Apresentação de uma Heurística para o Problema de Timetabling

Esther Maria da Silveira Wanderley
Bacharelado em Tecnologia da Informação
IMD
Email: esther.wanderley.110@ufrn.edu.br

Thuanny Carvalho Rolim de Albuquerque
Bacharelado em Tecnologia da Informação
IMD
Email: thuanny.albuquerque.344@ufrn.edu.br

***Resumo*—Este trabalho tem como principal objetivo apresentar uma heurística como resolução de um problema de *Timetabling* modelado em grafos. O problema escolhido consiste na distribuição de horários para determinadas disciplinas respeitando as devidas restrições dadas pelo problema. Cada disciplina corresponde a um vértice do grafo e cada restrição será uma aresta. Por ser um problema NP-Difícil, o uso de heurísticas para solucionar o problema é uma prática comum. O método escolhido para a resolução foi a coloração de grafos, a partir de uma heurística criada pelas autoras, de maneira que os horários das disciplinas sejam distribuídos respeitando as devidas restrições. O resultado da resolução do problema será um horário com as respectivas disciplinas.**

Palavras-chave: *Timetabling Problem*, Grafos, Restrições, Coloração, Heurística, NP-Difícil

I. Introdução

O *Timetabling Problem* consiste na otimização da alocação de recursos disponíveis para um dado problema, onde as restrições e objetivos selecionados para o problema serão satisfeitos. Como esse é um problema que, facilmente, pode ser modelado em grafos, é demasiadamente comum a utilização de conceitos e algoritmos relacionados a essa representação de estrutura de dados.

Esse tipo de problema é facilmente aplicado em contextos de alocações de horário, por exemplo: determinar os horários que os trens passam por determinada estação, horários de funcionários em uma empresa, etc. Porém, também é possível aplicá-lo na alocação de determinados tipos de recursos, por exemplo: distribuição de leitos em um hospital por profissional, distribuição de locais para jogos de um time ou campeonato, etc.

Entretanto, a sua principal aplicação é dentro do contexto de alocação de recursos dentro do ambiente acadêmico. Sejam esses recursos salas, professores, alunos, disciplinas ou horários, a modelagem é realizada através do *Timetabling Problem*, na forma de grafos.

Por ser modelado em grafos, conceitos relacionados a essa representação de estrutura de dados são necessários para a resolução do problema. Alguns dos conceitos mais utilizados para esse tipo de solução são Fluxo Máximo e Coloração de Grafos. A partir de algoritmos feitos para esses conceitos, é possível obter uma resolução otimizada do problema inicial.

Portanto, conseguimos perceber que, através da abstração de um dado problema, é possível modelá-lo e resolvê-lo com conceitos de grafos.

No caso do presente trabalho, os recursos serão disciplinas, horários, professores ou salas. Será aprofundado o conceito de coloração de grafos e dos seus respectivos algoritmos aproximativos e heurísticas, a fim de resolver o problema real.

II. Descrição do Problema

No meio acadêmico, o *Timetabling Problem* é uma tarefa complexa que envolve a definição de horários e salas para uma variedade de cursos, disciplinas ou atividades extracurriculares, bem como a alocação de professores para essas disciplinas. O objetivo é criar um cronograma eficaz e capaz de atender às necessidades dos alunos e professores, a fim de otimizar tal resolução.

As restrições podem incluir a disponibilidade de salas e equipamentos específicos, a preferência dos professores em relação a horários e dias da semana, a exigência de espaços maiores para certos cursos, o tempo necessário para a transição entre as aulas, professores que não podem lecionar ao mesmo tempo, conflito de turmas ou disciplinas que possuem o mesmo professor responsável e a necessidade de garantir que os alunos não tenham conflitos de horário entre diferentes disciplinas.

Quanto maior a quantidade de recursos a serem alocados, maior será a complexidade do problema e das restrições que precisam ser consideradas. Além disso, as demandas e necessidades dos alunos e professores podem variar de semestre para semestre, o que torna a tarefa de alocação de horários algo recorrente nas instituições acadêmicas.

Por isso, a resolução eficiente e eficaz do problema de timetabling no meio acadêmico é fundamental para garantir o bom funcionamento das instituições de ensino, a satisfação dos alunos e professores e o sucesso geral do processo educacional.

III. Modelo em Grafos

Nesta parte do projeto, o problema será, de fato, montado e definido. Para isso, segue que: A é o conjunto de aulas

que precisam ser agendadas e H é o conjunto de horários disponíveis para as aulas. A quantidade de salas é um número M. A cardinalidade do conjunto das cores disponíveis c será a cardinalidade dos horários disponíveis para as aulas. Podemos representar o grafo como uma matriz de adjacência, visto que é necessário garantir que os vértices adjacentes estejam coloridos com cores diferentes.

No problema em questão, há restrições. Determinadas aulas não podem ocorrer em determinados horários ou no mesmo horário que algumas outras matérias. Seja o grafo G, tal que cada um de seus vértices é um elemento do conjunto A, ou seja, é uma aula. As arestas de G se dão a partir das restrições, aulas que não podem ocorrer no mesmo horário.

Cada vértice do grafo será representado por uma cor. Essa cor determina o horário em que as aulas ocorrerão.

A partir disso e das arestas de restrição, podemos aplicar um algoritmo de coloração em grafos, que respeite a adjacência dos vértices e as restrições das arestas.

No final, obteremos um grafo colorido com todas as aulas distribuídas de maneira eficiente e respeitando as suas devidas restrições.

Vejamos, agora, um exemplo do problema. Sejam as disciplinas de Matemática, Português, Biologia, Física, História. As restrições são dadas por:

- Matemática não pode ocorrer no mesmo horário que História e Biologia;
- Biologia não pode ocorrer no mesmo horário Física;
- Física não pode ocorrer no mesmo horário que Português.

Na modelagem do nosso problema em grafos, as disciplinas serão os vértices do grafo, enquanto as restrições serão representadas pelas arestas desse mesmo grafo.

Esse grafo será definido da seguinte forma:

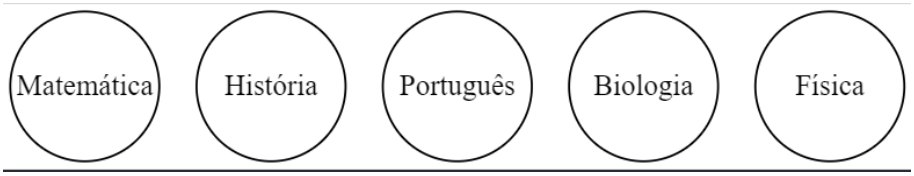

Figura 1: As disciplinas sendo representadas como os vértices do grafo

Agora, acrescentaremos as arestas, tal quais cada uma representa uma restrição.
Adicionando a restrição de que a aula de matemática não pode ocorrer ao mesmo tempo que a aula de história e biologia:

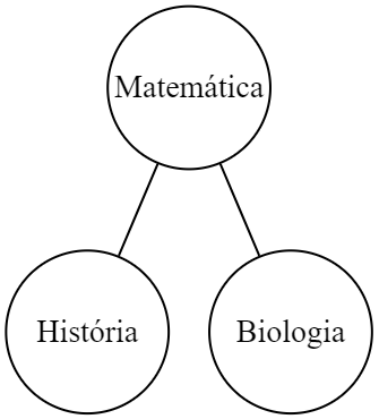

Figura 2: As restrições sendo representadas como as arestas do grafo

Adicionando a restrição que há entre Física e Biologia, obtemos:

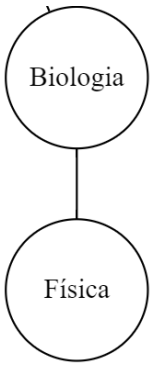

Figura 3: Restrições entre as disciplinas de Biologia e Física

Por último, verifica-se que Física não pode ocorrer no mesmo horário que Português:

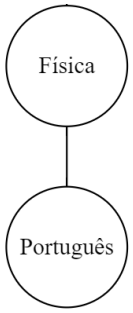

Figura 4: Restrições entre as disciplinas de Português e Física

Ao final, obteremos um grafo representando todas as disciplinas através dos seus vértices e as suas restrições pelas arestas do grafo. Como é possível ver na figura abaixo:

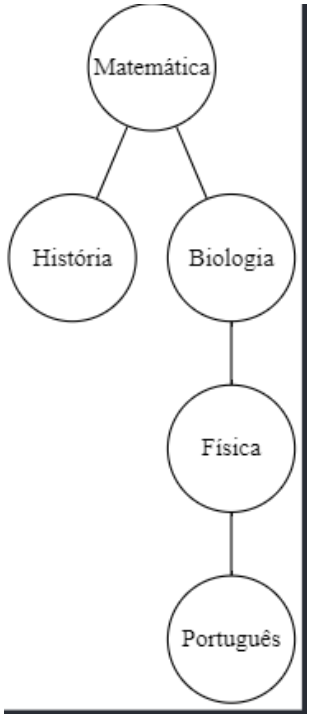

Figura 5: Grafo representando um exemplo do problema real

Dessa forma, obtemos o problema escolhido modelado em um grafo, cujas arestas representam as restrições definidas.

IV. Complexidade do Problema

A fim de mostrarmos a complexidade do *Timetabling Problem* , o modelaremos matematicamente. Segue:
Seja um conjunto finito *H* que representa os horários disponíveis, uma Coleção $\{T_1, T_2, \ldots, T_n\}$, onde $T_i \subseteq H$ (T_i é o conjunto de horários disponíveis do i-ésimo professor, tal que há n professores), uma Coleção $\{C_1, C_2, \ldots, C_m\}$, onde $C_j \subseteq H$ (C_j representa os horários disponíveis da j-ésima sala, tal que há m salas), uma matriz $R_{n\times m}$ de inteiros não-negativos (R_{ij} é o número de horários que o i-ésimo professor é requisitado para usar a j-ésima sala).
A questão é estabelecer se existe uma função de encontro

$$f(i,j,h) : \{1,\ldots,n\} \times \{1,\ldots,m\} \times H \mapsto \{0,1\}$$

(onde $f(i,j,h) = 1$ se e somente se o professor i usar a sala j no horário h) tal que:

(a) $f(i,j,h) = 1 \implies h \in (T_i \cap C_j)$;

(b) $\sum_{h\in H} f(i,j,h) = R_{ij}$, $\forall\ 1 \leqq i \leqq n \wedge 1 \leqq j \leqq m$;

(c) $\sum_{i=1}^{n} f(i,j,h) \leqq 1$, $\forall\ 1 \leqq j \leqq m \wedge h \in H$;

(d) $\sum_{j=1}^{m} f(i,j,h) \leqq 1$, $\forall\ 1 \leqq i \leqq n \wedge h \in H$.

De maneira que:

(a) garante que um encontro ocorre apenas quando tanto o professor quanto a sala estão disponíveis.

(b) garante que o número de encontros entre o professor i e a turma j é o número R_{ij}.

(c) Garante que nenhuma sala tem mais de um professor ao mesmo tempo.

(d) Garante que nenhum professor esteja em mais de uma sala simultaneamente.

Um professor i é chamado de k-professor se $|T_i| = k$. Um professor i é chamado de *Fechado* se

$$|T_i| = \sum_{j=1}^{m} R_{ij},$$

isto é, ele deve ensinar toda vez que estiver disponível.

Como definimos esse problema matematicamente, faremos a verificação da sua complexidade. Neste caso, especificamente, queremos demonstrar que o *Timetabling Problem* é NP-Difícil.

Por definição, um problema $\prod$ é dito NP-Difícil se todo problema $\prod'$ em NP se reduz polinomialmente a $\prod$.
Para essa demonstração, utilizaremos o problema SAT, que é NP-Completo (Teorema de Cook), logo é, também NP, por definição.
Verificaremos agora se o problema SAT se reduz polinomialmente ao *Timetabling Problem*. Já é demonstrado que o Problema 3-SAT se reduz polinomialmente ao *Timetabling Problem* com restrições, onde a cardinalidade do conjunto de horários é igual a 3, por Even, Shimon & Itai, Alon & Shamir, Adi. (1976).
Como o problema 3-SAT é uma variação do Problema SAT e também é um problema NP-Completo, este pertence à classe NP também. A mesma coisa é válida para o RTT (*Restricted Timetable Problem*) que, por ser NP-Completo, é NP-Difícil e uma variação do *Timetabling Problem*. Com essas definições, conseguimos concluir que o problema SAT se reduz polinomialmente ao *Timetabling Problem* de maneira análoga à demonstração supracitada. Logo, o *Timetabling Problem* é NP-Difícil.

V. Heurísticas construtivas e algoritmos aproximativos já apresentados para o problema

Ao longo dos anos foram estudadas e implementadas soluções, sejam com heurísticas, algoritmos aproximativos ou gulosos.

Nesta seção, buscaremos apresentar algumas dessas soluções.

A. Algoritmo Genético para o Timetabling Problem

A aplicação de algoritmos genéticos como heurísticas para problemas de otimização, como o próprio *Timetable Problem*, tornou-se uma área de pesquisa muito ativa em meados de

1980. Não há um único autor ou grupo responsável por isso, são milhares de contribuições ao longo dos anos que elevou o patamar dessa área de pesquisa para o que é hoje.

No contexto do *Timetable Problem*, o uso do algoritmo genético pode se dar a partir de:

1) Como representar a solução? Escolha de como a solução encontrada será representada, seja uma lista, um vector, um arquivo json, etc.

2) Fitness Function Essa função é a responsável por realizar a avaliação de cada solução candidata. No contexto, ela atribuiria uma pontuação baixa para solução que não respeitasse as restrições e uma pontuação alta para uma solução que respeita a restrição e é otimizada.

3) Geração da população inicial Nessa parte, são geradas diversas soluções a partir de heurísticas construtivas ou apenas soluções aleatórias. Essa população de soluções serão passadas e testadas ao longo das gerações do algoritmo.

4) Os operadores genéticos Aqui se diz respeito aos operadores bases da solução que será produzida no final.

   - Seleção - responsável por determinar os pais da próxima geração.
   - Crossover - realiza o cruzamento entre duas soluções pais, a fim de gerar uma nova solução baseada nas duas soluções "inspiradoras".
   - Mutação - acrescenta alterações aleatórias em soluções já existentes. Essas alterações combinadas com outras soluções geram outras soluções.

5) Critérios de parada Decidir até que ponto vai a execução do algoritmo, seja por número de gerações geradas, tempo ou até encontrar uma solução satisfatória.
6) Laço Evolutivo Laço responsável pela aplicação dos operadores genéticos e pela avaliação da solução até o critério de parada ser satisfeito.
7) Melhor solução encontrada A melhor solução encontrada será decidida a partir da Fitness Function, onde é determinada pelo valor que foi dado por ela. Dessa forma, a solução com a maior pontuação atribuída após a condição de parada ser satisfeita, será a a solução retornada pelo algoritmo, no formato escolhido desejado.

Comumente, são realizados vários testes em cima de um algoritmo genético até ser entregue uma solução que melhor satisfaça o problema.
As variações do algoritmo vão de mudanças nos parâmetros das operações genéticas, até pré-determinação da população inicial.

Uma implementação possível é a do Achini Kumari Herath (2017), onde ele realiza a comparação entre duas variações de algoritmos genéticos aplicado a uma restrição do *Timetable Problem*, encontrando a solução que satisfaz a restrição.

B. Adaptive Tabu Search Algorithm

Nesse tipo de solução, Fouad H. Awad, Ali Al-kubaisi, and Maha Mahmood (2022) utilizam a técnica de busca Tabu para conseguir encontrar a melhor solução para o problema.

Esse tipo de solução é dividido em algoritmo de construção e melhoria. O algoritmo de construção se baseia em um algoritmo com menor saturação, mas que entregue uma solução possível que será usada como uma solução inicial, enquanto o algoritmo de melhoria, nesse caso o algoritmo de busca tabu é utilizado para melhorar a solução inicial.
Nessa parte, são utilizadas quatro estruturas que são chamadas de vizinhanças, cada uma é responsável por uma ação: (N1) escolher um curso aleatoriamente e o coloca em um horário possível com o menor custo, (N2) selecionar aleatoriamente uma sala e aloca dois cursos, aleatoriamente, nessa sala, depois troca os horários desse cursos, (N3) Escolher aleatoriamente dois horários. Em seguida, trocar os horários e (N4)selecionar aleatoriamente um horário específico e troca-o por outro horário, dentro da faixa de 0 a 44, gerando o menor custo de penalidade.
Na seguida parte do algoritmo, voltada para a melhoria, São realizadas movimentações nas vizinhanças (N1 e/ou N2) para alcançar a viabilidade. O N1 é utilizado por um número específico de repetições. Em seguida, o algoritmo é encerrado se uma solução viável for alcançada. Caso contrário, o algoritmo continua utilizando a estrutura de bairro N2 por um número específico de repetições. Note que todos os estudos de caso são examinados antes que o algoritmo de melhoria seja utilizado, uma vez que as soluções precisam ser tornadas viáveis primeiro.

O autor apresenta uma adaptação do do algoritmo de busca tabu,onde ele depende da lista tabu que armazena as soluções e do custo das penalidades, que ele utilizará para melhorar a solução inicial e depois aplicar as vizinhanças N3 e N4.

Esse algoritmo apresentou melhoria em comparação com implementações comuns da busca tabu.

C. Fluxo em Redes

Nesse tipo de resolução, o problema é modelado em grafos e pode ser resolvido com algoritmos de fluxos em redes.

Moritz Willnauer (2020) utilizou os algoritmos de Ford-Fulkerson,Edmonds-Karp e algoritmo Push-Relabel para uma versão do *Timetable Problem*. Ele conseguiu modelar o problema matematicamente como um problema de fluxo máximo e realizou a comparação entre os 3 algoritmos implementados. A complexidade entre eles é muito parecida, $O(V^2E)$, e aumentam exponencialmente de acordo com a quantidade de arestas e vértices do grafo.
Embora os 3 algoritmos encontrem respostas para o problema, nem sempre a resposta será ótima.

D. Coloração

Outro tipo de heurística aplicada ao problema é a de Welsh-Powell, que consiste em um algoritmo guloso de coloração em grafos, onde ele ordena os graus dos vértices e vai colorindo os vértices de maior grau primeiro.
A complexidade desse algoritmo é $O(n^2)$.
Já foi realizado um trabalho pelas autoras sobre essa heurística e iremos tratar no projeto atual uma resolução com coloração e comparar com o desempenho do trabalho anterior onde foi estudado o algoritmo de Welsh-Powell

E. Problema Real: Alocação de Horários no Meio Acadêmico

A alocação de horários no meio acadêmico é um desafio complexo enfrentado por instituições de ensino superior em cada semestre. A construção de uma grade horária satisfatória, livre de conflitos, que atenda às restrições e necessidades das disciplinas e dos envolvidos, é de extrema importância para garantir um funcionamento eficiente das atividades acadêmicas. Esse problema envolve a distribuição de horários para disciplinas, considerando a disponibilidade de salas, professores, recursos e as preferências dos estudantes.

Diversos estudos têm sido realizados para abordar o problema de alocação de horários no meio acadêmico. Esses estudos visam desenvolver métodos e algoritmos eficientes que considerem as múltiplas restrições envolvidas, como a disponibilidade de recursos, a minimização de conflitos de horários, a maximização da utilização dos recursos e a satisfação das preferências dos estudantes.

F. Modelo em Grafos para o Timetabling

O modelo em grafos é amplamente utilizado para representar e resolver o problema de *timetabling*. Nesse modelo, as disciplinas, professores e salas são representados como vértices do grafo, enquanto as restrições e conflitos são representados por arestas. Essa representação permite uma abordagem estruturada e computacionalmente eficiente para a resolução do problema.

No modelo em grafos, a coloração dos vértices é utilizada para representar a alocação de horários. Cada cor atribuída a um vértice representa um horário específico, e vértices adjacentes (conectados por uma aresta) devem ter cores diferentes, garantindo que não haja conflitos de horários entre as disciplinas ou recursos alocados. Assim, a resolução do problema de *timetabling* se resume à atribuição de cores aos vértices do grafo.

Existem diferentes algoritmos para a coloração de grafos no contexto do *timetabling*. Algoritmos clássicos, como o Algoritmo de Coloração Sequencial e o Algoritmo de *Welsh-Powell*, têm sido amplamente estudados e aplicados. Além disso, abordagens mais avançadas, incluindo técnicas de busca exaustiva, heurísticas e metaheurísticas, têm sido exploradas para obter soluções mais otimizadas e eficientes.

A literatura científica apresenta uma variedade de estudos e abordagens para o *timetabling* utilizando o modelo em grafos. Esses estudos buscam aprimorar a qualidade e a eficiência das soluções, considerando diferentes restrições e objetivos específicos de cada contexto.

Em suma, a aplicação do modelo em grafos no *timetabling* tem se mostrado uma abordagem promissora e eficiente para resolver o desafiador problema de alocação de horários no meio acadêmico.

VI. Casos de teste ou geradores de casos de teste encontrados na literatura

Os casos de teste escolhidos encontra-se em: https://mat.tepper.cmu.edu/COLOR/instances.html . Estes, possuem como referências:

- REG: (De Gary Lewandowski (gary@cs.wisc.edu)) Problema baseado na alocação de registros para variáveis em códigos reais.
- LEI: (De Craig Morgenstern (morgenst@riogrande.cs.tcu.edu)) Grafos de Leighton com tamanho de coloração garantido. A referência é F.T. Leighton, Journal of Research of the National Bureau of Standards, 84: 489–505 (1979).
- SCH: (De Gary Lewandowski (lewandow@cs.wisc.edu)) Grafos de marcar aulas, com e sem salas de estudo.
- LAT: (De Gary Lewandowski (lewandow@cs.wisc.edu)) Problema do quadrado latino.
- SGB: (De Michael Trick (trick@cmu.edu)) Grafos da base de grafos de Stanford de Donald Knuth (Donald Knuth's Stanford GraphBase). Podem ser divididos em: Grafos de livro. Dada uma obra da literatura, um grafo em que cada nó representa um personagem é criado. Dois nós são conectados por uma aresta se os personagens correspondentes se encontram no livro. Knuth cria os grafos para cinco obras clássicas: Anna Karenina de Tolstoy (anna), David Copperfield de Dickens (david), A Ilíada de Homero (homer), Huckleberry Finn (huck) de Twain e Os Miseráveis de Hugo (jean). Grafos de jogos. Um grafo representando os jogos de uma temporada de futebol em uma faculdade pode ser um grafo em que os nós representam cada equipe da faculdade. Dois times se conectam por uma aresta se eles jogaram um contra o outro na temporada. Knuth dá o grafo para a temporada de futebol universitário de 1990. Grafos de milhas. Estes grafo são semelhantes a grafos geométricos em que os nós são colocados no espaço e dois nós se conectam se estão próximos o suficiente. Estes grafos, no entanto, não são aleatórios. Os nós são um conjunto de cidades dos Estados Unidos e a distância entre eles é dada pela distância em milhas de 1947. Estes grafos também existem graças a

Knuth. Grafos de dama. Dado um tabuleiro de xadrez n por n, um grafo de dama é um grafo de n^2 nós, cada um correspondendo a um quadrado do tabuleiro. Dois nós se conectam por uma aresta se o quadrado correspondente se encontra na mesma linha, coluna ou diagonal. Diferente de alguns outros grafos, o problema de coloração neste grafo tem uma interpretação natural: Dado tal tabuleiro, é possível colocar n conjuntos de n damas no tabuleiro de modo que não haja duas damas do mesmo conjunto na mesma linha, coluna ou diagonal? A resposta é sim se e somente se o grafo tem número de coloração n. Martin Gardner afirma sem provas que este é o caso se e somente se n não é divisível nem por 2 nem por 3. Em qualquer caso, a clique máxima no grafo não é maior que n e o valor de coloração não é menor que n.

- MYC: (De Michael Trick (trick@cmu.edu)) Grafos baseados na transformação de Mycielski. Estes grafos são difíceis de resolver porque eles são "triangle free"(clique número 2), mas o número de coloração aumenta o tamanho do problema.

VII. Proposta de solução

Como desenvolvido ao longo do trabalho, será utilizada a coloração de grafos para resolver o dito problema:

- Matemática não pode ocorrer no mesmo horário que História e Biologia;
- Biologia não pode ocorrer no mesmo horário Física;
- Física não pode ocorrer no mesmo horário que Português.

O grafo gerado anteriormente, reorganizando os vértices, foi:

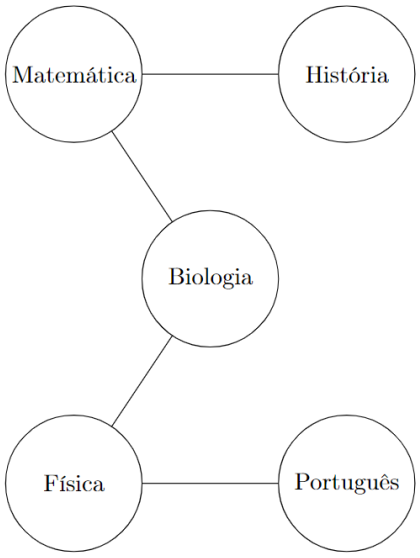

Figura 6: Grafo reorganizado

Agora, para a proposta de solução, devemos colorir os vértices de modo que vértices adjacentes tenham cores distintas. Assim, teremos aulas que não podem ocorrer no mesmo horário (vértices adjacentes) em horários (cores) distintos. A nossa proposta de algoritmo seleciona, inicialmente, um dos vértices de maior grau. Para fins ilustrativos, selecionemos, como vértice inicial, Matemática. A seguir, criamos *partições do conjunto de vértices* para cada cor disponível. Neste caso, teremos as partições H_1 para a cor 1 e H_2 para a cor 2. Caso a quantidade de horários não seja especificada, são criadas n partições, sendo n o número de vértices. Todas elas são iniciadas vazias. Coloriremos o vértice previamente selecionado (Matemática) da primeira cor disponível.

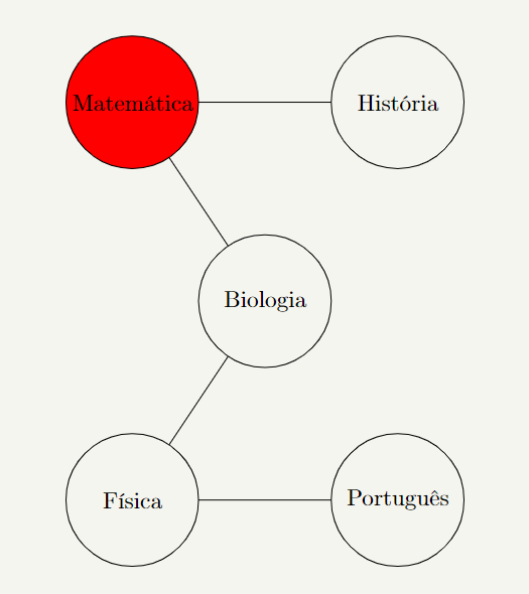

Figura 7: Grafo com o primeiro vértice colorido

Então, para cada vértice, vamos colori-lo da primeira cor disponível. Uma cor j só estará disponível para o vértice w se, e somente se, $H_j \cap \text{Adj}(w) = \emptyset$. Coloriremos, primeiro, os vértices adjacentes ao inicial e assim seguiremos, em um esquema de busca em largura. Assim, coloriremos, primeiro, os vértices adjacentes ao já colorido da primeira cor disponível para eles, obtendo a seguinte coloração:

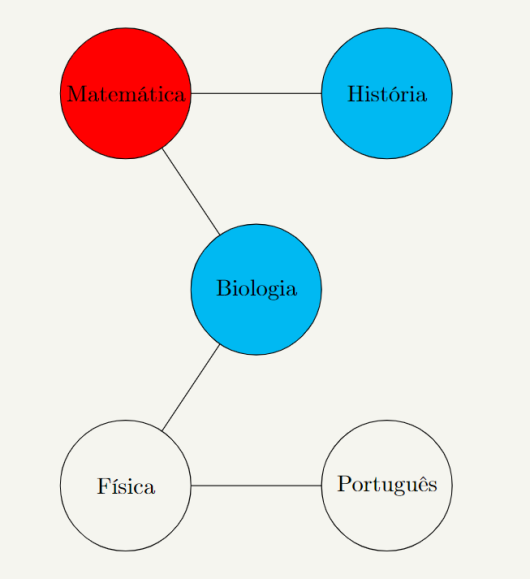

Figura 8: Grafo com os dois primeiros níveis coloridos

Continuando o esquema de busca em largura, coloriremos os vértices não-coloridos adjacentes a História, que, no caso, não existem, e, em seguida, os adjacentes a Biologia. Assim, teremos:

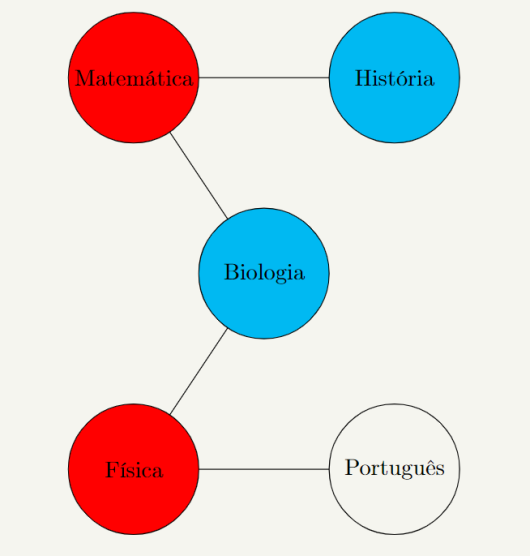

Figura 9: Grafo com os três primeiros níveis coloridos

Por fim, percorreremos os não-coloridos adjacentes a Física, colorindo, então, o grafo inteiro. Ao final da execução de nossa heurística, teremos o seguinte grafo:

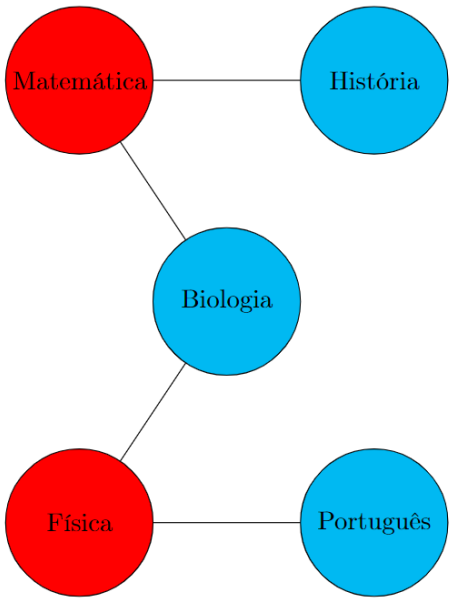

Figura 10: Grafo após algoritmo de Albuquerque-Wanderley

E as partições:
$H_1 = \{\text{Matemática, Física}\}$
$H_2 = \{\text{Biologia, Português, História}\}$
Por fim, o algoritmo deve retornar a quantidade de salas necessárias para tal alocação, ou seja, a maior quantidade de aulas que devem ocorrer num mesmo horário. Assim, devemos retornar a maior cardinalidade das partições geradas, que, nesse caso, será a de H_2.

A. *pseudocódigo:*

Segue o pseudocódigo para a solução proposta (Nomeamos a heurística com os sobrenomes da autoras):

Algorithm 1: Albuquerque-Wanderley

Input: Um grafo qualquer $G(A, R)$ e o número de cores c
Result: Número de horários que satisfazem o problema
fila Q;
$v \leftarrow$ aula com maior número de restrições (vértice de maior grau);
for $i \leftarrow 0$ ***to*** c **do**
  //Para cada conjunto de vértices de cor i
  $H_i \leftarrow \{\}$;
end
$H_0 \leftarrow H_0 \cup \{v\}$;
Q.push(v);
while Q *não estiver vazia* **do**
  $v \leftarrow Q$.dequeue();
  for w *adjacente a* v **do**
    if w *não está colorido* **then**
      $k \leftarrow \min\{j | Adj(w) \cap H_j = \emptyset\}$;
      $H_k \leftarrow H_k \cup \{w\}$;
      Q.push(w);
    end
  end
end
return $\max\{\#H_j\}$;

B. *Análise de complexidade:*

Analisando a complexidade do algoritmo, segue:
Primeiro, analisaremos a complexidade da obtenção do vértice de maior grau. Por se tratar de uma lista não-ordenada, podemos fazer uma busca linear para encontrar o vértice de maior grau com a complexidade de $O(n)$.

Após isso, há a inicialização do conjunto de cores vazios. Essa etapa depende do número de cores disponíveis. Portanto, há uma complexidade $O(c)$, em que c representa o número de cores disponíveis.

O próximo passo consiste na atribuição de cor ao primeiro vértice. Esse passo tem complexidade $O(1)$, visto que é constante.

O próximo passo se dá pela seleção de vértices a serem coloridos. O laço segue um esquema de BFS (Breadth-First Search), então a complexidade, desconsiderando a etapa de coloração, será $O(n+m)$, ou seja, o maior entre o número de vértices e o de arestas. Já para colorir, a escolha da cor se dá em $O(n)$. Assim, a complexidade da etapa de coloração será de $O(n^2 + mn)$.

Dessa forma, para obter a complexidade total vamos somar as complexidades de todos os passos:

$$O(n) + O(c) + O(1) + O(n^2 + mn) = O(n^2 + mn)$$

Portanto, a complexidade total da heurística proposta é de $O(n^2 + mn)$.

VIII. Experimentos

Agora serão apresentados os resultados dos casos de teste utilizados para testar o algoritmo em si.

A. *Metodologia*

1) Ferramentas Utilizadas: Computador:

Computador	Especificações
Samsung 550XCJ/ 550XCR	Processador Intel (R) Core (TM) i3-10110U 2.59 GHz 2 x 4 GB DDR4 1333 MHz SSD 256 GB Intel (R) UHD Graphics

Para a implementação dos algoritmos foi usado o sistema operacional Windows, mais especificamente o Windows 11 Home Single Language.
Já nos testes, foi utilizado o terminal WSL com comandos do Ubuntu.

2) Ferramentas de programação: Tanto o algoritmo escolhido quanto a heurística autoral foram implementados na linguagem C++, padrão ISO/IEC 14882:2014, ou simplesmente C++14.

Eles foram compilados utilizando o WSL pelo g++ na versão 9.4.0 através da linha:

g++ -Wall -std=c++14 src/main.cpp src/manager.cpp src/parser.cpp -o exec

A biblioteca *chrono* foi responsável pelas medições de tempo.

3) Materiais para os testes: Para os testes, foi utilizada uma biblioteca de instâncias de grafos para serem coloridos. Essa biblioteca se encontra disponível em: https://mat.tepper.cmu.edu/COLOR/instances.html

4) Orientações para a realização dos testes: Para a realização dos testes foi utilizado o repositório: https://github.com/stellamaris-22/unidade-2-paa-esther-e-thuanny.git, sendo possível cloná-lo com o comando:

git clone https://github.com/stellamaris-22/unidade-2-paa-esther-e-thuanny.git

Após ter o repositório baixado em sua máquina, é possível compilá-lo pelo terminal utilizando o comando:

g++ -Wall -std=c++14 src/main.cpp src/manager.cpp src/parser.cpp -o exec

Onde “exec” é o nome do seu executável.

Para, de fato, executar o código, utilize o comando:

./exec

Depois, irá aparecer a seguinte tela:

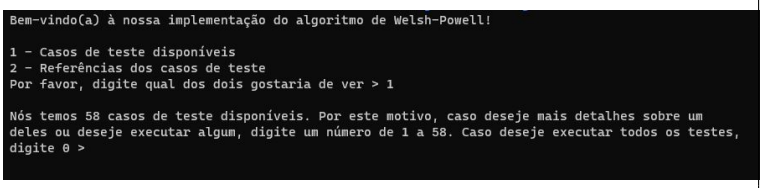

Figura 11: Tela de Inicialização para os testes

A partir daí, basta selecionar a opção desejada.

Caso queira adicionar algum teste, adicione o arquivo desejado na pasta "test_lib", e acrescente o nome do arquivo na linha 22 do arquivo "main.cpp", no vetor de strings chamado "file_names". Depois, é só executar o código normalmente e o resultado estará no arquivo "time_output.txt".

B. Resultados:

Nesta subseção será apresentada a tabela com os resultados obtidos, mostrando a instância do teste, quantidade de vértices, arestas, cores utilizadas e de tempo de execução.

- Tabela com o resultado dos testes para o algoritmo autoral

Instância	Vértices	Arestas	Cores	Tempo em *ms*
anna.col	138	986	8	0
david.col	87	812	11	0
fpsol2.i.1.col	496	11654	33	2
fpsol2.i.2.col	451	8691	29	3
fpsol2.i.3.col	425	8688	29	4
games120.col	120	1276	7	0
homer.col	561	3258	12	11
huck.col	74	602	10	0
inithx.i.1.col	864	18707	35	17
inithx.i.2.col	645	13979	31	6
inithx.i.3.col	621	13969	31	7
latin_square_10.col	900	307350	204	143
jean.col	80	508	7	0
le450_5a.col	450	5714	11	2
le450_5b.col	450	5734	10	3
le450_5c.col	450	9803	12	4
le450_5d.col	450	9757	11	4
le450_15a.col	450	8168	15	2
le450_15b.col	450	8169	15	2
le450_15c.col	450	16680	20	4
le450_15d.col	450	16750	19	4
le450_25a.col	450	8260	21	3
le450_25b.col	450	8263	16	2
le450_25c.col	450	17343	27	5
le450_25d.col	450	17425	20	3
miles250.col	128	774	7	0
miles500.col	128	2340	20	0
miles750.col	128	4226	19	1
miles1000.col	128	6432	11	1
miles1500.col	128	10396	44	2
mulsol.i.1.col	197	3925	34	1
mulsol.i.2.col	188	3885	31	1
mulsol.i.3.col	184	3916	31	1
mulsol.i.4.col	185	3946	31	1
mulsol.i.5.col	186	3973	31	1
myciel2.col	5	6	3	0
myciel3.col	11	20	4	0
myciel4.col	23	71	5	0
myciel5.col	47	236	6	0
myciel6.col	95	755	7	0
myciel7.col	191	2360	8	0
queen5_5.col	25	320	6	0
queen6_6.col	36	580	8	0
queen7_7.col	49	952	9	0
queen8_8.col	64	1456	12	0
queen8_12.col	96	2736	14	0
queen9_9.col	81	2112	12	0
queen10_10.col	100	2940	15	0
queen11_11.col	121	3960	14	0
queen12_12.col	144	5192	18	0
queen13_13.col	169	6656	20	1
queen14_14.col	196	8372	20	1
queen15_15.col	225	10360	20	1
queen16_16.col	256	12640	23	2
school1.col	385	19095	40	5
school1_nsh.col	352	14612	36	5
zeroin.i.1.col	211	4100	35	1
zeroin.i.2.col	211	3541	30	1
zeroin.i.3.col	206	3540	30	1

IX. Comparação com resultados de heurísticas anteriores

Nessa seção, faremos a comparação com os resultados obtidos no trabalho anterior, em que implementamos e aplicamos no mesmo banco de instâncias a heurística de *Welsh-Powell*.

Iremos apresentar, novamente, ambos algoritmos.
Segue o algoritmo implementado anteriormente de Welsh-Powell:

Algorithm 2: Welsh-Powell

Input: Um grafo qualquer $G(A, R)$ e o número de cores c
Result: Número de horários que satisfazem o problema
Ordene A de modo que grau$(n_i) \geq$ grau(n_{i+1});
for $i \leftarrow 0$ ***to*** c **do**
  //Para cada conjunto de vértices de cor i
  $H_i \leftarrow \{\}$;
end
$H_0 \leftarrow H_0 \cup \{n_0\}$;
for $i \leftarrow 1$ ***to*** n **do**
  $k \leftarrow \min\{j | Adj(n_i) \cap H_j = \emptyset\}$;
  $H_k \leftarrow H_k \cup \{n_i\}$;
end
return max$\{\#H_j\}$;

Agora, o algoritmo deste trabalho, que foi nomeado com o sobrenome das autoras:

Algorithm 3: Albuquerque-Wanderley

Input: Um grafo qualquer $G(A, R)$ e o número de cores c
Result: Número de horários que satisfazem o problema
fila Q;
$v \leftarrow$ aula com maior número de restrições (vértice de maior grau);
for $i \leftarrow 0$ ***to*** c **do**
  //Para cada conjunto de vértices de cor i
  $H_i \leftarrow \{\}$;
end
$H_0 \leftarrow H_0 \cup \{v\}$;
Q.push(v);
while Q *não estiver vazia* **do**
  $v \leftarrow Q$.dequeue();
  for w *adjacente a* v **do**
    if w *não está colorido* **then**
      $k \leftarrow \min\{j | Adj(w) \cap H_j = \emptyset\}$;
      $H_k \leftarrow H_k \cup \{w\}$;
      Q.push(w);
    end
  end
end
return max$\{\#H_j\}$;

A. Comparação de resultados

Como é possível verificar na Tabela 1, o novo algoritmo obteve um desempenho consideravelmente melhor quando comparado com o algoritmo implementado no projeto anterior, principalmente nos grafos com maiores quantidade de vértices. Isso se dá, pois ao se aplicar uma BFS no vértice, o algoritmo se torna menos guloso e permite que o algoritmo colora outros vértices primeiro, em relação ao algoritmo Welshell-Powell, trazendo a menor opção de cor possível para aquele vértice. Como isso vai se aplicando ao longo do algoritmo, a chance de um vértice ser colorido com a menor cor possível é maior do que no Welshell-Powell, resultando numa possível minimização do número de cores.

Essas diferenças ficam ainda mais visíveis na instância school1.col, onde há uma diferença de 279 cores, representando uma melhora significativa de 88%. Na instância, latin_square_10.col, houve uma diferença de 687 cores com uma melhora de 78%. No homer.col, a diferença foi de 38 cores e a melhora foi de 95%.

Em contrapartida, houve algumas instâncias como fpsol2.i.2.col, fpsol2.i.3.col, mulsol.i.2.col,mulsol.i.3.col, mulsol.i.4.col, mulsol.i.5.col, myciel2.col,zeroin.i.2.col, zeroin.i.3.col em que o algoritmo de Welshell-Powell obteve um desempenho minimamente melhor.

As instâncias inithx.i.2.col, inithx.i.3.col apesar de terem obtido o mesmo resultado no número de cores, o tempo de execução do algoritmo Albuquerque-Wanderley foi melhor.

X. Conclusão

Neste artigo, abordamos o problema do timetabling no contexto acadêmico e apresentamos um modelo baseado em grafos. Mostramos algumas heurísticas e trabalhos já feitos na área do problema e sugerimos uma nova abordagem algorítmica para a solução com coloração.
Implementamos essa abordagem e a comparamos com a implementação feita pelas autoras em um trabalho anterior do algoritmo de Welsh-Powell.
Obtivemos resultados relativamente otimistas, abrindo portas para futuros testes e melhorias no algoritmo, a fim de fornecer a melhor resposta possível para esse tipo de problema.

XI. Referências bibliográficas

- Tan, J. S., Goh, S. L., Kendall, G., & Sabar, N. R. (2021). A survey of the state-of-the-art of optimisation methodologies in school timetabling problems. Expert Systems with Applications, 165, 113943.
- "Timetabling Problem"
- "Graph Coloring"
- Babaei, H., Karimpour, J., Hadidi, A. (2015). A survey of approaches for university course timetabling problem. Computers & Industrial Engineering, 86, 43-59.

- Ceschia, S., Di Gaspero, L., & Schaerf, A. (2022). Educational timetabling: Problems, benchmarks, and state-of-the-art results. European Journal of Operational Research.
- de Lima, A. M., Carmo, R. (2018). Exact algorithms for the graph coloring problem. Revista de Informática Teórica e Aplicada, 25(4), 57-73.
- Awad, F., Al-kubaisi, A. and Mahmood, M. (2022) Large-scale timetabling problems with adaptive tabu search. Journal of Intelligent Systems, Vol. 31 (Issue 1), pp. 168-176. https://doi.org/10.1515/jisys-2022-0003
- Even, Shimon & Itai, Alon & Shamir, Adi. (1976). On the Complexity of Timetable and Multicommodity Flow Problems. SIAM J. Comput.. 5. 691-703. 10.1137/0205048.
- Herath, Achini Kumari, "Genetic Algorithm For University Course Timetabling Problem"(2017). Electronic Theses and Dissertations. 443.]
- Moritz Willnauer, "*Modelling and Solving a Scheduling Problem by Max-Flow*", Research Institute for Symbolic Computation (RISC), 2020.

Instância	Vértices	Arestas	CoresWP	TempoWP em *ms*	CoresAW	TempoAW
anna.col	138	986	40	2	8	0
david.col	87	812	50	1	11	0
fpsol2.i.1.col	496	11654	51	33	33	2
fpsol2.i.2.col	451	8691	26	31	29	3
fpsol2.i.3.col	425	8688	26	28	29	4
games120.col	120	1276	11	2	7	0
homer.col	561	3258	40	24	2	11
huck.col	74	602	15	0	10	0
inithx.i.1.col	864	18707	51	81	5	17
inithx.i.2.col	645	13979	31	54	31	6
inithx.i.3.col	621	13969	31	53	31	7
latin_square_10.col	900	307350	891	3282	204	143
jean.col	80	508	11	0	7	0
le450_5a.col	450	5714	30	62	11	2
le450_5b.col	450	5734	31	28	10	3
le450_5c.col	450	9803	43	41	12	4
le450_5d.col	450	9757	60	38	11	4
le450_15a.col	450	8168	62	32	15	2
le450_15b.col	450	8169	45	34	15	2
le450_15c.col	450	16680	77	64	20	4
le450_15d.col	450	16750	80	64	19	4
le450_25a.col	450	8260	75	31	21	3
le450_25b.col	450	8263	55	34	16	2
le450_25c.col	450	17343	79	71	27	5
le450_25d.col	450	17425	99	71	20	3
miles250.col	128	774	10	1	7	0
miles500.col	128	2340	28	2	20	0
miles750.col	128	4226	54	3	19	1
miles1000.col	128	6432	85	5	11	1
miles1500.col	128	10396	104	10	44	2
mulsol.i.1.col	197	3925	36	5	34	1
mulsol.i.2.col	188	3885	28	6	31	1
mulsol.i.3.col	184	3916	28	6	31	1
mulsol.i.4.col	185	3946	28	6	31	1
mulsol.i.5.col	186	3973	28	6	31	1
myciel2.col	5	6	2	0	3	0
myciel3.col	11	20	5	0	4	0
myciel4.col	23	71	11	0	5	0
myciel5.col	47	236	23	0	6	0
myciel6.col	95	755	47	2	7	0
myciel7.col	191	2360	95	11	8	0
queen5_5.col	25	320	12	0	6	0
queen6_6.col	36	580	15	0	8	0
queen7_7.col	49	952	18	0	9	0
queen8_8.col	64	1456	21	0	12	0
queen8_12.col	96	2736	27	2	14	0
queen9_9.col	81	2112	25	1	12	0
queen10_10.col	100	2940	29	2	15	0
queen11_11.col	121	3960	31	3	14	0
queen12_12.col	144	5192	34	6	18	0
queen13_13.col	169	6656	38	8	20	1
queen14_14.col	196	8372	41	12	20	1
queen15_15.col	225	10360	46	15	20	1
queen16_16.col	256	12640	49	21	23	2
school1.col	385	19095	319	109	40	5
school1_nsh.col	3528	14612	195	47	36	5
zeroin.i.1.col	211	4100	48	6	35	1
zeroin.i.2.col	211	3541	27	5	30	1
zeroin.i.3.col	206	3540	27	5	30	1

Tabela I: Tabela de comparação ente o algoritmo autoral e o algoritmo Welsh-Powell que foi implementado anteriormente