# Apresentação de um Algoritmo Exato 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 um algoritmo exato 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 algoritmos exatos para solucionar o problema é uma prática que vem avançando cada vez mais nas pesquisas. O método escolhido para a resolução foi a coloração de grafos, a partir de um algoritmo criada pelas autoras, de maneira que os horários das disciplinas sejam distribuídos respeitando as devidas restrições. Será gerada todas as colorações possíveis para o problema, a fim de encontrarmos a melhor.**

**Palavras-chave: *Timetabling Problem*, Grafos, Restrições, Coloração, Algoritmo Exato, 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 exatos, a fim de mostrar uma possível soluçaõ para 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 lista 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.

Os vértices desse grafo serão definidos da seguinte forma:


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

Agora, acrescentaremos as arestas, tais que 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:

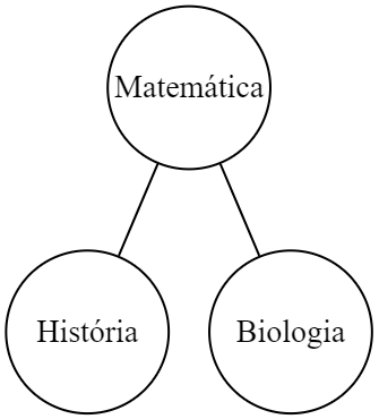


Figura 2: As restrições sendo representadas como as arestas do grafo

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

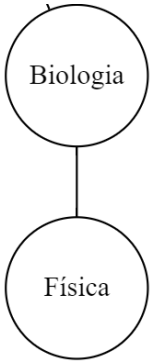


Figura 3: Restrições entre as disciplinas de Biologia e Física

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

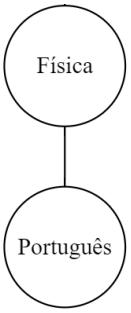


Figura 4: Restrições entre as disciplinas de Português e Física

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:

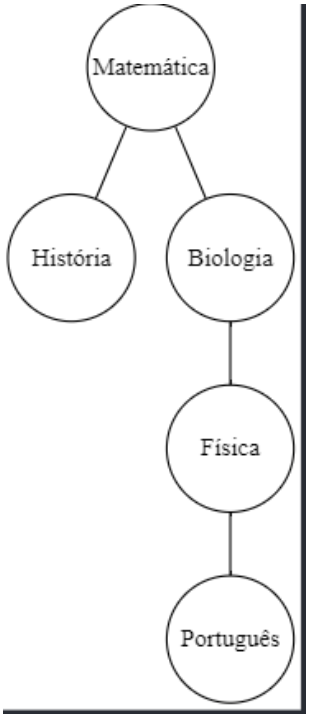


Figura 5: Grafo representando um exemplo do problema real

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, se 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* resolvido com coloraçaõ de grafos é NP-Difícil.

## V. Estado-da-Arte de algoritmos exatos para o problema de Coloração

Nesta seção, mostraremos o Estado-da-Arte dos algoritmos exatos para o problema de Coloração em grafos. Atualmente, há algoritmos exatos apresentados utilizando as técnicas de Programação Dinâmica e Branch-and-Bound [7].

### A. Programação Dinâmica:

Quando se trata de programação dinâmica para o problema de Coloração em Grafos, há três algoritmos base: O algoritmo de Lawler [6], com o tempo de processamento de $O^*(2.4423^n)$, o algoritmo de Eppstein [4], que é uma adaptação do algoritmo de Lawler [6] e possui um tempo de

processamento de $O(2.4150^n)$ e o algoritmo de Byskov [2], que é uma adaptação do algoritmo de Eppstein e possui um tempo de processamento de $O(2.4023^n)$.

*1) Algoritmo de Lawler:* Esse foi o primeiro algoritmo de programação dinâmica a ser apresentado para o problema de Coloração de Grafos e se baseia no teorema de Wang:
**Teorema 1:** Todo grafo possui uma coloração ótima na qual (pelo menos) uma das cores é um conjunto independente maximal.
Segue o algoritmo:

**Algorithm 1:** LAWLER (G)

```
Input: Um grafo G qualquer
Result: O número cromático de G
n ← |V(G)| ;
X ← array indexada do 0 até o 2^n − 1;
X[0] ← 0 ;
for S ← 1 to 2^n − 1 do
    s ← f(S);
    X[s] ← ∞
    for I ∈I(G[S]) do
        i ← f(S/I);
        if X[i] + 1 < X[s] then
            X[s] ← X[i] + 1
        end
    end
end
return X[2^n − 1]
```

Dessa forma, o algoritmo exato de Lawler retorna o número cromático do Grafo em $O(2.4423^n)$ e $\Theta(2^n)$.

*2) Algoritmo de Eppstein:* O Algoritmo de Eppstein é uma adaptação do algoritmo de Lawler.
Ele realiza duas modificações: A primeira é o pré-processamento dos subgrafos 3-coloríveis do grafo de entrada. A outra é preencher o vetor $X$ (a tabela de processamento dinâmico) em uma ordem diferente que permite pular o processamento de conjuntos independentes maximais além de um certo tamanho.
Segue o algoritmo:

Dessa forma, o algoritmo exato de Eppstein retorna o número cromático do Grafo em $O(2.4150^n)$ e $\Theta(2^n)$.

O próprio Eppstein propôs um algoritmo que retorna a coloração ótima, a partir do número cromático encontrado com o algoritmo anterior.
Esse novo algoritmo busca por um grafo $G[T]$ $k' - colorivel$ em $G$, onde $k' \in \{1, 2, \ldots, k\}$ e $T \subset V(G)$. Se $k' = k - 1$, então pelo Teorema de (Madsen, Nielsen and Skjernaa), os vértices em $G - T$ constituem um conjunto independente maximal $I$ que pode ser removido do grafo. Esse processo é repetido para os vértices de $G$ restantes até que um conjunto vazio seja encontrado.

**Algorithm 2:** Eppstein (G)

```
Input: Um grafo G qualquer
Result: O número cromático de G
n ← |V(G)| ;
X ← array indexada do 0 até o 2^n − 1;
X[0] ← 0 ;
for S ← 1 to 2^n − 1 do
    i ← f(S);
    Utilizar o algoritmo de Beigel e Eppstein em G[S].
    if 𝒳(G[c(S)]) ≤ 3 then
        X[i] ← 𝒳(G[c(S)])
    end
    else
        X[i] ← ∞
    end
    for S ← 1 to 2^n − 1 do
        i ← f(S);
        if 3 ≤ X[i] < ∞ then
            X[s] ← X[i] + 1 ;
            for I ∈ I(G − S) tal que |I| ≤ |S|/X[S] do
                j ← f(S ∩ I) ;
                if X[i] + 1 < X[j] then
                    X[j] ← X[i] + 1
                end
            end
        end
    end
end
return X[2^n − 1]
```

Segue o algoritmo de coloração ótima:

**Algorithm 3:** EppsteinOptColor (G)

```
Input: Um grafo G
Result: Uma Coloração ótima de G
X ← array calculada no Algoritmo de Eppstein;
S ← V(G) ;
for T2^n − 1 até 0 do
    s ← f(S);
    t ← f(T); i ← f(S/T)
    if T ⊂ S && X[i] = 1 && X[t] = X[s] − 1 then
        Coloque a mesma cor em cada vértice de S/T ;
    end
end
```

Dessa forma, o algoritmo exato de Eppstein retorna uma coloração ótima do Grafo $G$ em $O(2.4150^n)$ e $\Theta(2^n)$ também.

*3) Algoritmo de Byskov:* Esse algoritmo é uma adaptação do algoritmo de Epstein, a melhoria consiste em buscar pelos

subgrafos 4-coloríveis de G após encontrar os 3-coloríveis. Esse é, até o atual momento, o melhor de algoritmo de programação dinâmica para o problema de coloração em grafos.

Segue o algoritmo:

**Algorithm 4:** Byskov (G)

**Input:** Um grafo $G$
**Result:** O número cromático de $G$
$n \leftarrow |V(G)|$ ;
$X \leftarrow$ array indexada do 0 até o $2^n - 1$;
$X[0] \leftarrow 0$ ;
**for** $S \leftarrow 1$ ***to*** $2^n - 1$ **do**
  Utilizar o algoritmo de Beigel e Eppstein em $G[S]$ para encontrar os subgrafos 3-coloríveis de $G$ como no Algoritmo de Eppstein.
**end**
**for** $I \in I(G)$ **do**
  **for** *todo* $S \subset (V(G))/I$ **do**
    $i \leftarrow f(S)$;
    **if** $X[i] = 3$ **then**
      $j \leftarrow f(S \cup I)$;
      **if** $X[j] > 4$ **then**
        $X[j] \leftarrow 4$;
      **end**
    **end**
  **end**
**end**
**for** $S \leftarrow 1$ ***to*** $2^n - 1$ **do**
  $i \leftarrow f(S)$ **if** $4 \leq X[i] < \infty$ **then**
    **for** $I \in I(G - S)$ *tal que* $|I| \leq |S|/X[S]$ **do**
      $j \leftarrow f(S \cup I)$ ;
      **if** $X[j] > X[i] + 1$ **then**
        $X[j] \leftarrow X[i] + 1$ ;
      **end**
    **end**
  **end**
**end**
**return** $X[2^n - 1]$

Dessa forma, o algoritmo exato de Byskov retorna o número cromático do Grafo em $O(2.4023^n)$ e $\Theta(2^n)$.

### *B. Branch-and-Bound*

Quando se trata de Branch-and-Bound para o problema de Coloração em Grafos, há dois algoritmos base: O algoritmo de Brelaz [1], que se baseia no procedimento guloso **DSATUR** , e o algoritmo de Zykov, que é baseado nas próprias árvores de Zykov.

*1) Algoritmo de Brelaz:* Esse algoritmo é baseado no procedimento guloso **DSATUR** para determinar um limite superior para o número cromático.

O próprio Brelaz [1] define o *grau de saturação* de um vértice como sendo o número de cores diferentes que foram atribuídas aos seus vizinho durante a coloração.

Cada cor possui um índice $i \in \mathbf{N}$. A cada passo da heurística, o vértice com o maior grau de saturação é selecionado para receber a cor tal que o $i$ mínimo . Se mais de um vértice possui o mesmo grau de saturação, então o vértice com o maior número de vizinho é escolhido como o desempate. Se ainda assim, houver um empate então o vértice é escolhido aleatoriamente .

Segue o algoritmo:

**Algorithm 5:** DSATUR (G)

**Input:** Um grafo $G$
**Result:** Uma coloração $\zeta$ de $G$
$n \leftarrow |V(G)|$ ;
$I_i$ é uma cor de $G$, para $1 \leq i \leq n$;
Pegue uma ordenação $(v_1, v_2, \cdots, v_n)$ para $V(G)$, tal que $d_G(v_{i+1})$, para $i \in \{1, \cdots, n\}$;
Adicione $v_1$ a $I_1$ **for** *todo* $i \leftarrow 1$ *até* $n$ **do**
  $I_i \leftarrow \emptyset$;
**end**
**while** *Há vértices descoloridos* **do**
  Selecione o vértice não colorido $v$ que possui o maior grau de saturação. Se houver mais de um vértice com o maior grau de saturação, escolha aquele que possui o maior $d_G(v)$. Se ocorrer um empate novamente, escolha um desses vértices aleatoriamente. ;
  $j \leftarrow i$;
  **while** $v$ *não está colorido &&* $k \leq n$ **do**
    **if** $N_G(v) \cap I_j = \emptyset$ **then**
      Adicione $v$ a $I_j$;
      Adicione $I_j$ a $\zeta$;
    **end**
    **else**
      $j \leftarrow j + 1$;
    **end**
  **end**
**end**
**return** $\zeta$

O algoritmo de Brelaz [1] sofreu uma adaptação no trabalho de Sewell [9] e San Segundo [8], onde eles propõem um novo critério de desempate no passo de escolher o vértice com o maior grau de saturação.
O algoritmo adaptado por Sewell [9] escolhe o vértice que compartilha a maior quantidade de cores disponíveis com os seus vizinhos no subgrafo descolorido.
Já o algoritmo de San Segundo [8] escolhe o vértice que compartilha a maior quantidade de cores com os vértices que possuem o mesmo grau de saturação.

Há ainda outros trabalhos publicados por Furini, Gabriel e Ternier [5], que fazem uma adaptação do algoritmo de Brelaz.

O próprio San Segundo [8] realizou os testes dos algoritmos de Brelaz [1] e Sewell [9], utilizando as instâncias DIMACS [3]. O próprio autor concluiu que os três performaram de maneira muito parecida mas que o algoritmo de San Segundo foi mais eficiente.

*2) Algoritmo de Zykov:* O algoritmo de Zykov é baseado em sua recorrência que afirma que, se $G$ é um grafo e para qualquer par de vértices $x, y \in V(G)$ que não compartilham uma aresta, uma coloração ótima de G pode atribuir a mesma cor a $x$ e $y$ ou não.

A base desse algoritmo são as *árvores de Zykov*[16]. Segue os algoritmos necessários para a execução do algoritmo de Zykov:

**Algorithm 6:** Color(G)

**Input:** Um grafo $G$
**Result:** o menor valor de $q$
$n \leftarrow |V(G)|$ ;
**if** *$G$ é um grafo completo* **then**
  $q \leftarrow min\{n, q\}$;
**end**
**else if** *$G$ não tem uma $q-$ clique* **then**
  Escolha $x, y \in V(G)$ tal que $\{x, y\} \notin E(G)$;
  Color($G'_{xy}$) contração de vértices ;
  Color($G''_{xy}$) adição de arestas ;
**end**
**return** $q$

**Algorithm 7:** Zykov(G)

**Input:** Um grafo $G$
**Result:** o número cromático de $G$
$n \leftarrow |V(G)|$ ;
$\mathcal{X}(G) \leftarrow Color(G)$;
**return** $\mathcal{X}(G)$

## VI. 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.

## VII. 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.

## VIII. Casos de teste ou geradores de casos de teste encontrados na literatura

Os casos de teste mais utilizados 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.

## IX. 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:

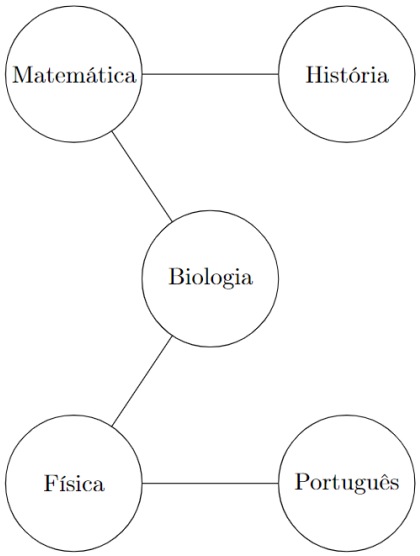


Figura 6: Grafo reorganizado

### A. *Técnica escolhida para o desenvolvimento do algoritmo:*

Backtracking é uma técnica algorítmica utilizada para encontrar soluções para problemas computacionais, especialmente em situações onde é necessário explorar todas as possíveis opções de forma sistemática. O método envolve a busca incremental por uma solução, recuando (backtrack) quando um caminho não conduz a uma solução válida. Durante a busca, o algoritmo mantém um estado parcial, testando e explorando diversas opções até encontrar a solução desejada ou determinar que não existe uma solução viável. Backtracking é frequentemente empregado em problemas de otimização, jogos, quebra-cabeças e em situações em que é necessário encontrar combinações ou configurações específicas de um conjunto de opções.

### B. *pseudocódigo:*

Segue o pseudocódigo para a solução proposta (Nomeamos o algoritmo com as inicias das autoras):

**Algorithm 8:** colorir($v$)

```
Input: um vértice v do grafo G
Result: o vértice colorido
for cor : cores do
    if cor ∩ adjacentes(v)= ∅ then
        cor ← cor ∪ {v};
        break;
    end
end
```

**Algorithm 9:** ET($v$)

```
Input: um vértice v do grafo G
Result: Todas as colorações possíveis de G
colorir(v);
verticesNaoColoridos.remove(v);
if !verticesNaoColoridos.empty() then
    for vertice i : verticesNaoColoridos do
        ET(i)
    end
end
```

### C. *Análise de complexidade:*

Analisando a complexidade do algoritmo, segue:
Este algoritmo executa o backtracking e ele é chamado uma vez para cada vértice do grafo, excluindo as chamadas recursivas, o que multiplicará à complexidade $O(n)$.

Para o procedimento de colorir, sua complexidade de verificação de interseção é de $O(n)$.

A seguir, fazemos a chamada recursiva para cada vértice ainda não colorido, ou seja, $n-1$ vezes, depois $n-2$, depois $n-3$ e assim por diante até não haver mais vértices não coloridos.

Assim, teremos como custo por etapa $O(n^2+n!)$, devido ao $n$ multiplicando o custo inicial, que é o mesmo que $O(n!)$.

## X. 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<br><br>550XCJ/<br>550XCR | Processador Intel (R) Core (TM) i3-10110U 2.59 GHz<br>2 x 4 GB DDR4 1333 MHz<br>SSD 256 GB<br>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, as autoras criaram alguns grafos a serem coloridos, visto que ambos os algoritmos a serem testados possuíam uma complexidade muito grande e testas com a bilioteca antiga ficaria inviável. Os testes para esses algoritmos se encontram no diretório *test_lib*

*4) Orientações para a realização dos testes:* Para a realização dos testes foi utilizado o repositório: Implementação Algoritmo ET, sendo possível cloná-lo com o comando:

```
git clone https://github.com/stellamaris-22/unidade-3-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:

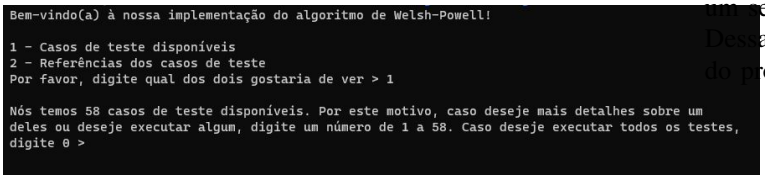


Figura 7: Tela de Inicialização para os testes

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* |
|---|---|---|---|---|
| original.col | 5 | 4 | 2 | 0 |
| cria.col | 4 | 4 | 2 | 0 |
| corinthians.col | 3 | 3 | 3 | 0 |
| ts.col | 4 | 3 | 2 | 0 |
| sm.col | 5 | 5 | 2 | 1 |
| id.col | 5 | 7 | 3 | 1 |
| realmadrid.col | 6 | 5 | 2 | 6 |
| panic.col | 4 | 5 | 4 | 0 |
| es.col | 5 | 8 | 4 | 1 |

## XI. Comparação com resultados de heurísticas anteriores

Nessa seção, faremos a comparação com os resultados obtidos com a implementação do algoritmo $ET$ e do algoritmo de Zykov.

### *A. Comparação de resultados*

Como é possível verificar na Tabela I, os resultados foram bem parecidos, embora o algoritmo com branch-and-bound possuiu mais uniformidade nos resultados. Já o de backtracking oscilou um pouco mais.

## XII. 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 do algoritmo de Zykov.
Obtivemos resultados muito parecidos entre os dois, embora um se apresentou mais uniforme do que o outro.
Dessa forma, a técnica a ser escolhida depende do contexto do problema.

## XIII. Referências bibliográficas

## Referências

[1] Daniel Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22:251–256, 04 1979.

[2] Jens M. Byskov. Chromatic number in time $o(2.4023^n)$ using maximal independent sets. Technical Report 45, BRICS (Basic Research in Computer Science) Report Series, 2002.

[3] Center for Discrete Mathematics and Theoretical Computer Science and DIMACS Implementation Challenges. Dimacs implementation challenges. http://dimacs.rutgers.edu/archive/Challenges/, 1994. Online; accessed 04 July 2018.

[4] David Eppstein. Small maximal independent sets and faster exact graph coloring. *J. Graph Algorithms Appl.*, 7:131–140, 2000.

[5] Fabio Furini, Virginie Gabrel, and Ivan-Christian Ternier. An improved dsatur-based branch-and-bound algorithm for the vertex coloring problem. *Networks*, 69(1):124–141, 2017.

[6] Eugene L. Lawler. A note on the complexity of the chromatic number problem. *Inf. Process. Lett.*, 5:66–67, 1976.

[7] Alane Marie de Lima and Renato Carmo. Exact algorithms for the graph coloring problem. *Revista de Informática Teórica e Aplicada*, 25:57, 11 2018.

[8] Paulo S. Segundo. A new dsatur-based algorithm for exact vertex coloring. *Computers & Operations Research*, 39(7):1724–1733, 2012.

[9] Edward C. Sewell. A branch and bound algorithm for the stability number of a sparse graph. *INFORMS Journal on Computing*, 10(4):438–447, 1998.

- 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. ]

| Instância | Vértices | Arestas | CoresET | TempoET em *ms* | CoresZ | TempoZ |
|---|---|---|---|---|---|---|
| original.col | 5 | 4 | 2 | 0 | 5 | 0 |
| cria.col | 4 | 4 | 2 | 0 | 4 | 0 |
| corinthians.col | 3 | 3 | 3 | 0 | 3 | 0 |
| ts.col | 4 | 3 | 2 | 0 | 3 | 0 |
| sm.col | 5 | 5 | 2 | 1 | 3 | 0 |
| id.col | 5 | 7 | 3 | 1 | 3 | 0 |
| realmadrid.col | 6 | 5 | 2 | 6 | 3 | 2 |
| panic.col | 4 | 5 | 4 | 0 | 3 | 0 |
| es.col | 5 | 8 | 4 | 1 | 3 | 0 |

Tabela I: Tabela de comparação ente o algoritmo autoral e o algoritmo de Zykov


- Moritz Willnauer, "*Modelling and Solving a Scheduling Problem by Max-Flow*", Research Institute for Symbolic Computation (RISC), 2020.