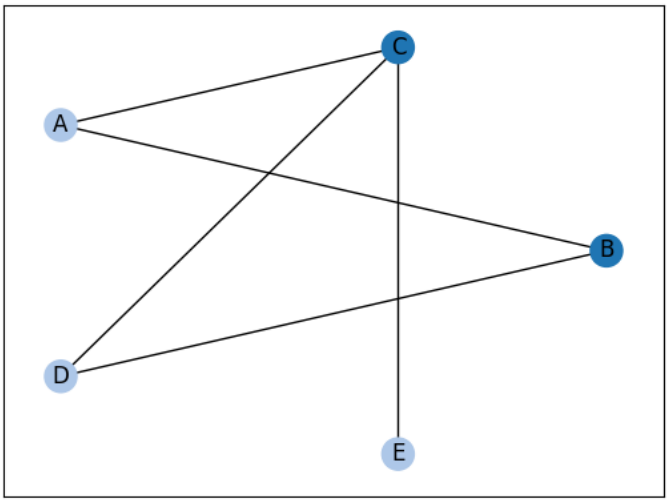

Figure 3. Processamento intermediário - Dataset Pequeno.

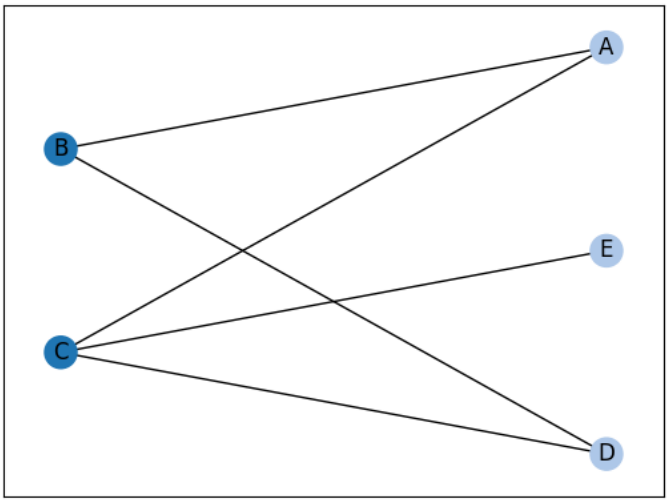

Figure 4. Resultado final da análise - Dataset Pequeno.

5.2. Dataset Médio

```
===== RESULTADOS =====
Algoritmo utilizado: DSATUR
Número mínimo de cores utilizadas (Horários): 3
Cor atribuída a cada disciplina:
  E: Azul Escuro
  B: Azul Claro
  C: Azul Claro
  H: Azul Claro
  A: Azul Escuro
  F: Azul Escuro
  G: Laranja Escuro
  I: Azul Escuro
  D: Azul Claro
  J: Azul Claro
Tempo de execução aproximado: 0.000552 segundos
======================
```

Figure 5. Resultados Dataset Médio.

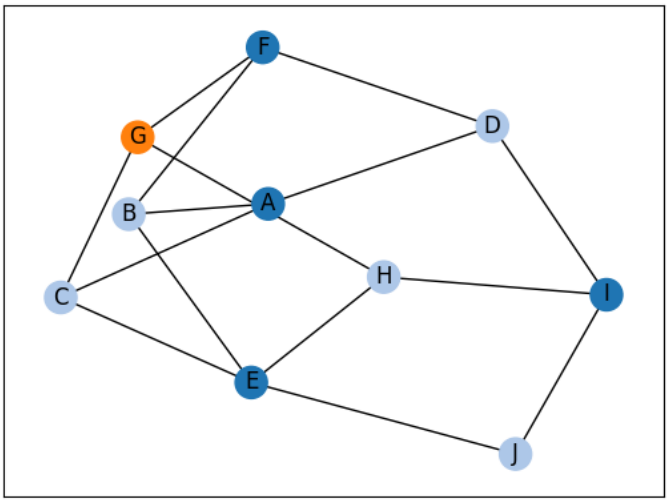

Figure 6. Visualização inicial do grafo - Dataset Médio.

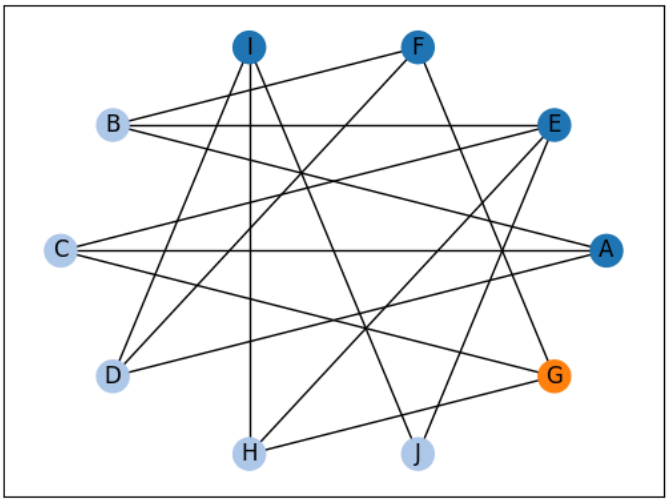

Figure 7. Processamento intermediário - Dataset Médio.

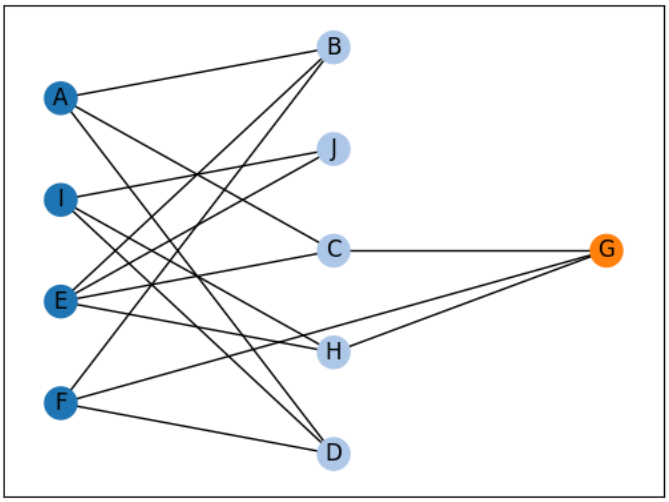

Figure 8. Resultado final da análise - Dataset Médio.

5.3. Dataset Grande

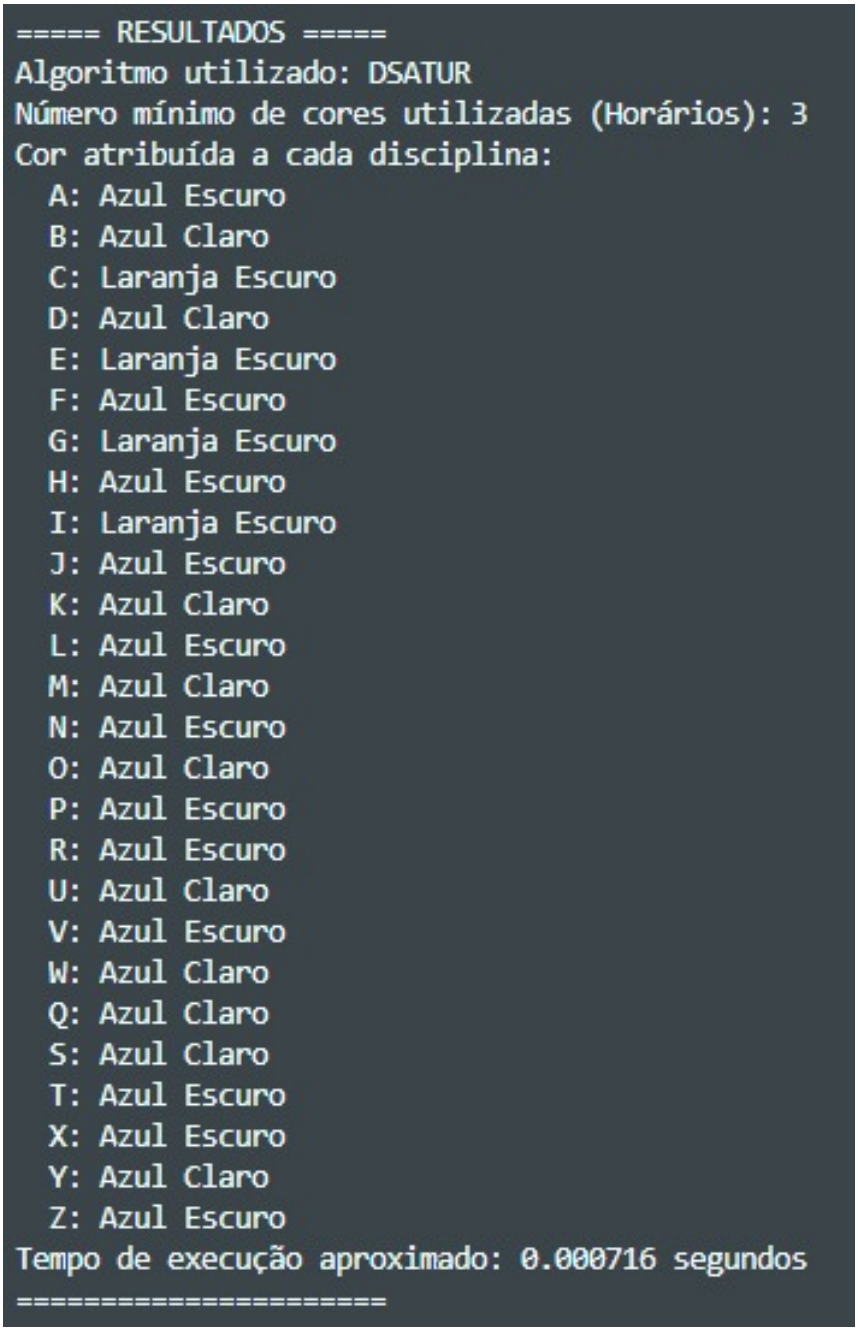

Figure 9. Resultados Dataset Grande.

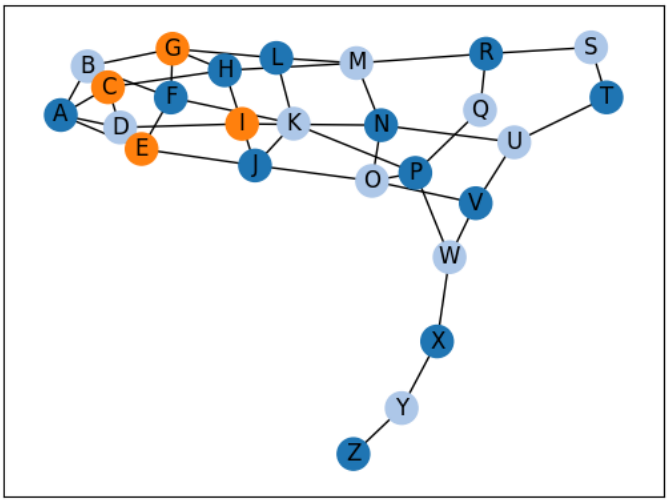

Figure 10. Visualização inicial do grafo - Dataset Grande.

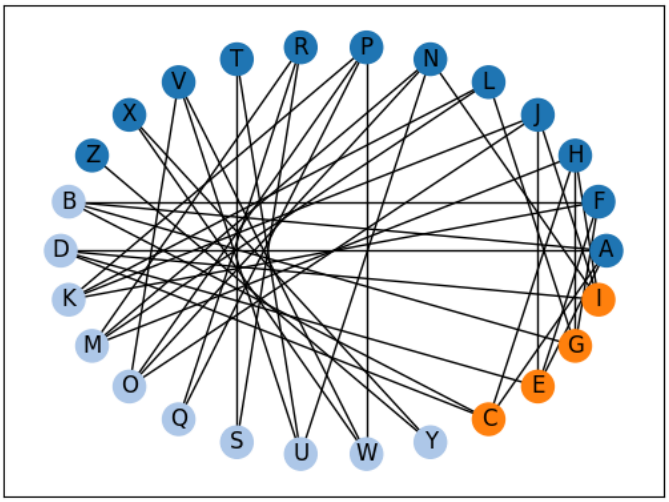

Figure 11. Processamento intermediário - Dataset Grande.

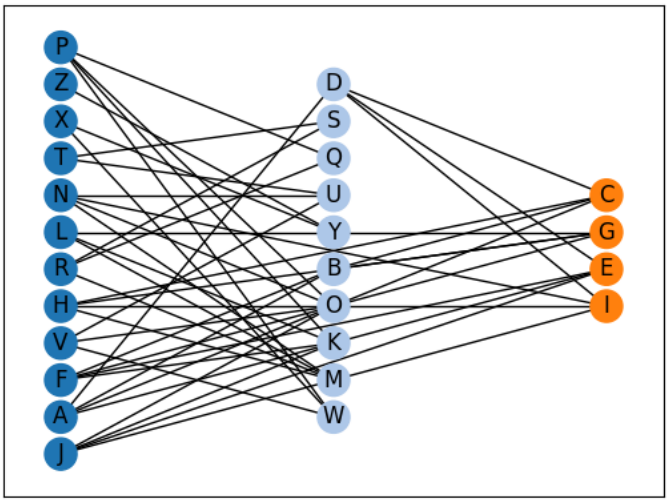

Figure 12. Resultado final da análise - Dataset Grande.

5.4. Dataset de Teste 1

```
===== RESULTADOS =====
Algoritmo utilizado: DSATUR
Número mínimo de cores utilizadas (Horários): 5
Cor atribuída a cada disciplina:
  B: Azul Escuro
  C: Azul Claro
  D: Laranja Escuro
  E: Laranja Claro
  A: Verde Escuro
  F: Azul Claro
  H: Azul Escuro
  G: Azul Escuro
  I: Azul Claro
  J: Laranja Escuro
  K: Azul Escuro
  L: Azul Claro
  M: Azul Escuro
  Q: Azul Claro
  R: Azul Escuro
  U: Azul Escuro
  S: Azul Claro
  T: Laranja Escuro
  P: Azul Escuro
  O: Laranja Escuro
  N: Azul Claro
  V: Azul Claro
  Y: Azul Claro
  X: Azul Escuro
  W: Laranja Escuro
  Z: Azul Escuro
Tempo de execução aproximado: 0.000746 segundos
======================
```

Figure 13. Resultados Dataset 5 cores.

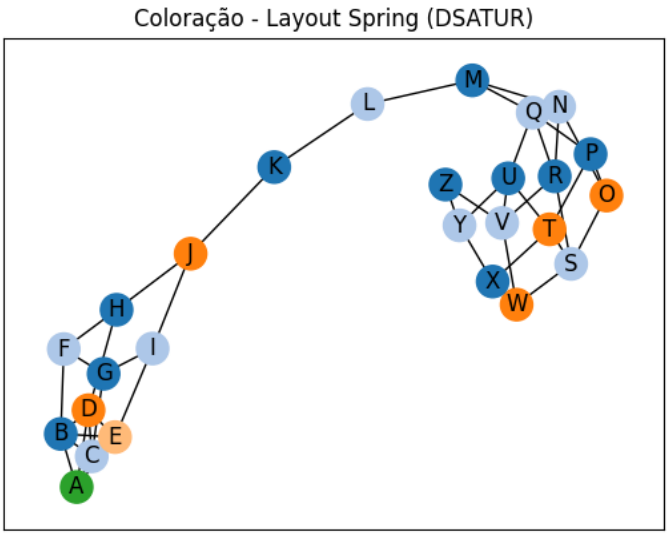

Figure 14. Visualização inicial do grafo - Dataset de Teste 1.

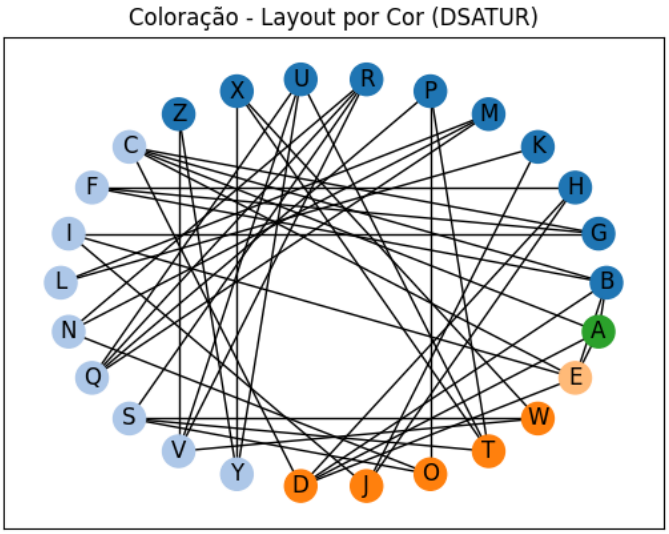

Figure 15. Processamento intermediário - Dataset de Teste 1.

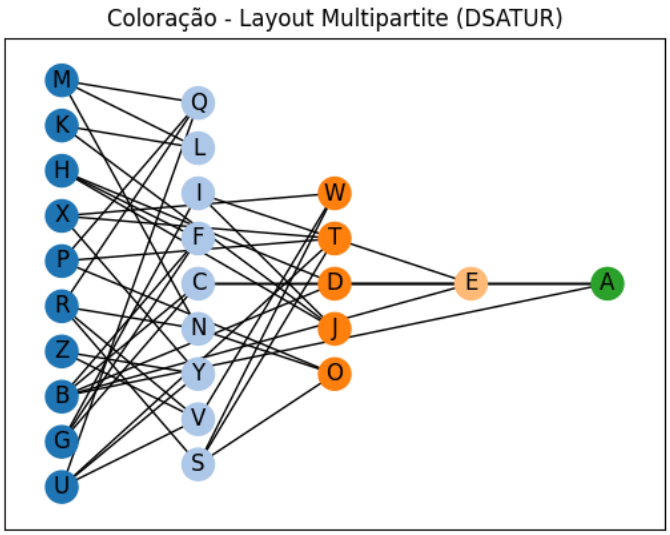

Figure 16. Resultado final da análise - Dataset de Teste 1.

5.5. Dataset de Teste 2

```
===== RESULTADOS =====
Algoritmo utilizado: DSATUR
Número mínimo de cores utilizadas (Horários): 10
Cor atribuída a cada disciplina:
  J: Azul Escuro
  A: Azul Claro
  B: Laranja Escuro
  C: Laranja Claro
  D: Verde Escuro
  E: Verde Claro
  F: Vermelho
  G: Salmao
  H: Roxo Escuro
  I: Roxo Claro
  K: Azul Claro
  L: Azul Escuro
  M: Azul Claro
  Q: Azul Escuro
  R: Azul Claro
  U: Azul Claro
  S: Azul Escuro
  T: Laranja Escuro
  P: Azul Claro
  O: Laranja Escuro
  N: Azul Escuro
  V: Azul Escuro
  Y: Azul Escuro
  X: Azul Claro
  W: Laranja Escuro
  Z: Azul Claro
Tempo de execução aproximado: 0.001107 segundos
======================
```

Figure 17. Resultados Dataset 10 cores.

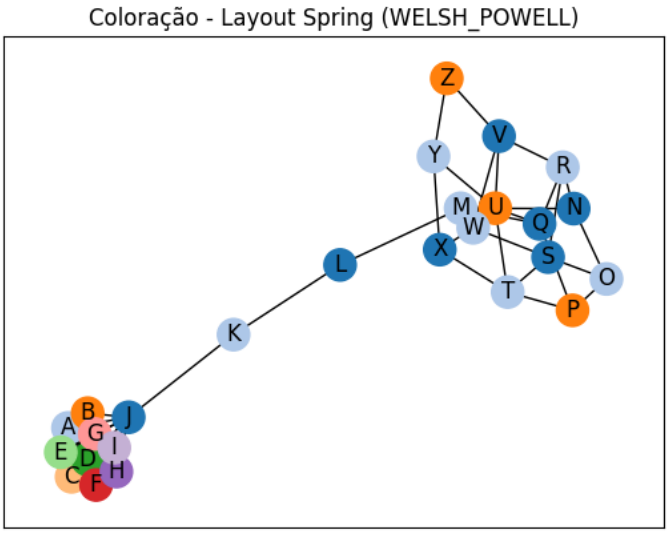

Figure 18. Visualização inicial do grafo - Dataset de Teste 2.

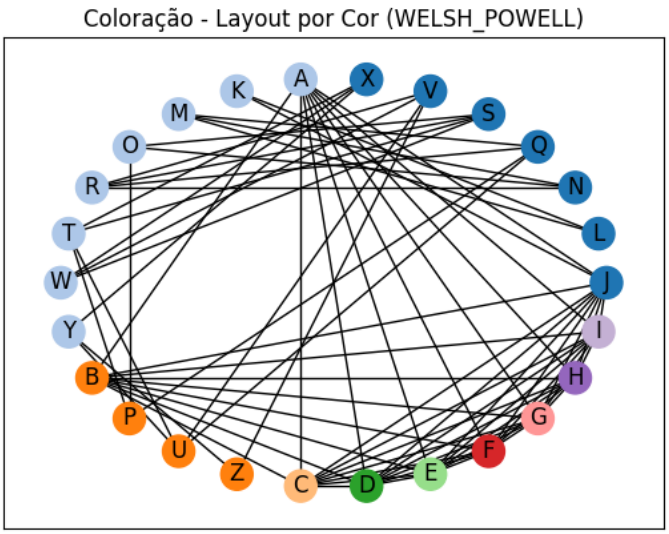

Figure 19. Processamento intermediário - Dataset de Teste 2.

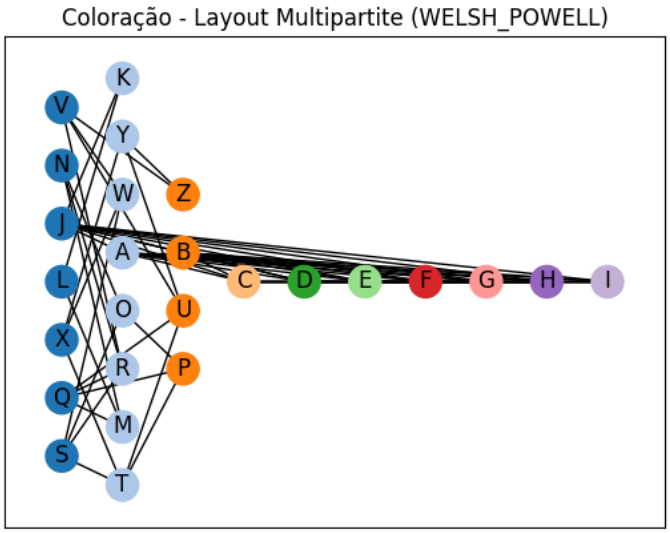

Figure 20. Resultado final da análise - Dataset de Teste 2.

6. Conclusão

O desenvolvimento deste trabalho permitiu consolidar os conhecimentos adquiridos na disciplina de Teoria dos Grafos por meio da implementação prática do problema de coloração de grafos. A utilização das bibliotecas *NetworkX* e *GCol* em Python demonstrou-se eficiente para a construção, visualização e análise dos grafos gerados a partir dos *datasets* fornecidos. A experimentação com diferentes tamanhos de instâncias — pequeno, médio e grande — evidenciou o impacto do aumento da complexidade na execução dos algoritmos e na visualização das estruturas resultantes. Além disso, a comparação entre distintos layouts de posicionamento possibilitou uma compreensão mais clara das relações entre os vértices e da distribuição das cores. Assim, o trabalho reforçou a importância dos grafos como ferramenta de modelagem e análise em problemas computacionais, ao mesmo tempo em que destacou o valor da prática e da pesquisa complementar para o aprendizado efetivo dos conceitos teóricos.

Referências

- An Introduction to the GCol Library.
  Disponível em: `https://gcol.readthedocs.io/en/latest/demo/Demo.html#node-coloring-and-visualization`.
  Acesso em: nov. 2025.

- Tutorial NetworkX.
  Disponível em: `https://networkx.org/documentation/stable/tutorial.html`.
  Acesso em: nov. 2025.