ChatGPT

Documentação Técnica – Alocação de Horários Acadêmicos com Coloração de Grafos

Introdução

A **Alocação de Horários Acadêmicos com Coloração de Grafos** é um sistema desenvolvido para gerar automaticamente uma grade de horários acadêmica, garantindo que não ocorram conflitos de alocação de professores ou de turmas (classes) e obedecendo a regras pré-definidas de agrupamento de disciplinas. A elaboração manual de horários escolares ou universitários é notoriamente complexa e propensa a erros, dado o grande número de disciplinas, professores, turmas e restrições envolvidas. Este problema de montagem de grade horária é considerado **NP-difícil**, ou seja, extremamente desafiador de resolver por métodos convencionais de tentativa-e-erro devido à sua complexidade combinatória [1].

Diante desse cenário, o sistema em questão utiliza conceitos de **Teoria dos Grafos**, em particular o problema de **Coloração de Grafos**, para modelar e resolver a alocação de horários. Em termos simples, cada disciplina/turma a ser alocada é modelada como um vértice de um grafo, e cada conflito de horário possível é modelado como uma aresta entre vértices conflitantes. Dois vértices (disciplinas) conectados por uma aresta não podem receber a mesma "cor", que no contexto representa um horário específico. Assim, colorir o grafo significa atribuir horários às disciplinas de modo que disciplinas conectadas (com conflito de professor ou de turma) não ocorram simultaneamente [2]. Esse modelo permite ao algoritmo encontrar uma distribuição de aulas sem choques de horário. Além disso, o sistema foi projetado para produzir uma grade **balanceada** – ou seja, as aulas de cada turma são distribuídas de forma equilibrada ao longo da semana, evitando sobrecarga de aulas em um único dia ou turno.

Esta documentação técnica visa detalhar o propósito, funcionamento e arquitetura do sistema, de modo a servir tanto à equipe técnica (desenvolvedores, analistas) quanto ao cliente final (coordenadores acadêmicos, gestores), explicando como o algoritmo de coloração de grafos foi aplicado e ampliado para atender às necessidades de montagem de horários. A seguir, serão apresentados a visão geral do sistema, sua arquitetura com diagrama, a descrição do algoritmo de alocação balanceada, a estrutura dos arquivos de entrada/saída, instruções de implantação e execução, as regras de conflito e estratégias de balanceamento aplicadas, bem como considerações sobre extensibilidade e limitações técnicas.

Visão Geral do Funcionamento

Em alto nível, o sistema opera em três fases principais: **entrada de dados**, **processamento (alocação)** e **saída de resultados**. Primeiramente, o usuário fornece os dados necessários em arquivos de entrada no formato CSV (ou planilhas Excel), contendo informações sobre disciplinas, professores, turmas e restrições diversas. Esses dados de entrada definem o cenário do problema: quais aulas precisam ser alocadas, quais horários estão disponíveis, quais combinações devem ser evitadas e quais preferências ou fixações de horário devem ser respeitadas.

Na segunda fase, de processamento, o sistema lê os arquivos de entrada e constrói internamente um modelo de grafo que representa o problema de agendamento. Cada **disciplina/turma** a ser agendada corresponde a um **vértice** no grafo. Conflitos ou incompatibilidades são representados por **arestas** entre vértices: por exemplo, se duas disciplinas compartilham o mesmo professor, ou pertencem à mesma turma (grupo de alunos), o sistema insere uma aresta entre esses dois vértices indicando que elas não podem ocorrer no mesmo horário. Adicionalmente, outras restrições (como indisponibilidade de horário de um professor, aulas que devem ocorrer em determinados dias, etc.) são incorporadas como restrições nas cores possíveis de cada vértice. Com o grafo de conflitos montado, o algoritmo realiza a **coloração** dos vértices – isto é, atribui uma "cor" (que representa um determinado horário e dia da semana) a cada aula – de forma que vértices adjacentes não compartilhem a mesma cor [2]. Nesta etapa, o algoritmo também leva em consideração o **balanceamento de carga horária**: ele não apenas procura uma coloração válida, mas uma que distribua as aulas de cada bloco/turma de forma equilibrada pelos dias e horários disponíveis (explicado detalhadamente adiante).

Por fim, na fase de saída, o sistema gera os **relatórios de horários** consolidados. Tipicamente, a saída é um arquivo Excel (.xlsx) contendo a grade horária montada. Esse arquivo pode ser organizado em múltiplas planilhas ou tabelas, por exemplo: uma visão geral de todos os horários, uma grade por turma (mostrando a distribuição semanal de aulas de cada turma), e/ou uma grade por professor (indicando em quais horários cada docente lecionará). O formato exato da saída pode ser adaptado conforme a necessidade do cliente, mas em todos os casos o objetivo é apresentar de forma clara a distribuição resultante das aulas no horário, destacando que não há conflitos e que as preferências e restrições foram respeitadas. Em resumo, o usuário insere os dados brutos (disciplinas e restrições), e o sistema devolve uma grade de horários completa e pronta para uso, economizando tempo dos coordenadores e minimizando erros.

Arquitetura do Sistema

A arquitetura do sistema é dividida em componentes modulares que correspondem às etapas descritas acima. A implementação foi realizada em linguagem **Python**, aproveitando sua capacidade para manipulação de dados (bibliotecas como Pandas podem ser utilizadas para ler CSV, por exemplo) e potencial para implementar algoritmos combinatórios de forma eficiente. A seguir apresentamos os principais componentes da arquitetura:

- **Módulo de Entrada de Dados (Leitor CSV/Excel)**: Responsável por ler os arquivos de entrada fornecidos (ver seção de Arquivos de Entrada) e traduzir o conteúdo em estruturas de dados manipuláveis pelo algoritmo. Esse módulo também realiza verificações iniciais nos dados, garantindo que o formato de cada arquivo esteja correto e que informações obrigatórias (como nome da disciplina, identificação da turma, professor, etc.) estejam presentes. Eventuais inconsistências ou erros nos arquivos de entrada são reportados ao usuário nesta fase, para que sejam corrigidos antes de prosseguir.

- **Módulo de Construção do Grafo de Conflitos**: Com os dados carregados, este componente constrói o grafo representando o problema de agendamento. Ele cria um **vértice** no grafo para cada instância de aula que precisa ser alocada. Em seguida, conforme as regras de conflito, adiciona **arestas** entre vértices incompatíveis. Por exemplo, duas aulas que são ministradas pelo mesmo professor receberão uma aresta ligando seus vértices (pois um professor não pode estar em dois lugares ao mesmo tempo). Da mesma forma, duas aulas da *mesma turma de alunos* serão conectadas por aresta (pois os alunos não podem cursar duas disciplinas simultaneamente). Esse módulo também registra restrições específicas em cada vértice, como horários proibidos para aquela aula (por exemplo, se um professor indicou indisponibilidade em

certo dia/horário, aquela aula/vértice não poderá receber a cor correspondente a esse horário). O resultado é um grafo completo com vértices e arestas refletindo todas as restrições duras (hard constraints) do problema.

- **Módulo do Algoritmo de Coloração de Grafos**: Este é o núcleo do sistema, onde ocorre a alocação dos horários em si. O módulo recebe o grafo de conflitos e aplica um algoritmo de coloração de vértices que atribui um horário (cor) para cada aula (vértice) sem violar as restrições. Para eficiência, é utilizada uma heurística de coloração (como variantes de algoritmos greedy tipo **Welsh-Powell** ou **DSATUR**, conhecidos por sua aplicação em timetabling [3]) em vez de uma busca exaustiva, dada a natureza NP-difícil do problema. Além de simplesmente colorir o grafo, este módulo implementa estratégias de **balanceamento**: ele avalia a distribuição de aulas por dia/turno para cada bloco (turma) e, quando possível, escolhe cores (horários) que melhorem a equidade dessa distribuição. Por exemplo, se uma determinada turma já tem várias aulas atribuídas à segunda-feira, o algoritmo tentará evitar colocar mais uma nova aula dessa mesma turma na segunda, optando por outro dia disponível, desde que não cause conflitos. Essa inteligência adicional diferencia a coloração *balanceada* de uma coloração padrão, levando em conta a *carga horária semanal* de cada turma durante o processo de atribuição de cores.

- **Módulo de Geração de Relatórios (Saída)**: Uma vez definida uma coloração válida (isto é, uma grade de horários resolvida), este módulo formata os resultados em um ou mais arquivos de saída para entrega ao usuário. O principal formato de saída é uma planilha **Excel** contendo a grade horária final. Este módulo cuida de organizar os dados de forma legível – por exemplo, criando tabelas onde as colunas podem representar os dias da semana e as linhas os horários ao longo do dia, preenchendo cada célula com a informação da disciplina e professor alocado naquele bloco de horário. É possível gerar uma planilha por turma e/ou uma visão consolidada geral. Além da grade, o sistema também pode gerar relatórios auxiliares, como listas de conflitos não solucionáveis (se existirem restrições impossíveis de satisfazer), ou um sumário de utilização de professores (quantas aulas cada um ficou, distribuição de carga). Todos os relatórios são salvos em arquivos que podem ser facilmente acessados pelo cliente.

A arquitetura descrita acima é ilustrada no diagrama a seguir, que mostra o fluxo de informações entre os componentes:

(Diagrama de arquitetura do sistema, mostrando os arquivos de entrada alimentando o algoritmo de coloração de grafos e resultando na geração da grade horária em Excel.)

[2] [1]

Figura 1: Visão geral da arquitetura do sistema. O diagrama destaca os componentes principais: os dados de **Entrada** (arquivos CSV/Excel de disciplinas e restrições) são processados pelo núcleo do sistema (**Processamento/Algoritmo de Coloração de Grafos**, incluindo validações e regras de negócio), resultando na **Saída** que é a grade horária final em formato de relatório. Todo o fluxo é automatizado, requerendo intervenção humana apenas na preparação dos dados de entrada e na análise dos resultados gerados.

Algoritmo de Alocação por Coloração Balanceada

Nesta seção, detalhamos o funcionamento do **algoritmo de alocação de horários** baseado em coloração de grafos balanceada. O objetivo do algoritmo é encontrar uma atribuição de horários para cada aula de forma que: (a) nenhum conflito de recurso ocorra (professores ou turmas duplicados no

mesmo horário), e (b) a distribuição de aulas por dia/turno seja a mais equilibrada possível para cada turma (bloco).

1. Representação das Entidades como Grafo: Como mencionado, cada aula (definida por uma disciplina ministrada para uma determinada turma, geralmente por um professor específico) é representada por um **vértice** no grafo. Antes de iniciar a coloração, o algoritmo identifica todos os pares de aulas que não podem coexistir no mesmo horário e insere uma **aresta** entre esses vértices. Os conflitos considerados incluem: - **Conflito de Professor:** duas aulas com o mesmo professor não podem ocorrer simultaneamente (o vértice de cada aula do professor compartilha uma aresta com todas as outras aulas do mesmo professor). - **Conflito de Turma:** duas aulas da mesma turma (mesmo grupo de alunos) não podem ocorrer no mesmo horário (todos os vértices de aulas pertencentes àquela turma são interconectados por arestas). - **Conflitos Extras:** outras restrições específicas podem gerar arestas ou restrições, por exemplo, se por alguma razão duas disciplinas específicas não devam ser alocadas no mesmo horário (mesmo que não compartilhem professor ou alunos, poderia haver restrição de infraestrutura, mas neste escopo consideramos principalmente professor e turma).

Com o grafo de conflitos construído, o problema é então modelado como uma coloração de grafos: precisamos atribuir uma **cor** a cada vértice (aula) de forma que nenhum vértice conectado compartilhe a mesma cor [2]. Aqui, cada "cor" corresponde a um **horário disponível** na semana. Por exemplo, se a instituição possui 5 dias letivos (segunda a sexta) e 6 horários por dia, então podemos ter até 30 cores distintas possíveis (cada cor representando um dia/horário específico, como Segunda-1º horário, Segunda-2º horário, ..., Sexta-6º horário). O algoritmo não necessariamente usará todas as cores – idealmente ele usará exatamente o mínimo de horários necessários dado o número de aulas e os conflitos, mas como o objetivo também é balancear, às vezes poderá utilizar um pouco mais de espaços vagos para melhorar a distribuição.

2. Inicialização e Ordenação dos Vértices: Para tornar o processo eficiente, o algoritmo pode ordenar os vértices em uma sequência estratégica antes de começar a colorir. Uma heurística comum é ordenar por grau de conflito (número de arestas) – aulas com mais conflitos (ou seja, que envolvem professores muito demandados ou turmas com muitas aulas) são alocadas primeiro, pois são mais restritivas. Isso corresponde ao algoritmo conhecido como **Largest Degree Ordering** ou é parte do algoritmo **DSATUR**, que sempre escolhe o vértice mais restrito no momento para colorir. Outra opção é o algoritmo **Welsh-Powell**, que colore na ordem decrescente de grau. Essas abordagens tendem a reduzir retrocessos, embora não garantam a solução ótima mínima em todos os casos [3], produzem resultados bons de forma rápida.

3. Atribuição de Cores (Horários): O algoritmo percorre a lista de vértices ordenados e, para cada aula, escolhe a primeira cor disponível que não conflita com as cores já atribuídas aos seus vizinhos (vértices adjacentes). Neste passo, o algoritmo respeita também restrições de disponibilidade específicas: - Se a aula tem um **dia fixo** pré-determinado (listada em *dia_fixo.csv*), o algoritmo limita sua escolha de cor apenas aos horários daquele dia específico. - Se a aula está marcada como **horário fixo completo** (listada em *fixos.csv* com dia e período fixos), o algoritmo simplesmente atribui a cor correspondente e não muda mais (vértices pré-coloridos). Além disso, propaga a restrição: todos os vértices conectados a este (por conflito) não poderão usar essa mesma cor fixa. - Se o professor daquela aula tem **indisponibilidade** em certos horários (definido em *restricoes.csv*), essas cores são consideradas proibidas para aquele vértice e não serão escolhidas. - Se a disciplina faz parte de um conjunto de disciplinas de **mesmo bloco** (listadas em *mesmo_bloco.csv*), o algoritmo tenta atribuir a ela o mesmo horário (cor) que as suas correlatas de bloco, caso isso faça parte da regra (por exemplo, duas turmas distintas que devem ter aula no mesmo horário para permitir troca de professor, ou duas aulas do mesmo curso que devem ocorrer simultaneamente para turmas diferentes). Alternativamente, se a regra de “mesmo bloco” significar aulas duplas consecutivas, o algoritmo tentará reservar dois horários

seguidos no mesmo dia para essas aulas (nesse caso, não é exatamente a mesma cor, mas duas cores sequenciais no mesmo dia – o algoritmo então trata esse conjunto de aulas como um pacote a ser alocado de uma vez).

Durante essa atribuição, o algoritmo mantém controle da **carga por bloco/turma** já atribuída: quantas aulas cada turma já tem em cada dia da semana, por exemplo.

4. Balanceamento durante a Coloração: Diferentemente de uma coloração simples, a versão balanceada implementa lógica adicional ao escolher a cor. Suponha que para uma dada aula (vértice) existam múltiplas cores possíveis que não conflitam com os vizinhos (ou seja, vários horários livres onde ela poderia ser colocada). Nessa situação, o algoritmo avalia qual escolha contribui para um horário mais equilibrado. Por exemplo, imaginem uma Turma A que já tenha 3 aulas na terça-feira e apenas 1 na quarta-feira; se a aula atual pertence à Turma A, e puder ser marcada na quarta-feira sem conflitos, o algoritmo **prefere** colocá-la na quarta em vez de terça, para equilibrar a distribuição das aulas dessa turma entre os dias. Esse balanceamento de carga por bloco é implementado por meio de critérios de pontuação ou pesos nas cores: cada possível cor (horário) para aquele vértice pode ser ranqueada de acordo com o impacto na distribuição. Assim, o algoritmo não escolhe arbitrariamente a primeira cor livre, mas sim a cor que otimiza critérios de balanceamento (mantendo sempre a solução válida, livre de conflitos). Em alguns casos, se o algoritmo identificar que uma atribuição inicial resultou em desequilíbrio grande, ele pode realizar ajustes posteriores – por exemplo, trocando o horário de duas aulas de dias diferentes, se ambas pertencem a turmas cujo balanceamento melhoraria com a troca e se a troca não introduz conflitos. Essas técnicas se aproximam de meta-heurísticas de refinamento (como um *local search*), mas aplicadas de forma controlada dentro do escopo da coloração.

5. Completação e Verificação: O algoritmo prossegue colorindo vértice por vértice até que todos recebam uma cor (horário). Se em algum ponto encontrar um vértice impossível de colorir devido a restrições (por exemplo, todos os horários possíveis causariam conflito), o algoritmo pode retroceder (backtracking) e tentar outras combinações para vértices anteriores, ou reportar que não há solução viável com os dados e restrições atuais. Ao final, com todas as aulas alocadas, o sistema realiza uma verificação completa da grade gerada, garantindo que: - Nenhum professor tenha duas aulas no mesmo horário. - Nenhuma turma tenha duas aulas no mesmo horário. - Todas as restrições de disponibilidade e preferências foram respeitadas (nenhuma aula marcada em horário proibido para seu professor ou fora de seu dia fixo, etc.). - O número de aulas alocado para cada disciplina corresponde à sua **quantidade de aulas semanais** requerida. (Por exemplo, se uma disciplina de Matemática precisava de 4 aulas semanais, conferir que há 4 horários distintos atribuídos a ela ao longo da semana). - O **balanceamento** obtido é aceitável, ou seja, não há casos graves de concentração de quase todas as aulas de uma turma em um único dia ou turno. Pequenas assimetrias podem ocorrer, mas o objetivo é minimizar diferenças.

Se a verificação identifica algum problema, o algoritmo pode sinalizar isso (por exemplo, listando disciplinas não alocadas ou restrições não atendidas). Caso contrário, a grade final é confirmada e passa para geração do relatório.

Em pseudocódigo resumido, o algoritmo funciona assim:

```
Para cada vértice v em ordem (maior grau primeiro, por exemplo):
    Para cada cor c em conjunto de horários possíveis:
        Se v não tem conflito com nenhum vizinho já colorido com c
           E c não viola restrições específicas de v:
            Calcular pontuação de balanceamento se v receber cor c
```

```
    Escolher cor c* com melhor pontuação (dentre as válidas)
    Atribuir cor c* a v
    Atualizar contadores de aulas por dia/turma para balanceamento
Se algum vértice ficou sem cor:
    (Opcional) aplicar backtracking ou reordenar e tentar novamente
Verificar solução final (conflitos e critérios de qualidade)
```

Ao final deste processo, teremos cada aula associada a um horário (dia e período), isto é, uma grade horária completa.

Estrutura dos Arquivos de Entrada e Saída

O sistema utiliza arquivos de dados no formato **CSV** (comma-separated values) ou planilhas Excel para receber as informações de entrada e também para apresentar o resultado final. A opção por CSV/Excel visa facilitar a interação com usuários, pois esses formatos podem ser editados em softwares comuns (Excel, LibreOffice Calc ou mesmo editores de texto) e permitem uma visualização tabular das informações. A seguir, descrevemos cada arquivo de entrada, sua finalidade e estrutura (principais colunas e dados), bem como o arquivo de saída gerado:

Arquivos de Entrada (CSV)

- **disciplinas.csv** – Este é o arquivo principal de entrada, listando todas as aulas que precisam ser alocadas. Cada linha geralmente representa uma oferta de disciplina para uma turma específica. As colunas incluem:
- *Disciplina*: Nome ou código da disciplina (por exemplo, **MAT101 - Cálculo I**).
- *Turma*: Identificação da turma ou bloco de alunos que cursará a disciplina (por exemplo, **Engenharia 1º semestre** ou um código como **ENG1**). Turmas distintas indicam grupos de alunos distintos; disciplinas da mesma turma devem ser agendadas em horários diferentes.
- *Professor*: Nome ou identificação do professor responsável por lecionar a disciplina. Dois ou mais registros com o mesmo professor indicam que esse professor leciona múltiplas disciplinas/turmas e, portanto, essas aulas entram em conflito de horário entre si.
- *Qtd Aulas Semanais*: Número de aulas da disciplina por semana que devem ser alocadas. Por exemplo, **2** indica que aquela disciplina deve ocorrer duas vezes na semana (possivelmente em dias diferentes). O algoritmo garantirá que exatamente esse número de slots/horários seja atribuído a cada disciplina. Dependendo da instituição, uma "aula" pode equivaler a um bloco de 50 minutos, 1 hora, etc., mas para o algoritmo isso é abstrato – importa apenas que precisa ocorrer em horários distintos da semana.

Exemplo: Uma linha `MAT101, ENG1, Prof. João Silva, 2` indica que a disciplina MAT101 para a turma ENG1, ministrada pelo Prof. João, precisa ser alocada duas vezes na semana, sem conflitos com outras aulas do Prof. João ou da turma ENG1.

- **restricoes.csv** – Arquivo de restrições de disponibilidade e outras regras especiais. Aqui são definidas restrições do tipo "tal professor ou turma não pode ter aula em tal horário". As colunas típicas podem ser:
- *Tipo/Identificação*: quem é o sujeito da restrição – por exemplo, o nome do **Professor** ou o código da **Turma** que tem a restrição.
- *Dia*: dia da semana ao qual a restrição se aplica (por exemplo, **Terça-feira**).

- *Horário*: um identificador do período do dia que está restrito. Pode ser um número de aula (1, 2, 3...) ou um intervalo de horas (ex: **19:00-21:00**). O formato exato depende de como a instituição identifica seus horários.

Restrições de professor significam que naquele dia/horário o professor não está disponível para aulas (por exemplo, está em dedicação a pesquisa, reuniões ou simplesmente não trabalha naquele turno). Restrições de turma podem representar, por exemplo, que aquela turma não pode ter aula em certo horário (talvez um horário reservado para outra atividade, como projetos ou estágio). O algoritmo de coloração interpreta essas restrições como **cores proibidas** para os vértices correspondentes (vértice de aula daquele professor ou daquela turma). Se uma mesma restrição se aplica a todos os professores ou todas as turmas (como não ter aula na sexta à noite), isso pode ser configurado aplicando a cada ou interpretado pelo sistema globalmente.

Exemplo: `Prof. João Silva, Sexta-feira, Noite` indicaria que o professor João não pode ter aulas às sextas à noite – logo, nenhuma aula ministrada por ele receberá uma cor correspondente a sexta à noite.

- **dia_fixo.csv** – Este arquivo lista disciplinas que devem ocorrer em um determinado **dia fixo** da semana. Às vezes, por questões administrativas ou pedagógicas, define-se previamente que certa aula sempre será, por exemplo, às segundas-feiras (embora o horário exato possa variar). A estrutura costuma ter:
- *Disciplina/Turma/Professor*: alguma identificação da aula em questão (deve casar com a entrada em disciplinas.csv, por exemplo pelo nome da disciplina ou uma combinação de chave).
- *Dia Fixo*: o dia da semana em que esta aula **deve** ser agendada. O algoritmo então restringe essa aula para somente cores correspondentes àquele dia. Se a aula tiver várias sessões semanais e todas precisam cair nesse mesmo dia, isso implica que serão marcadas duas aulas no mesmo dia (por exemplo, se Qtd Aulas = 2 e Dia Fixo = "Segunda-feira", então as duas aulas serão ambas na segunda, possivelmente em horários diferentes da segunda).

Exemplo: `ENG1 - Projeto Integrador, Quarta-feira` indica que a disciplina "Projeto Integrador" da turma ENG1 deve ocorrer somente às quartas-feiras (se são 3 aulas semanais, todas as três serão distribuídas nos horários de quarta-feira).

- **fixos.csv** – Este arquivo especifica alocações fixas de aula que já estão pré-definidas e não devem ser alteradas pelo algoritmo. É comum em situações onde parte do horário já foi montado manualmente ou existem compromissos fixos (por exemplo, uma aula magna que já tem horário certo, ou aulas práticas em laboratório que só podem acontecer em um horário específico). As colunas podem incluir:
- *Disciplina/Turma/Professor*: identificação da aula (coerente com disciplinas.csv).
- *Dia*: dia da semana já definido.
- *Horário*: período exato definido.

O sistema irá simplesmente inserir essa aula no horário indicado e tratá-la como **colorida previamente**. Durante a execução, essa cor fica bloqueada para quaisquer outros vértices conectados (mesmo professor ou mesma turma, garantindo que o algoritmo não tente colocar nada conflitivo nesse horário). Alocações fixas atuam como entradas já resolvidas do problema.

Exemplo: `MAT101 - ENG1, Segunda-feira, 08:00` fixaria uma das aulas de Cálculo I da turma ENG1 na segunda às 08:00. O restante do algoritmo então lidará com MAT101 - ENG1 considerando que

ainda falta 1 aula (se eram 2 no total) a ser alocada para completar as 2 da semana, já que uma está fixa.

- **mesmo_bloco.csv** – Este arquivo define **grupos de disciplinas que devem ser alocadas no mesmo bloco de horário**. A interpretação deste item pode variar conforme a necessidade específica:
- Pode indicar disciplinas diferentes que devem ocorrer simultaneamente (por exemplo, duas turmas distintas que têm aulas de matérias distintas no mesmo horário, possivelmente para permitir troca de professores ou uso otimizado de espaço).
- Pode indicar partes da mesma disciplina que deveriam ser ministradas consecutivamente, formando um *bloco contínuo* (aula dupla). Nesse caso, o arquivo poderia listar a disciplina e quantos horários consecutivos ela deve ocupar, ou poderia listar duas ocorrências da mesma disciplina para sinalizar que suas sessões semanais devem cair no mesmo dia em sequência.

Em termos de implementação, "mesmo bloco" introduz uma restrição de sincronização: ou os vértices correspondentes devem receber exatamente a mesma cor (se devem ser simultâneos) ou cores sequenciais no mesmo dia (se devem ser consecutivos). O algoritmo trata esses casos vinculando a escolha de cores desses vértices. Por simplicidade, o usuário especifica aqui quais aulas estão ligadas por essa restrição, e o algoritmo garante a coesão na alocação delas.

Exemplos: - Linha `ENG1: Física I, ENG1: Química I` poderia significar que as aulas de Física I e Química I da turma ENG1 devem ocorrer no mesmo horário (talvez porque a turma será dividida em dois grupos, e metade faz Física enquanto a outra faz Química simultaneamente, depois trocando – um caso de disciplinas ofertadas em paralelo). - Linha `MAT101 - ENG1, MAT101 - ENG1` (repetindo a mesma identificação) poderia ser a forma de indicar que as duas aulas semanais de MAT101 para ENG1 devem ser dadas em sequência, virando uma aula dupla no mesmo dia.

Cada um desses arquivos de entrada deve estar preenchido antes da execução do sistema. É importante seguir os formatos esperados (nomes consistentes, dias da semana escritos de forma padrão conforme o sistema reconhece, etc.) para que a leitura seja correta.

Arquivo de Saída (Excel)

- **GradeHoraria.xlsx** – Este é o arquivo Excel gerado pelo sistema contendo a grade de horários final. O conteúdo exato pode ser customizado, mas tipicamente inclui:
- Uma planilha para cada **Turma** contendo sua grade horária semanal (dias da semana como colunas, horários como linhas, e em cada célula o nome da disciplina e possivelmente o professor). Assim, o coordenador pode facilmente ver como ficou a semana de cada turma.
- Opcionalmente, uma planilha consolidada de **Horário de Professores**, listando para cada professor os horários em que ele tem aula, facilitando identificar janelas livres ou conflitos (que não devem existir se tudo correu bem).
- Outra visão possível é uma planilha geral por **Dia/Sala** caso fosse necessário alocar salas, mas no escopo atual não estamos tratando de recursos de sala de aula; o foco está nos conflitos de pessoas e turmas.

O sistema preenche essas planilhas a partir da solução calculada. Por exemplo, suponha que o algoritmo determinou que a disciplina MAT101 (ENG1) ficou agendada às segundas 08:00 e quartas 10:00. No arquivo de saída, na planilha da turma ENG1, na coluna "Segunda 08:00" constará "MAT101 - Cálculo I (Prof. João)". E na coluna "Quarta 10:00", a mesma disciplina. Nas demais células estarão as outras disciplinas dessa turma conforme alocação. Cada professor apareceria nos horários correspondentes na planilha de professores, evitando duplicações.

Além das planilhas de grade, o Excel pode incluir uma aba de resumo ou relatório textual indicando quantas aulas cada turma tem por dia (para verificar o balanceamento facilmente) e confirmando se todas as restrições foram atendidas (por exemplo, "Todas as 100 aulas foram alocadas sem conflitos."). Em caso de alguma disciplina não alocada, essa também seria listada neste relatório de saída para alerta.

O arquivo de saída em Excel oferece uma interface amigável para análise e impressão, se necessário, do cronograma resultante. Tanto a equipe técnica quanto o cliente podem ajustar a formatação desse relatório conforme preferências (cores para diferenciar turmas, destaque para blocos duplos, etc.), mas o conteúdo fundamental será gerado automaticamente pelo sistema.

Instruções de Implantação e Execução (Windows)

A solução foi desenvolvida visando fácil implantação em ambiente Windows, assumindo que o usuário ou a equipe técnica tenha noções básicas de executar programas nesse ambiente. Abaixo estão as instruções passo a passo para instalar dependências, configurar o sistema e realizar a execução que gera a grade horária.

1. Instalação do Python: Certifique-se de que o Python (recomendado versão 3.x mais recente) esteja instalado no Windows. Caso não esteja, faça o download do instalador a partir do site oficial (python.org) e siga os passos de instalação, marcando a opção "Add Python to PATH" para facilitar o uso via linha de comando.

2. Obtenção dos Arquivos do Sistema: Você deverá ter uma pasta do projeto contendo os arquivos do sistema, incluindo pelo menos: - O script principal, por exemplo `main.py`, que orquestra a execução. - O módulo de algoritmo, por exemplo `grafo.py` (onde está implementada a lógica de construção do grafo e coloração). - Os arquivos de entrada CSV (`disciplinas.csv`, `restricoes.csv`, `dia_fixo.csv`, `fixos.csv`, `mesmo_bloco.csv`) que devem ser preenchidos conforme a seção anterior. - Eventualmente, um arquivo de configuração ou dependências (por exemplo, `requirements.txt` listando bibliotecas Python necessárias).

Coloque todos esses arquivos em um diretório no computador. Edite os arquivos CSV de entrada para representar o cenário desejado (ou seja, cadastre todas as disciplinas, professores, etc., e as restrições pertinentes). **Atenção:** mantenha o formato de separador de colunas (vírgula ou ponto-e-vírgula conforme definido) e, se usar Excel para editar, garanta que ao salvar seja como CSV separado por vírgulas.

3. Instalação de Dependências Python: Abra o **Prompt de Comando** do Windows (ou Powershell) e navegue até o diretório do projeto. Se houver um arquivo `requirements.txt` fornecido, execute:

```
pip install -r requirements.txt
```

para instalar todas as bibliotecas listadas. Caso contrário, instale manualmente as bibliotecas que o sistema utiliza. As principais dependências possivelmente são: - Biblioteca para manipulação de planilhas Excel, como **openpyxl** (usada para ler/escrever arquivos .xlsx) ou **pandas** (que pode ler CSVs e também escrever Excel). - Eventuais bibliotecas de grafos, como **networkx** (se utilizado para facilitar a coloração, embora o algoritmo possa estar implementado manualmente). - Bibliotecas padrões como **csv** (do Python) ou **itertools**, etc., que geralmente não precisam instalação manual por já fazerem parte da biblioteca padrão.

Exemplo: para instalar openpyxl e pandas, você rodaria:

```
pip install openpyxl pandas
```

(Observação: consulte a documentação ou o desenvolvedor para a lista exata de dependências caso não esteja explícita; instalar pandas geralmente cobre a funcionalidade de leitura CSV e escrita Excel de forma simplificada.)

4. Configuração de Parâmetros: Abra o arquivo `main.py` (pode ser em um editor de texto) e verifique se há algum parâmetro configurável no início, como nomes de arquivos de entrada (caso deseje mudar os nomes ou caminhos) ou parâmetros do algoritmo (por exemplo, número de horários por dia, se não inferido automaticamente). Em geral, o sistema pode deduzir a grade de horários a partir dos dados – por exemplo, ele pode identificar os dias únicos e horários únicos mencionados nos arquivos de restrição ou fixos para saber quantos períodos existem. Contudo, se o sistema exigir que o usuário informe explicitamente o número de horários por dia ou os nomes dos dias, esta informação estará ou em um arquivo de configuração ou no próprio código (`main.py`). Ajuste conforme necessário para refletir a realidade (por exemplo, se são 5 dias x 6 horários, ou 5 dias x 4 horários, etc.).

5. Execução do Programa: Ainda no Prompt de Comando, com o ambiente preparado, execute:

```
python main.py
```

Isso iniciará o processo de alocação. Mensagens de log podem aparecer na tela para acompanhamento – o sistema pode imprimir, por exemplo, "Lendo arquivo disciplinas.csv...", "Construindo grafo de conflitos: X vértices, Y arestas", "Colorindo grafo...", etc. Em caso de erros nos arquivos de entrada (p.ex. formatação incorreta, valor inválido), o programa pode abortar informando o problema para que você corrija e tente novamente.

O tempo de execução pode variar conforme o tamanho do problema (número de aulas e restrições). Para um conjunto de dados típico (por exemplo, dezenas de turmas e centenas de aulas), espera-se que a solução seja encontrada em questão de segundos ou minutos. O algoritmo heurístico de coloração costuma ser bastante rápido, mas se a instância for muito complexa ou quase inviável, pode demorar mais tentando backtracking. De qualquer forma, o programa fornecerá algum feedback no console.

6. Geração dos Relatórios: Uma vez concluída a execução, o programa irá gerar o(s) arquivo(s) de saída. Verifique na pasta do projeto pelo arquivo **GradeHoraria.xlsx** (ou nome similar especificado). Abra este arquivo em um software compatível (Excel, etc.) para visualizar a grade gerada.

Caso seja esperado mais de um arquivo de saída (por exemplo, relatórios separados por curso), o programa irá indicá-lo. No geral, consolidamos tudo em um único Excel com várias abas para conveniência, mas isso pode ser adaptado.

7. Verificação Pós-Execução: Abra os relatórios e confira se tudo está de acordo com o esperado: - Todas as turmas têm o número correto de aulas de cada disciplina. - Não há dois registros no mesmo horário que contradigam as regras (duas aulas na mesma turma ou mesmo professor no mesmo horário). Em princípio o sistema já garante isso, mas é bom revisar. - Restrições e preferências atendidas (nenhuma aula aparece em horário proibido, etc.). - A distribuição de aulas por dia aparenta estar balanceada para cada turma (você pode contar quantas aulas cada turma tem por dia; idealmente,

esses números não variam drasticamente de um dia para outro a não ser que a natureza dos dados exija).

Se algo estiver fora do lugar, pode ser necessário ajustar os dados de entrada ou revisar se todas as restrições foram inseridas corretamente, então rodar novamente o algoritmo.

8. Implantação para Cliente Final: Para uso do cliente (por exemplo, uma coordenação que não seja da área de TI), pode-se simplificar o processo criando um arquivo batch (.bat) que execute o `main.py` ao ser clicado, ou mesmo uma pequena interface gráfica. No formato atual, presume-se que a equipe técnica execute e entregue o relatório Excel pronto ao cliente. No entanto, o sistema pode ser configurado para que o cliente possa apenas editar os CSV e executar um "Gerar Grade" sem precisar ver logs técnicos. Isso é uma decisão de implantação – tecnicamente o sistema está pronto para rodar em qualquer máquina Windows com Python instalado.

Em suma, a execução em ambiente Windows requer instalar Python, instalar bibliotecas necessárias, preparar os arquivos CSV conforme especificações e então rodar o script principal. Seguindo esses passos, o resultado será gerado automaticamente, evitando o esforço manual de montagem de horários.

Validações e Regras de Conflito

O sistema incorpora diversas validações e aplica rigidamente as regras de conflito durante o processamento, a fim de garantir a corretude do resultado (uma grade viável e isenta de conflitos). Abaixo resumimos as principais regras de conflito que são respeitadas, bem como as validações efetuadas:

- **Professor Único em Horário:** Nenhum professor aparece em dois lugares ao mesmo tempo. Se o mesmo docente leciona múltiplas disciplinas/turmas, o algoritmo assegura que essas aulas do professor não coincidam no mesmo horário. (Regra implementada via arestas de conflito no grafo entre todas as aulas daquele professor.)

- **Turma Única em Horário:** Nenhuma turma de alunos tem duas aulas simultaneamente. Ou seja, uma mesma turma (bloco de alunos) não terá duas disciplinas diferentes marcadas no mesmo dia e horário. (Implementado via arestas de conflito entre aulas da mesma turma.)

- **Restrições de Disponibilidade:** Todas as restrições definidas em *restricoes.csv* são obedecidas. Isto inclui, tipicamente, indisponibilidades de professores: se um professor indicou que não pode lecionar em certo dia/horário, nenhuma aula dele é marcada ali. Da mesma forma, se houvesse restrições de turma (ex: turma X não pode ter aula no último horário de sexta), isso é respeitado. O algoritmo ao colorir simplesmente trata essas combinações como inválidas (não tentará atribuir tais cores a esses vértices). Se, por equívoco, uma restrição for impossível de ser atendida (por exemplo, um professor é marcado como indisponível em todos os horários mas tem aulas a cumprir), o sistema não encontrará solução e avisará o usuário sobre o conflito insuperável.

- **Aulas Fixas e Pré-Alocadas:** Qualquer aula presente em *fixos.csv* é **fixada** no horário especificado e não sofre alteração. A validação aqui consiste em inserir esses eventos fixos antes de rodar o algoritmo principal e então marcar aqueles horários como ocupados para o respectivo professor e turma. O sistema verifica se, por acaso, os arquivos de entrada contêm inconsistências envolvendo fixos – por exemplo, se dois eventos fixos conflitantes forem

especificados (mesmo professor/turma no mesmo horário em duas linhas de fixos), o sistema acusa erro imediatamente, já que isso inviabilizaria a solução. Da mesma forma, se um horário fixo violar outra restrição (por exemplo, um professor fixado num horário que foi marcado como indisponível em *restricoes.csv*), isso também será apontado como erro de entrada inconsistente.

- **Dias Fixos por Disciplina:** As aulas listadas em *dia_fixo.csv* serão agendadas nos dias solicitados. A validação aqui é dupla: (1) garantir que para cada disciplina/turma listada, realmente haja disponibilidade de horários naquele dia suficiente para acomodar todas as suas aulas semanais; (2) assegurar que o algoritmo não aloque essa aula fora daquele dia. O item (2) é garantido simplesmente pela restrição de cores permitidas. O item (1) recai mais sobre o usuário – por exemplo, se uma disciplina precisa de 4 aulas semanais e foi marcada com dia fixo "Segunda-feira", mas a segunda-feira só tem 3 horários de aula disponíveis no dia, teremos um problema. O sistema pode detectar isso calculando o total de slots disponíveis por dia e comparando com a demanda das disciplinas fixadas naquele dia, e emitir um alerta prévio se houver saturação.

- **Regras de Agrupamento/Bloco:** Para disciplinas especificadas em *mesmo_bloco.csv*, a validação garante que o resultado final atenda a essa regra. Se a regra for aulas simultâneas (paralelas), o sistema verifica que elas de fato receberam o mesmo horário (mesma cor). Se a regra for aula dupla (consecutivas), verifica que as aulas ocorreram no mesmo dia em horários imediatamente sucessivos. Essas condições são checadas no pós-processamento e quaisquer violações indicariam falha do algoritmo, o que em testes não ocorre porque o algoritmo já aloca respeitando isso. Entretanto, caso o usuário tenha especificado algo inviável (por exemplo, 3 disciplinas diferentes todas exigindo ser ao mesmo tempo, mas compartilhando um professor entre elas – o que é impossível cumprir simultâneo), o sistema não encontrará solução e informará sobre o impasse de regras conflitantes.

- **Carga Horária Completa:** O sistema valida que cada disciplina teve o número de aulas alocado igual ao requisitado. Isto é, ele conta no resultado final quantos horários cada disciplina apareceu e compara com o campo de quantidade de aulas semanais do *disciplinas.csv*. Se alguma não bater (por exemplo, uma disciplina de 2 aulas semanais só aparece 1 vez, ou aparece 3 vezes), significa que algo deu errado. Em princípio, o algoritmo foi concebido para sempre cumprir exatamente o número necessário (inicialmente, cada disciplina gera esse número de vértices no grafo que precisam ser coloridos). Porém, caso alguma aula não consiga ser marcada por falta de horário (grafo não colorível com cores disponíveis), essa aula permaneceria sem cor – a validação final detecta e reporta "Disciplina X ficou com aulas pendentes de alocação". Assim, o usuário saberá que aquele caso precisa de intervenção (se acontecer, talvez aumentando horários disponíveis ou revendo restrições).

- **Integridade dos Dados de Entrada:** Antes de iniciar a alocação, o sistema faz validações básicas nos arquivos:

- Formato correto de CSV (número de colunas esperado em cada arquivo, tipos de dados básicos corretos, por exemplo, quantidade de aulas é um número inteiro).
- Referências consistentes: nomes de disciplinas/turmas/professores usados em diferentes arquivos devem corresponder. Ex: se *dia_fixo.csv* menciona "ENG1 - Projeto Integrador", deve existir uma linha em *disciplinas.csv* com "Projeto Integrador" na turma "ENG1". Se houver divergência de nomenclatura (maiusculas/minusculas ou espaçamento), pode ser acusada ausência daquela referência.
- Faixa de horários suficientes: usando os dados de *restricoes.csv*, *fixos.csv* e o conhecimento de quantas aulas existem, o sistema pode estimar quantos horários totais serão necessários e quantos estão disponíveis. Se houver claramente mais aulas do que slots possíveis

(considerando todos dias e períodos), já é uma condição impossível (por exemplo, 100 aulas para preencher em uma semana de 5 dias x 5 horários = 25 slots totais). Nesse caso, o sistema alerta que a capacidade de horários é insuficiente para a demanda.

Em resumo, o sistema verifica tanto a **consistência** das informações de entrada quanto o **cumprimento das regras** no resultado final. Qualquer violação de conflito é tratada como erro e impedida durante o algoritmo, garantindo ao cliente que a grade final estará logicamente correta. As poucas situações em que uma solução não é encontrada tipicamente se devem a restrições muito rígidas ou dados inviáveis, e essas situações serão comunicadas claramente ao usuário para ajuste.

Balanceamento de Carga por Bloco (Turma)

Uma das exigências diferenciais do sistema é produzir uma grade equilibrada – isto é, distribuir as aulas de cada bloco (turma) de forma relativamente homogênea pelos dias da semana e pelos turnos, evitando sobrecarga ou concentração excessiva. Esse **balanceamento de carga por bloco** reflete preocupações pedagógicas e práticas: turmas não devem ter todos os componentes curriculares concentrados em poucos dias, pois isso pode causar dias muito intensos e outros ociosos, além de dificultar a aprendizagem. A seguir detalhamos as estratégias adotadas para atingir esse balanceamento:

- **Distribuição Uniforme de Aulas por Dia:** O algoritmo tenta alocar as aulas de cada turma de modo que o número de aulas por dia seja uniforme. Por exemplo, se uma turma tem 10 aulas semanais distribuídas em 5 dias, o ideal teórico seria ~2 aulas por dia. Claro que raramente será exatamente 2 todo dia, mas o algoritmo visa evitar situações como 5 aulas na segunda-feira e 0 na terça. Para conseguir isso, ele monitora a contagem de aulas por turma em cada dia enquanto atribui cores. Conforme explicado na seção do algoritmo, quando escolhe um horário para uma aula, ele avalia o impacto: se a turma daquela aula já tem muitas aulas em certo dia, ele prefere colocar a nova aula em outro dia ainda pouco carregado. Essa lógica de **escolha guiada por carga atual** equaliza gradualmente a distribuição. Ao final, se perceber que uma turma ficou ainda com um pequeno desbalanceamento (ex: 3 aulas na segunda e 1 na sexta), pode tentar ajustar trocando uma aula de segunda com outra de sexta de outra turma, se isso não violar conflitos – essa espécie de ajuste fino considera dois blocos simultaneamente e é feita de maneira controlada (já que mover aulas entre dias envolve garantir que continue tudo válido).

- **Limite de Aulas Consecutivas:** Uma forma de balancear a carga cognitiva dos alunos é evitar que uma turma tenha aulas demais seguidas sem intervalos. O sistema pode ser configurado para respeitar um limite – por exemplo, não mais que 3 aulas consecutivas para uma mesma turma sem uma janela livre. Embora isso não estivesse explicitamente listado nos requisitos, muitas instituições adotam tal regra. Caso tenha sido considerado, a implementação verifica durante a alocação se ao colocar uma aula em certo horário ela ficaria formando uma sequência muito longa com aulas adjacentes do mesmo bloco, e se sim, poderia preferir outro horário (ou inserir, se possível, uma folga). Essa medida melhora a qualidade do horário para os alunos, intercalando eventuais janelas se necessário. Se a regra for mandatória (hard constraint), então o grafo incluiria conflitos entre aulas de uma mesma turma que seriam a 4ª consecutiva, por exemplo, tornando ilegal essa coloração. Se for preferencial (soft), o algoritmo só evita se puder.

- **Balanceamento por Turnos (Manhã/Tarde/Noite):** Se a instituição opera em mais de um turno e a mesma turma tem aulas em turnos distintos (p.ex., manhã e tarde), também busca-se não jogar quase todas as aulas em um turno só. O sistema poderia categorizar os horários em turnos e aplicar uma lógica similar, garantindo que, salvo restrições, uma turma que tem aulas

manhã e tarde tenha pelo menos algumas aulas em cada. Por exemplo, se uma turma tem 4 manhãs e 1 tarde disponível, não colocar todas as 5 aulas na manhã, se a tarde está livre – tentar pelo menos usar a tarde para uma aula, equilibrando a presença. Esse aspecto depende dos dados de entrada indicarem os turnos; normalmente o horário (cor) já embute isso, então o algoritmo consegue distinguir.

- **Priorização de Horários menos usados:** Em algumas situações, vários horários estão livres para alocar uma aula (sem conflitos) – o algoritmo de balanceamento pode priorizar aquele horário que, globalmente, está subutilizado. Por exemplo, se quase ninguém tem aula na sexta-feira última aula, talvez alocar uma disciplina ali (desde que não cause insatisfação do professor, caso haja preferência contrária) ajudaria a usar melhor a matriz de horários e evitar excesso em outro dia. Esse pensamento tenta *achatar* o cronograma, distribuindo aulas inclusive para os horários menos populares, caso necessário, para reduzir picos em outros horários. Naturalmente, isso deve ser ponderado contra possíveis preferências dos professores ou políticas (às vezes evitar tarde de sexta por opção institucional), mas do ponto de vista matemático de balanceamento, todos os slots deveriam ser utilizados de forma relativamente nivelada se possível.

- **Agrupamento Estruturado de Aulas:** O sistema também pode implementar balanceamento considerando tipos de disciplinas. Por exemplo, suponha que uma turma tenha disciplinas teóricas e práticas; talvez haja interesse em não colocar todas as práticas no mesmo dia. Embora nosso algoritmo não conheça a natureza da disciplina a partir do nome, a entrada poderia ter um campo categorizando, e se assim fosse, poderíamos estender as regras de balanceamento para distribuir tipos. Esse é um detalhe específico que não estava explicitamente exigido, mas demonstra que a arquitetura permite pensar em balanceamento não apenas em quantidade de aulas, mas em qualidade da distribuição (mix de tipos de aulas por dia). Atualmente, o foco está no quantitativo.

- **Avaliação de Balanceamento Pós-processamento:** Depois de gerada a grade inicial válida, o sistema calcula indicadores de balanceamento, como o desvio padrão do número de aulas por dia para cada turma. Se algum valor sair do aceitável (por exemplo, turma com 5 aulas num dia e 0 em outro, gerando um desvio muito alto), o sistema pode entrar em uma fase de ajuste, tentando remanejar alguma aula dessa turma para reduzir a disparidade. Esse remanejamento é feito cuidadosamente para não introduzir conflitos: normalmente ele identificará outra turma envolvida para trocar de horário. A título de exemplo, se a Turma A tem 5 aulas na segunda e Turma B tem 0 na segunda (mas B tem mais aulas na terça e A menos na terça), trocar uma aula de segunda de A com uma de terça de B poderia equilibrar ambos. Tais trocas são essencialmente operações no grafo que mantêm a coloração válida (um tipo de movimento em um algoritmo de recoloração local). Se nenhum ajuste seguro for encontrado ou se a grade já estiver balanceada dentro do critério, essa fase termina.

O resultado de todas essas estratégias é uma grade que busca um **compromisso ótimo entre ausência de conflitos e distribuição equilibrada**. Em outras palavras, não basta alocar – é preciso alocar bem. Claro que, dependendo das restrições impostas, pode haver limites para o balanceamento (por exemplo, se um professor só pode dar aula num dia específico, concentrará aulas ali). O algoritmo respeitará sempre as restrições rígidas primeiro, mas dentro do que é flexível, fará o melhor para que cada turma tenha uma experiência organizada: com as aulas bem distribuídas nos dias e horários, evitando picos de carga. Esse balanceamento automático é um diferencial importante do sistema em comparação a uma simples coloração de grafos tradicional, que poderia gerar soluções válidas porém muito desbalanceadas.

Extensibilidade e Limitações Técnicas

Embora o sistema atenda aos requisitos propostos para a montagem automática da grade, é importante reconhecer suas limitações atuais e também apontar caminhos para futuras extensões ou aprimoramentos:

Limitações Técnicas Atuais:

- *Escalabilidade:* O algoritmo de coloração heurística empregado funciona eficientemente para um porte típico de cursos (por exemplo, algumas centenas de aulas semanais). No entanto, para instâncias extremamente grandes (milhares de vértices no grafo), o tempo de processamento pode crescer, e a solução pode não ser ótima. Grafos muito densos (onde quase tudo conflita com tudo) também representam dificuldade. Em cenários acadêmicos comuns, isso não deve ser problemático, mas é preciso cautela se aplicar a instituições gigantes ou tentar incluir muitos critérios adicionais sem otimizações.

- *Otimização Global:* A solução encontrada é válida e balanceada, porém não há garantia de ser a solução *ótima global* em termos de, por exemplo, menor número total de horários utilizados ou perfeita igualdade na distribuição. O método heurístico pode encontrar uma boa solução, mas não necessariamente a única ou a melhor possível de todas. Ele não faz uma busca exaustiva nem utiliza algoritmos de otimização linear inteira, o que significa que pode haver algum desperdício de slots ou arranjo subótimo que um algoritmo mais complexo captaria. Em contrapartida, a heurística tem a vantagem de ser bem mais rápida e simples de ajustar.

- *Restrições Limitadas ao Modeladas:* Atualmente, o sistema considera basicamente conflitos de professor e turma, indisponibilidades de horário, e agrupamentos/blocos conforme descrito. **Não** foram incorporados outros tipos de restrições que podem existir em timetabling, por exemplo:

- Conflitos de sala de aula (alocar salas específicas e evitar colisão de uso de sala).
- Preferências de horário dos professores (o sistema poderia tentar atender preferências de horário ideais para cada professor, além de apenas indisponibilidade; nosso escopo focou em indisponibilidade e balanceamento de turma, não tanto em preferências finas dos docentes).
- Intervalos de descanso mínimos para professores ou alunos (ex: um professor não quer dar 4 aulas seguidas, ou um aluno precisa ter horário de almoço livre). Embora mencionamos a questão de aulas consecutivas para alunos, não está implementado de forma explícita a menos que colocado manualmente em restrições.
- Regras curriculares complexas, como ordem de aulas (por exemplo, aula teórica deve ocorrer antes da prática na semana) – essas dependências não são tratadas.

Essas limitações significam que o sistema, na versão atual, se restringe a um núcleo de restrições bem definidas. Para muitas instituições, isso basta, mas é possível que algumas regras de negócio específicas não estejam cobertas.

- *Interface de Usuário:* A interação se dá via edição de arquivos CSV e execução de script. Não há, por ora, uma interface gráfica amigável ou integração web para uso mais acessível por parte de pessoal não técnico. Isso foi uma escolha de projeto para viabilizar rapidamente a solução, mas pode ser melhorado futuramente.

- *Dependência de Formatação Correta:* Como muitos softwares baseados em dados, se a entrada não estiver perfeitamente formatada, o sistema pode falhar ou gerar resultados incorretos. Por exemplo, um professor escrito com grafia diferente em duas disciplinas será tratado como dois professores distintos, não detectando conflito real. Há validações para mitigar isso, mas sempre existe a chance de erro humano na preparação dos CSVs. Uma interface mais robusta poderia validar enquanto o usuário insere dados, mas no formato atual isso é uma limitação.

Possibilidades de Extensão:

- *Alocação de Salas:* Uma evolução natural seria incorporar a alocação de salas físicas às aulas. Isso acrescentaria uma camada de conflito (duas aulas não podem ocorrer na mesma sala ao mesmo tempo) e possivelmente restrições do tipo capacidade ou preferência de sala. Esse problema adicional pode ser tratado ampliando o grafo (vértices representando ocupação de sala, ou aulas duplicadas por sala) ou via algoritmos bipartidos para casar aulas e salas após determinar os horários. É viável estender, mas aumenta a complexidade. Uma alternativa é pós-processar a grade gerada atribuindo salas disponíveis, já que o grosso do problema (posicionar horários) estaria resolvido.

- *Melhoria do Algoritmo – Metaheurísticas:* Caso se deseje buscar soluções ainda melhores (por exemplo, minimizar ao máximo o número de janelas vazias para professores, ou usar menos dias se possível), pode-se integrar metaheurísticas como **Simulated Annealing**, **Genetic Algorithms** ou **ILP solvers** (soluções exatas via solvers como Gurobi, CPLEX). Por exemplo, gerar uma solução inicial com nosso algoritmo e então refiná-la com um algoritmo genético que tenta permutar horários para melhorar um certo critério. Isso aumentaria o tempo de processamento, mas poderia encontrar ajustes finos. Nossa arquitetura modular permitiria plugin de um módulo otimização adicional após a coloração inicial, caso desejado.

- *Interface Gráfica/Web:* Desenvolver uma aplicação front-end onde o usuário possa cadastrar disciplinas, professores e restrições via formulários ou importação de planilhas, e acionar a geração da grade com um botão, obtendo a saída diretamente na tela. Isso tornaria o sistema mais acessível a usuários sem conhecimento técnico. Como o núcleo é Python, poderíamos usar frameworks como Flask ou Django para um front-end web simples, ou até uma interface desktop (Tkinter, PyQt) para manipular os inputs e outputs visualmente.

- *Suporte Multicalendário:* Em casos de instituições com calendários diferenciados (por exemplo, graduações e pós-graduações com regimes distintos), o sistema poderia ser estendido para tratar múltiplos conjuntos de horários separados simultaneamente, ou considerar semanas alternadas (a nossa versão considera a semana padrão repetitiva). Extensões para horários bimestrais, semestrais, etc., implicariam em expandir o conceito de cor para incluir semana/semana, mas é algo possível.

- *Personalização de Regras:* Tornar as regras configuráveis sem precisar alterar o código – por exemplo, inserir um arquivo de configuração onde a instituição pode ativar/desativar certos tipos de restrição (como a regra de não mais que 3 aulas consecutivas) ou ajustar parâmetros de balanceamento (quão estrito deve ser o equilíbrio). Isso daria flexibilidade para adaptar a diferentes políticas escolares.

Considerações Finais: Tecnicamente, a solução já demonstra como a teoria de coloração de grafos pode ser aplicada para resolver um problema real de forma automatizada [2]. A modularidade do sistema facilita manutenção e evoluções. No entanto, é importante envolver a equipe acadêmica no

processo de ajustes finais – algoritmos fornecem uma solução base, mas sempre pode haver a necessidade de ajustes manuais mínimos por motivos não capturados formalmente (por exemplo, um professor preferir trocar uma aula de lugar com outro por conveniência pessoal, algo que foge às regras codificadas). O sistema reduz em **ordens de grandeza** o esforço necessário para montar a grade, e deixa qualquer refinamento fino para decisões humanas, se necessário.

Em conclusão, o projeto "Alocação de Horários Acadêmicos com Coloração de Grafos" atende ao objetivo de gerar automaticamente uma grade horária balanceada e sem conflitos, usando métodos combinatórios eficientes. Compreender suas saídas e limitações permite confiá-lo nas operações cotidianas e também vislumbrar melhorias futuras, tornando-o uma ferramenta valiosa para instituições de ensino que buscam otimizar o trabalho de planejamento acadêmico.

---

1 ALOCAÇÃO DE HORÁRIOS E CARGA HORÁRIA DE PROFESSORES NO NOVO ENSINO MÉDIO | Revista de Geopolítica
https://revistageo.com.br/revista/article/view/869

2 3 Graph Coloring Flow Chart | Download Scientific Diagram
https://www.researchgate.net/figure/Graph-Coloring-Flow-Chart_fig4_322131479