Validação de Instâncias de Coloração de Vértices Aplicado à Alocação de Disciplinas

Leandro de Souza Oliveira

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DA FACOM-UFMS

Data da Defesa:

Assinatura:_____

Validação de Instâncias de Coloração de Vértices Aplicado à Alocação de Disciplinas

Leandro de Souza Oliveira

Orientador: *Vagner Pedrotti*

Monografia apresentada como requisito parcial para aprovação na Componente Curricular Não-Disciplinar Trabalho de Conclusão de Curso do curso de Graduação em Ciência da Computação, da Universidade Federal de Mato Grosso do Sul.

UFMS - Campo Grande
novembro/2025

Agradecimentos

Em memória ao meu pai, Francisco de Assis Costa de Oliveira.

Resumo

Este trabalho apresenta o desenvolvimento de um validador de instâncias de coloração em grafos que modelam o problema de distribuição de disciplinas na FACOM, focado exclusivamente na alocação de horários para turmas, não na definição de quais docentes ministram cada disciplina. A instância é uma tradução doutra abordagem já implementada na instituição que resolve o problema com programação linear. Restrições típicas do processo são conflitos de horários entre disciplinas do mesmo curso e semestre ou lecionadas pelo mesmo docente.

Como se trata de um problema NP-difícil, o algoritmo implementado verifica a consistência de uma instância de horários para turmas, buscando identificar uma configuração válida e que auxilie na automatização já presente. Testes realizados com instâncias reais da FACOM demonstraram que a abordagem é promissora, embora ainda incompleta. O validador mostrou-se competente em dois dos três semestres analisados, porém enfrentou dificuldades com um semestre letivo mais complexo.

Sumário

CAPÍTULO

1

Introdução

1.1 Introdução

Uma etapa obrigatória da organização semestral de toda instituição de ensino é a distribuição de disciplinas em salas e horários de maneira que atendam a um conjunto de restrições e objetivos institucionais. Essa tarefa envolve restrições que vão desde limitações físicas de espaços adequados até critérios pedagógicos que são determinados pela grade curricular de cada curso. Devido a complexidade dessas relações, torna-se evidente os benefícios de automatizar esse processo, reduzindo significativamente o esforço manual, economizando tempo e entregando um resultado mais objetivo de acordo com os critérios estabelecidos pela própria instituição.

Na FACOM, o processo de distribuição já é auxiliado por um sistema automatizado, o intuíto deste trabalho é apresentar uma etapa adicional, um passo de pré-validação de uma determinada instância do problema da distribuição de disciplinas usando coloração em grafos.

1.2 Justificativa

Uma instância do problema de alocação de disciplinas reúne informações necessárias para a construção de um horário para a grade curricular de um semestre: o conjunto de disciplinas, docentes e os estudantes atendidos, bem como as restrições físicas de salas e pedagógicas (pré-requisitos, conflitos de horário para disciplinas obrigatórias)

Como o problema de alocação de horários com conflitos é NP-completo, do

inglês, *timetabling problem* (Even et al., 1976). O tempo para resolver uma instância cresce exponencialmente com o número de turmas, consequentemente até mesmo os *solvers* do estado da arte podem demandar da ordem de horas ou dias para entregar a solução ótima - ou, mais comumente, uma solução próxima da ótima.

Nesse contexto, uma instância introduzida pode ser *inviável*: ou seja, não existe solução. A identificação dessa inviabilidade costuma ser tão custoso quanto resolver o próprio problema.

Diante disso, uma instância inválida introduzida pode acarretar desperdício de tempo computacional. A inclusão de uma nova etapa de pré-validação, portanto, tem o potencial de diminuir esse impacto.

1.3 Objetivos

O objetivo é propor uma abordagem para modelar o problema de alocação de disciplinas como uma variação do problema de coloração de vértices e adaptar o algoritmo exato de Zykov (Zykov, 1949) para tratar as restrições primárias do problema com coloração de vértices. Em seguida, utilizamos emparelhamento máximo de grafos bipartidos para validação de restrições secundárias (alocação de salas). Encontrando, assim, qualquer solução válida o mais rápido possível.

CAPÍTULO

2

Contextualização

Neste capítulo serão introduzidos alguns conceitos e exemplos básicos de grafos e coloração de vértices, em sequência vamos introduzir o algoritmo de Zykov e finalmente, o problema de alocação de disciplinas e como ele pode ser traduzido para coloração de vértices

2.1 Definições Gerais

Denota-se por $G = (V, E)$ um grafo não dirigido, V representa o conjunto de vértices enquanto E representa os pares de vértices ligados por uma aresta, Ou seja, dados $x, y \in V$ e $xy \in E$ significa que existe uma aresta entre x e y em G.

Uma *clique* C é um subconjunto de V tal que, para quaisquer dois vertices $x, y \in C$, temos $xy \in E$.

Diz-se que uma clique qualquer C_a é *maximal* se não existe nenhuma outra clique C_b tal que $C_a \subset C_b$.

Já uma *clique máxima* é uma clique de maior cardinalidade entre todas as cliques de G. Uma clique $C_{\max}$ é máxima se, e somente se, $|C_{\max}| \geq |C'|$ para todas as cliques C' de G.

Um grafo G é chamado *completo* se, e somente se, o conjunto de vértices da clique maxima $C_{\max}$ é igual a V, ou seja, todos os vértices do grafo estão conectados entre si. O grafo completo com n vértices é denotado por K_n.

2.2 Coloração de Vértices

Uma *k-coloração* de um grafo é o ato de atribuir uma cor entre k cores disponíveis para cada um de seus vértices. Uma coloração qualquer em um

grafo G é dita como *própria* se não existirem dois vértices adjacentes em G que compartilhem a mesma cor. O problema da coloração de vértices consiste em encontrar o menor número k tal que G seja *k-colorível*. Esse valor de k se chama de *número cromático*, também definido como $\chi(G)$. Encontrar esse valor é NP-difícil (Karp, 1972).

Seja o grafo exposto na Figura 2.1, observa-se uma 4-coloração que configura uma das soluções ótimas. Note que é impossível colorir o vértice rotulado por A com qualquer uma das outras cores já utilizadas, uma vez que A é adjacente a todos os demais vértices do grafo.

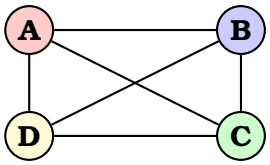


Figura 2.1: O grafo completo K_4. Seu número cromático é 4.

O número cromático de todo grafo completo é igual à sua quantidade de vértices, formalmente $\chi(K_n) = n$. Consequentemente, uma clique C qualquer de um grafo G é colorível com $|C|$ cores pois toda clique é um subgrafo $K_{|C|}$ de G.

2.2.1 Coloração por listas

Uma k-coloração por listas, do inglês, *list coloring*, é uma generalização da k-coloração introduzida por Vizing (Vizing, 1976).

Seja $\mathcal{C}$ um conjunto de cores. Em uma *coloração por listas*, cada vértice $v \in V(G)$ possui uma lista de cores permitidas $L(v) \subseteq \mathcal{C}$. Um grafo G é dito *L-colorível* se existe uma função função $f : V(G) \to \mathcal{C}$ tal que $f(v) \in L(v)$ para todo $v \in V(G)$ e $f(u) \neq f(v)$ sempre que $uv \in E(G)$.

Note que a k-coloração como previamente definida é um caso especial de coloração por listas, em que todos os vértices possuem a mesma lista de cores $\{1, 2, \ldots, k\}$.

2.3 Algoritmo de Zykov

Em 1949, Zykov introduziu um teorema que entrega o número cromático de um grafo (Zykov, 1949). Zykov afirma que a partir de um grafo G, dado um par de vértices $x, y \in V(G)$ não adjacentes, podemos gerar dois novos grafos:

- G' onde x, y se contraem em um único vértice z; e
- G'' onde criamos uma aresta $xy \in E(G'')$.

Zykov então afirma que o número cromático de G é dado pela seguinte recorrência:

$$\chi(G) = \min(\chi(G'), \chi(G'')).$$

Em outras palavras uma coloração ótima do grafo original é dada pelo menor número cromático entre G' e G''. Uma coloração ótima de G' atribui a mesma cor para x, y em G enquanto uma coloração ótima em G'' atribui cores distintas para x e y em G.

2.3.1 Árvore de Zykov

Essa recorrência constrói uma árvore binária denomidada *Árvore de Zykov*. Podemos então, explorar essa árvore recursivamente para encontrar $\chi(G)$ de qualquer grafo G. Desta forma, cada nó folha da árvore de Zykov contém um grafo completo K_v, que implica em uma coloração própria de $\chi(K_v) = v$, tornando G *v-colorável*, enquanto o número cromático de G é o menor número cromático encontrado em todos os nós folhas da árvore, ou seja:

$$\chi(G) = \min\{\, v \mid K_v \text{ é folha da árvore de Zykov de G} \,\}.$$

O algoritmo apresentado a seguir, baseado em (Neto and Gomes, 2014) explora a árvore de Zykov inteira e é um algoritmo exato para a coloração de vértices. Em um capítulo posterior vamos discutir como pretendemos melhorar o desempenho do algoritmo para o nosso caso de uso.

Algoritmo 1: ZYKOV(G)

Entrada: Um grafo G
Saída: Número cromático de G

se G *é completo* **então**
  $q \leftarrow |V(G)|$;
fim
senão
  Escolha $x, y \in V(G)$ tais que $\{x, y\} \notin E(G)$;
  $r_1 \leftarrow$ ZYKOV(G'_{xy}) ; `// contração de vértices x e y`
  $r_2 \leftarrow$ ZYKOV(G''_{xy}) ; `// adição de aresta entre xy`
  $q \leftarrow \min(r_1, r_2)$;
fim
retorna q;

A figura a seguir ilustra a árvore de Zykov de um grafo G. Os nós coloridos são folhas, condição de parada para a recursão do algoritmo.

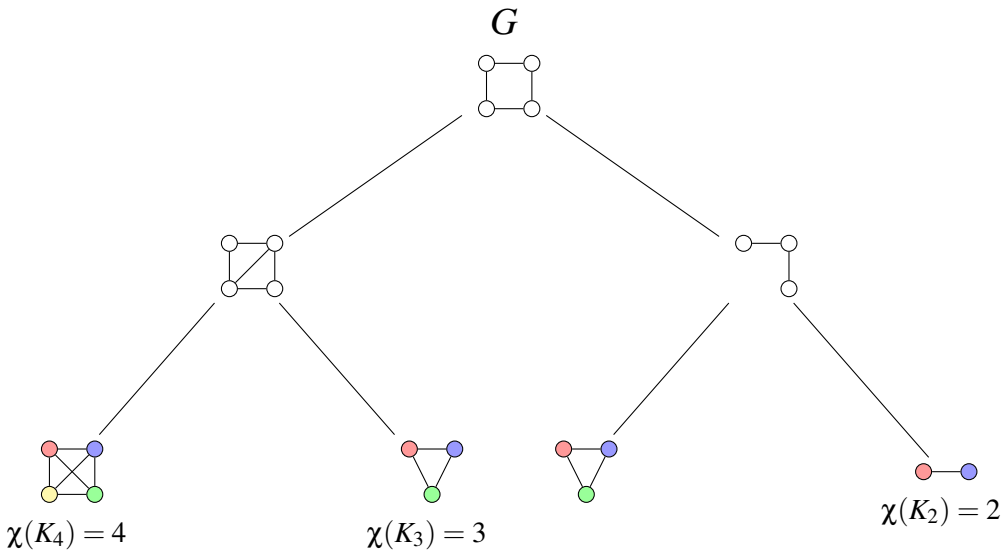


Figura 2.2: Árvore de Zykov para G, $\chi(G) = 2$

2.4 Alocação de disciplinas

A alocação de disciplinas visa distribuir aulas em períodos de tempo distintos, garantindo que nenhuma turma compartilhe professores, salas ou estudantes simultaneamente.

O problema de alocação de disciplinas é bem documentado na literatura. (Welsh and Powell, 1967) ilustraram inicialmente a relação entre problemas de programação de horários em termos gerais com a coloração.

Mais tarde, (Carter, 1986) observou que os problemas práticos de alocação de disciplinas diferem da coloração clássica por incluírem restrições adicionais, como limites de capacidade de sala, e preferências de tempo, Carter chamou essas restrições de *restrições secundárias*.

Modelos lineares como o de (Mulvey, 1982), abordam o problema sob uma perspectiva distinta da coloração de grafos. O método de Mulvey formula o problema diretamente como um modelo de programação linear inteira, essas modelagens puramente algébricas acabam integrando diretamente as restrições secundárias na formulação matemática, que não é possível com a coloração de vértices.

2.4.1 Restrições

As restrições são condições essenciais que devem ser satisfeitas para que um cronograma seja válido. Por exemplo, duas disciplinas lecionadas pelo mesmo docente não podem ser programadas no mesmo horário, uma disciplina obrigatória para calouros não pode ser programada no mesmo horário que outra disciplina, também obrigatória para calouros.

Por outro lado, as restrições também podem ser classificadas como relacionadas ao espaço.

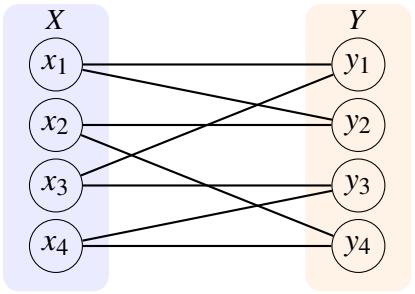
No contexto da alocação de disciplinas, essas restrições referem-se à disponibilidade e capacidade das salas de aula. Cada instituição possui um número limitado de salas, com capacidades e finalidades específicas, teóricas ou laboratoriais. Assim, ao elaborar o cronograma, é obrigatório garantir que as disciplinas sejam atribuídas a salas compatíveis em capacidade e tipo.

É importante ressaltar que, como (Carter, 1986) observou, restrições como as de sala são secundárias e não são imediatamente mapeadas na coloração de horários.

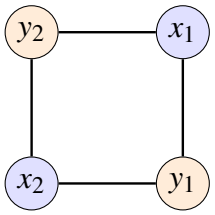
Logo, nem toda coloração de horário satisfaz completamente essas restrições secundárias.

2.5 Grafos bipartidos

Um grafo G é dito *bipartido* se $V(G)$ pode ser particionado em dois conjuntos: $V(G) = (X,Y)$ tal que para todas as arestas $ab \in E(G)$, $a \in X$ e $b \in Y$. Denominamos X e Y de partes do grafo (Harris et al., 2008).



(a) G bipartido com partições X e Y.



(b) O ciclo C_4 é bipartido, conhecido também como $K_{2,2}$

Figura 2.3: Dois grafos bipartidos

2.5.1 Emparelhamento máximo

Um emparelhamento de um grafo qualquer G é um conjunto de arestas independentes, ou seja um conjunto de arestas $M \subseteq E(G)$ onde nenhum par de arestas compartilha vértices.

Para grafos bipartidos $V(G) = (X,Y)$, um emparelhamento pode ser descrito como um conjunto de arestas que liga cada vértice de X a, no máximo, um vértice de Y, e vice-versa.

O problema do emparelhamento máximo, então, consiste em encontrar um conjunto de arestas de maior cardinalidade que satisfaz as condições descritas.

CAPÍTULO

3

Metodologia

Este capítulo descreve a modelagem e métodos propostos para pré-validar uma instância do problema da alocação de disciplinas dentro da Faculdade de Computação da UFMS (FACOM-UFMS) com coloração de vértices e o algoritmo de Zykov.

3.1 Modelagem FACOM

Inicialmente, vamos definir como é construído o grafo que representa o problema da coloração para a alocação de horários e salas da FACOM, considere a seguinte hierarquia:

$$\text{FACOM} \rightarrow \text{cursos} \rightarrow \text{disciplinas} \rightarrow \text{turmas} \rightarrow \text{aulas}.$$

Resumidamente, a FACOM oferece distintos cursos, cada um com suas disciplinas; cada disciplina pode ter distintas turmas e cada aula corresponde a uma ocorrência específica no calendário semanal de uma turma.

Como nossa modelagem é uma tradução do problema de alocação de horários, cada cor representa um horário. Uma vez que tentamos colorir aulas que permitem períodos de tempo distintos pela grade curricular, nossa modelagem necessariamente precisa ser uma de coloração por listas.

3.1.1 Vértices

Cada vértice do grafo representa uma aula específica, e não apenas uma turma. Essencialmente, uma turma quase sempre estará associada a múl-

tiplos vértices, cada um correspondente a diferentes aulas que ocorrem em diferentes momentos da semana.

3.1.2 Cores

Cada vértice v de um grafo G possuí uma lista $L(v)$ de cores permitidas em v, cada cor é um horário disponível em um dia da semana, note que restrições físicas de salas e laboratórios não são mapeadas com cores. Dada uma coloração válida, uma busca por um ensalamento válido é feito em sequência.

3.1.3 Arestas

Uma aresta xy representa que as aulas de x e y não podem ocorrer no mesmo horário, consideramos os seguintes casos onde isso é verdade:

- x e y são lecionadas pelo mesmo docente, essa restrição também mapeia as restrições de aulas distintas pertencentes à mesma turma.
- x e y são disciplinas obrigatórias do mesmo semestre letivo, que compartilham o mesmo conjunto de estudantes

Além dessas arestas estruturais, também permitimos a inserção de arestas inseridas manualmente, conforme necessidades administrativas.

3.1.4 Ensalamento

Um ensalamento pode ser encontrado em tempo polinomial via emparelhamento máximo de grafos bipartidos, para cada cor k de G em uma configuração *L-colorável*: definimos as duas partições para um grafo bipartido $B_k = (X, Y)$. A primeira partição X de B_k representa todas as aulas em um mesmo horário, essencialmente vértices com a mesma cor k em G.

Cada vértice $v \in V(G)$ também guarda consigo, uma segunda lista $L'(v)$ de salas permitidas, então a segunda partição Y consiste de todas as salas disponíveis, formalmente:

$$Y = \bigcup_{v \in G} L'(v),$$

Ligamos, com arestas, os vértices correspondentes de X e cada uma de suas cores permitidas em Y, computando o emparelhamento máximo M_{max}, se $|M_{max}| = |X|$, então um ensalamento válido foi encontrado, e está presente em M_{max}.

3.2 Algoritmo de Zykov

O algoritmo de Zykov baseado em (Zykov, 1949) é um algoritmo exato para a coloração, vamos explorar agora como modificar o algoritmo para ele melhor se comportar dado o contexto de coloração por listas com ensalamento no qual estamos inseridos

3.2.1 Coloração por listas

A recorrência introduzida no capítulo 2 continua sendo verdadeira para a coloração em listas. A principal mudança sendo a possibilidade da solução não ser *L-colorável*.

É fundamental alterar o passo da contração de dois vértices do algoritmo. Se temos $x, y \in V(G)$ tal que $L(x), L(y) \subseteq \mathcal{C}$ mapeiam suas respectivas cores aceitas. A contração de x e y, nos gera z que herda apenas as cores compatíveis entre ambos, ou seja, sua interseção.

$$L(z) = L(x) \cap L(y).$$

.

Essa contração, nos gera um novo nó que representa, então, mais de uma aula. Logo é de se esperar que todo nó folha dentro da árvore de Zykov represente um conjunto de aulas. A Figura 3.1 exemplifica esse processo.

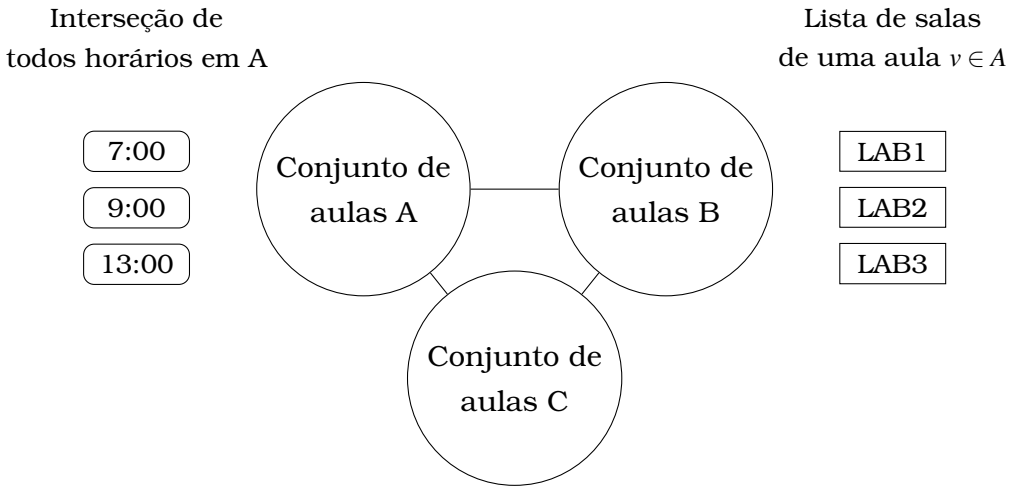


Figura 3.1: Nó folha em uma árvore de Zykov

3.2.2 Critério de parada

Diferente do uso clássico, interrompemos a busca assim que uma coloração *válida* é encontrada (Um nó folha precisa ser L-colorável, e um ensalamento válido para cada cor ser encontrado).

3.2.3 Branch-and-Bound

Qualquer algoritmo baseado em árvores de Zykov tem complexidade de $O(2^{\frac{n(n-1)}{2}})$ (de Lima and Carmo, 2018), logo, se torna fundamental adotar estratégias que nos ajudem a percorrer a árvore da melhor maneira possível.

Para auxiliar nesse objetivo, adotamos a abordagem do *Branch-and-Bound* (Lawler and Wood, 1966), que é uma técnica do paradigma de divisão e conquista.

Como o algoritmo de Zykov já é baseado em particionar um problema original em subproblemas, o aspecto de *Branching* do *Branch-and-Bound* já é satisfeito por padrão.

Sobra, então o *Bounding* e a *Poda*, descritos abaixo.

3.2.4 Bounding

São calculados dois limites para o valor possível de $\chi(G)$: um superior e outro inferior. Empregamos limites simples e eficientes, localmente em cada nó:

- **Limitante inferior** $LI(G)$: tamanho da maior clique encontrada em G ou nos nós antecessores de G na árvore.
- **Limitante Superior** $LS(G)$: O maior grau de todos os vértices em G + 1

Normalmente, com o *Branch-and-Bound* usamos esses limitantes para podar a busca e encontrar rapidamente a solução ótima, em nosso caso, utilizamos os limitantes apenas para guiar uma fila de prioridade escolher os melhores nós para serem explorados, exploramos nós com o *menor valor* para:

$$LS(G) - LI(G).$$

Decrescendo essa diferença, exploramos a árvore de modo que os limitantes rápidamente se encontram, que indica proximidade a um nó folha qualquer.

3.2.5 Podas

Toda vez que exploramos um nó, uma verificação extensiva é feita para a validação de uma L-coloração viável e ensalamento do mesmo. Um nó que não passa todas as etapas da verificação é prontamente podado por invalidez, cortando-se toda sua subárvore e removendo-a do espaço de busca. As etapas dessa validação serão discutidas no capítulo 4.

CAPÍTULO

4

Trabalho Desenvolvido

Este capítulo descreve o desenvolvimento e implementação dos modelos descritos no capítulo anterior, além de descrever melhorias que foram desenvolvidas ao longo do tempo de modo que melhor atendem os requisitos específicos da coloração da nossa instância.

4.1 Construção das instâncias

Todas as instâncias utilizadas foram construídas a partir de uma base de dados real da FACOM e todos os seus cursos, disciplinas, turmas e, portanto, aulas. Essa base de dados é mantida em diversos arquivos *.csv* que descrevem as diversas relações necessárias para a construção do grafo correspondente a cada instância, que seria um semestre letivo da FACOM.

O arquivo de saída é um arquivo de texto comum que descreve o grafo da coloração por listas com todas suas restrições e listas de cores e salas, uma funcionalidade importante para o desenvolvimento de testes foi a possibilidade de particionar o grafo em cursos selecionados, em vez de considerar todos, sempre.

O código que constroi o grafo foi implementado em *Node.js*, um ambiente de execução de *JavaScript* fora do contexto de um navegador. O alto nível de abstração permitiu um desenvolvimento mais ágil, sendo que, para um gerador de instâncias o desempenho de processamento não é um fator crítico.

4.2 Ambiente de desenvolvimento

Para resolver o problema de fato, foi utilizada a *IDE Visual Studio 2022*, utilizando a linguagem C++ e o compilador MSVC no padrão ISO C++20. Como todo o código foi desenvolvido sem auxilio de framework algum, as ferramentas da linguagem C++ descritas em (Stroustrup, 2013) foram essenciais para a construção de abstração de código que permitiu o andamento do desenvolvimento em formas gerais. O código fonte pode ser encontrado em (de Souza Oliveira, 2025b).

4.3 Estruturas de dados utilizadas

Foi criada uma abstração de grafos, seguindo o paradigma de orientação a objetos com uma matriz de adjacência para representar as arestas. Optamos pela modelagem com matrizes por conta do tamanho das instâncias: se considerarmos 500 vértices, a matriz resultante consome cerca de 1 MB de memória - valor suficientemente pequeno para residir no *cache* de processadores modernos.

Para a lógica de contração de vértices, empregou-se uma estrutura de dados para computar sua junção: O *Union–Find*, que nos permite duas operações: `UNION()` para a consolidação de dois conjuntos de vértices em um só, e `FIND()` - que retorna o representante dos vértices consolidados. Foi utilizada a técnica de compressão de caminhos, do inglês, *path compression*, o que garante operações de tempo constante por `FIND()` (Wu and Otoo, 2005).

Com o *Union-Find*, podemos resgatar o conjunto de *aulas* representadas por um vértice após uma sequência de contrações. Para as listas de cores, usamos vetores para cada vértice.

4.4 Pré-processamento

No princípio de nossa implementação, não havia pré-processamento do grafo original, mas como o espaço de busca é muito amplo e, para o caso da FACOM, o grafo da instância é bastante esparso. Tornou-se então claro que técnicas de pré-processamento eram de suma importância para obter qualquer resultado.

Nenhuma técnica de pré-processamento que reduz o conjunto de soluções válidas foi implementada, nosso objetivo aqui é validar uma instância e podar possíveis soluções válidas seria problemático, esse pré-processamento ocorre antes de cada nó da árvore de Zykov ser processado, da raiz até a folha, as técnicas utilizadas são:

Adição de arestas: interseção vazia. Considere dois vértices não vizinhos $x, y \in V(G)$ e $L(x), L(y) \subseteq \mathcal{C}$ suas respectivas cores aceitas, a etapa de contração geraria um vértice z tal que $L(z) = L(x) \cap L(y)$, se $L(z) = \emptyset$ uma *L-coloração* com z é impossível, logo adiciona-se uma aresta xy

Contração de vértices: interseção unitária. Se $L(x) = L(y)$ e $|L(x)| = |L(y)| = 1$, os vértices compartilham exatamente uma cor, qualquer *L-coloração* válida exige que ambos sejam coloridos com a mesma cor. Logo, contraem-se x e y em z.

Adição de arestas: verificação ensalamento. De forma análoga, contemplamos a interseção de *salas*. Dados dois não vizinhos x y supomos que eles tem a mesma cor e executamos a rotina de ensalamento descrita no capítulo 3, caso o emparelhamento M_{max} não descreva um ensalamento válido, x e y não podem ter a mesma cor, e adicionamos uma aresta xy.

Propagação de restrições. Ao fixar uma cor em um vértice v, ou seja, $|L(v)| = 1$, eliminamos essa cor das listas de seus vizinhos, antecipando podas.

Ao antecipar essas contrações e adições obrigatórias, diminuímos significativamente a ramificação da árvore. Assim, o processamento parte de uma configuração mais enxuta preservando todas as soluções válidas.

4.5 Estratégias de Ramificação

A escolha do par de vértices não vizinhos x, y para a exploração da árvore é muito importante. Como cada decisão gera dois novos grafos, é fundamental escolher vertices promissores, especialmente em instâncias maiores.

Para isso, foram desenvolvidas diferentes heurísticas de seleção. Apenas uma encontrou resultados satisfatórios, todas as outras acarretavam em *backtracking* excessivo e constante na árvore de Zykov, sugerindo a exploração repetida de ramos inviáveis. Abaixo descrevemos a heurística que demonstrou o melhor desempenho experimentalmente.

Seja $N(v)$ o conjunto de vizinhos de um vértice $v \in V(G)$

Algoritmo 2: Heurística de seleção de vértices para ramificação

Entrada: Um grafo não completo G, listas de salas L' e pesos $w(v)$ (tamanho das contrações em v)

para cada *par de vértices não adjacentes* $a,b \in V(G)$ **faça**
  $I(a,b) \leftarrow N(a) \cap N(b)$; `// interseção de vizinhos de` a `e` b
  $f(a,b) \leftarrow \sum_{u \in I(a,b)} w(u)$; `// cada vizinho tem um peso, baseado em suas contrações`
  $g(a,b) \leftarrow |L'(a) \cap L'(b)|$;
fim
Ordene todos os pares a,b em $f(a,b)$ em ordem decrescente, com desempate por $g(a,b)$, também decrescente ; `// cada par` (a,b) `é inserido` $w(a)$ `vezes na lista`
$S \leftarrow 1\%$ primeiros pares da ordenação de $f(a,b)$ e $g(a,b)$;
Escolha um par $\{x,y\}$ aleatoriamente de S;
retorna $\{x,y\}$

Essa heurística emprega pesos em sua formulação. Cada vertice tem um peso $w(v)$ que é computado pela quantidade de vertices consolidados em v após contrações de vértices que criaram v a partir de nós antecessores.

Algumas das outras abordagens testadas foram: selecionar qualquer par de não adjacentes aleatoriamente; encontrar $\{x,y\}$ com o maior grau de x e y (e o inverso); encontrar $\{x,y\}$ tal que x faz parte de uma clique maximal; encontrar $\{x,y\}$ de modo que maximize o tamanho da interseção da lista de cores.

4.6 Validação de um nó

Como discutido no capítulo 3, os nós da árvore de Zykov são podados por invalidez. Dado um grafo G: para encontrar falhas estruturais de *L-coloração* em G, precisamos de um conjunto de vértices que obrigam cores distintas entre si, ou seja, uma clique $\mathcal{C} \subseteq V(G)$.

Essa validação estrutural contempla os vértices de $\mathcal{C}$ e não $V(G)$ como um todo, tornando-se importante computar grandes cliques de modo que uma cobertura ampla de G seja feita. Estudamos heurísticas para esse problema em (de Souza Oliveira, 2025a).

Encontrada uma clique $\mathcal{C}$, uma *L-coloração* para $\mathcal{C}$ pode ser computada repetindo a estratégia para o ensalamento discutido no capítulo 3, o *emparelhamento máximo de um grafo bipartido*. Criamos um grafo Bipartido $B_{\mathcal{C}}$ tal que a primeira partição contém todos os v tal que $v \in \mathcal{C}$ e a segunda contém a união de todas as cores permitidas pelos vértices de $\mathcal{C}$, formalmente:

$$\bigcup_{v \in C} L(v),$$

Ligamos, com arestas, os vértices correspondentes de C e cada uma de suas cores permitidas, encontrando uma solução para o emparelhamento máximo com o algoritmo de fluxo máximo de Ford-Fulkerson (Ford Jr. and Fulkerson, 1956). Repetimos o processo para encontrar o ensalamento, porém, como cada vértice representa, potencialmente mais de uma aula já que vértices podem ser contraídos na árvore de Zykov, consultamos a estrutura do *Union-Find* e recuperaramos os vértices consolidados em um só, para encontrar um ensalamento para cada cor.

Uma etapa extra de validação para o ensalamento foi desenvolvida. Dado um nó de Zykov, antes da ramificação pela contração de $\{x, y\}$. Verificamos se essa contração viola o ensalamento, nesse caso, podamos o nó antes de ser criado.

4.7 Otimizações adicionais

Vamos apresentar algumas técnicas desenvolvidas que não se enquadram nas seções anteriores, as abordagens prestes a serem introduzidas foram projetadas para execução intermitente dentro da busca, uma vez que seu custo em tempo de execução é muito caro para frequencias maiores.

4.7.1 Validação recursiva

Suponha que um grafo G fornecido na entrada não seja L-colorável. Caso as falhas na estrutura de G não sejam detectadas precocemente na árvore de Zykov e dado a ramificação exponencial do espaço de busca, é provável que a confirmação dessa inconsistência demore para ocorrer.

Suponha, então, que temos um nó intermediário G' da árvore de Zykov de G. Nele foi identificada uma clique C que viola alguma restrição para G'. Nesse caso, podemos selecionar um subgrafo R de G, onde R contém apenas os vértices originais e suas arestas que participaram da formação de C em G'. Em seguida validamos recursivamente a árvore de Zykov de R; uma invalidez em R implica diretamente em uma invalidez na instância completa de G.

Como não foi possível, em tempo hábil, gerar instâncias comprovadamente inválidas para fins de teste. Essa etapa foi desabilitada na versão final do programa.

4.7.2 Poda Retroativa

Analogamente, suponha que estamos em um nó intermediário G' de G, encontramos uma clique C que viola uma restrição de G'. Buscamos, então, nós na árvore de Zykov que ainda não foram explorados, verificando se cada um contém a mesma clique C. Caso encontrada, verificamos a validade de C naquele nó para uma possível poda.

4.8 Validação de saída

Se um nó folha for validado, uma solução para o problema foi encontrada e a execução é interrompida. Assim uma etapa de verificação de integridade final é executada, assegurando a consistência da implementação, essa etapa compreende quatro critérios de verificação.

Primeiro, para todo $v \in V(G)$ a escolha de cor c para v respeita a condição $c \subset L(v)$.

Analogamente, para uma sala s de v, a sala satisfaz $s \subset L'(v)$.

Terceiro, verificamos se cada vértice adjacente em G não compartilham a mesma cor.

Finalmente, temos um par $\{c,s\}$ de cor e sala para cada aula; verificamos, então, que não existe nenhuma aula que compartilhe o mesmo par.

Com isso, podemos assegurar que a solução encontrada satisfaz as condições do problema.

4.9 Uso do programa

Propomos, então, a seguinte estratégia de validação para a FACOM: recomenda-se executar o verificador com um limite de tempo definido em alguns minutos. Repetindo a execução diversas vezes para a mesma instância. Caso encontre uma solução, considere a instância como válida.

Caso contrário, o programa acusa a invalidez da instância em qualquer uma das execuções. A detecção é determinística e a instância é, de fato, inválida.

Entretanto, se todas as execuções atingirem o limite de tempo, o resultado é inconclusivo.

CAPÍTULO

5

Resultados Computacionais

Este capítulo apresenta o ambiente e os resultados computacionais obtidos, em conjunto com uma análise do que os dados representam em termos da viabilidade do programa.

5.1 Ambiente computacional

A implementação e resolução foram executadas em uma máquina com as seguintes especificações:

- Processador: AMD Ryzen 5 4500, 6 núcleos físicos, frequência máxima de 4.1 GHz.
- Memória RAM: 16 GB DDR4 a 2400 MHz.
- Armazenamento: SSD SATA de 128 GB.
- Sistema operacional: Windows 11.

5.2 Instâncias

Diversas instâncias foram construídas, todas com base em três semestres letivos reais da FACOM, referentes aos períodos letivos de 2025-1, 2025-2 e 2022-1.

Vamos apresentar um estudo de seis instâncias, três delas foram construídas com todos os cursos da FACOM, enquanto as outras três foram elaboradas com foco nos cursos de Engenharia de Software e Sistemas de Informação;

sendo, portanto, sub-grafos das instâncias completas. Vale ressaltar que esses subgrafos incluem também disciplinas optativas, que podem ser oriundas de outros cursos da FACOM. O total de cores possíveis para cada instância é *35*, referente a quantidade de horários disponíveis de segunda a sexta pela UFMS.

Todas as instâncias são válidas e foram testadas 100 vezes cada. Todos os testes de uma instância i são representados pelo conjunto T_i, foi implementada uma política de *timeout* para cada execução: caso uma solução não seja encontrada dentro de um período de 120 segundos, interrompemos o teste; chamamos o subconjunto de T_i que encontrou uma solução dentro desse período de R_i.

Nas primeiras duas colunas da seguinte tabela, ilustramos os dados dos grafos originais de cada instância; as demais colunas apresentam os tamanhos dos conjuntos R_i e T_i, bem como L_m: que é a média do valor de L da *L-coloração* de cada teste em R_i.

Tabela 5.1: Dados das instâncias de testes

instância	$\|V(G)\|$	$\|E(G)\|$	L_m	$\|R_i\|$	$\|T_i\|$
2025-1	329	3386	30,00	71	100
2025-2	317	3530	30,00	76	100
2022-1	355	3850	33,00	2	100
2025-1 (ES+SI)	188	1734	23,21	84	100
2025-2 (ES+SI)	185	1812	22,93	96	100
2022-1 (ES+SI)	217	2022	28,17	6	100

Para alguns testes, não foi possível encontrar uma solução dentro do tempo proposto. Como fazemos uso de heurísticas que incorporam a geração de números pseudo-aleatórios, é de se esperar uma distinção na ordem em que a árvore de Zykov é explorada.

É importante explorar a árvore de maneira a evitar entrar em ramos com um espaço de solução pequeno ou nulo, Uma vez que entramos em um ramo desses, o tempo para sair ou encontrar uma solução é proporcional ao tamanho do ramo, que tende a ser muito grande quando os nós sendo explorados estão longe de uma folha.

Outra consideração é que soluções similares entre si, simétricas, residem muito próximas umas das outras nas folhas da árvore de Zykov, dificultando o processo de encontrar ramos promissores.

Observamos, então, uma grande diferença entre a exploração da árvore de Zykov nas instâncias de 2025 e 2022. Uma possível explicação aborda o espaço de solução para 2022, que pode ser bem menor em relação a 2025, diminuindo as chances de explorar ramos promissores. Os dados que apoiam

essa teoria são: a quantidade maior de aulas no grafo original e a quantidade maior de cores utilizadas.

5.3 Pré-processamento

Sabemos que a altura máxima de uma árvore de Zykov é igual a quantidade de arestas restantes para tornar G em $K_{|V(G)|}$ (de Lima and Carmo, 2018). Observando a densidade dos grafos na seção anterior, vemos que todas as instâncias da FACOM constroem grafos esparsos. As etapas de pré-processamento descritas no capítulo 4 procuram remediar isso. Nas duas figuras a seguir, demonstramos como essas rotinas afetam o grafo original após o processamento do nó raiz, começando pela quantidade de vértices.

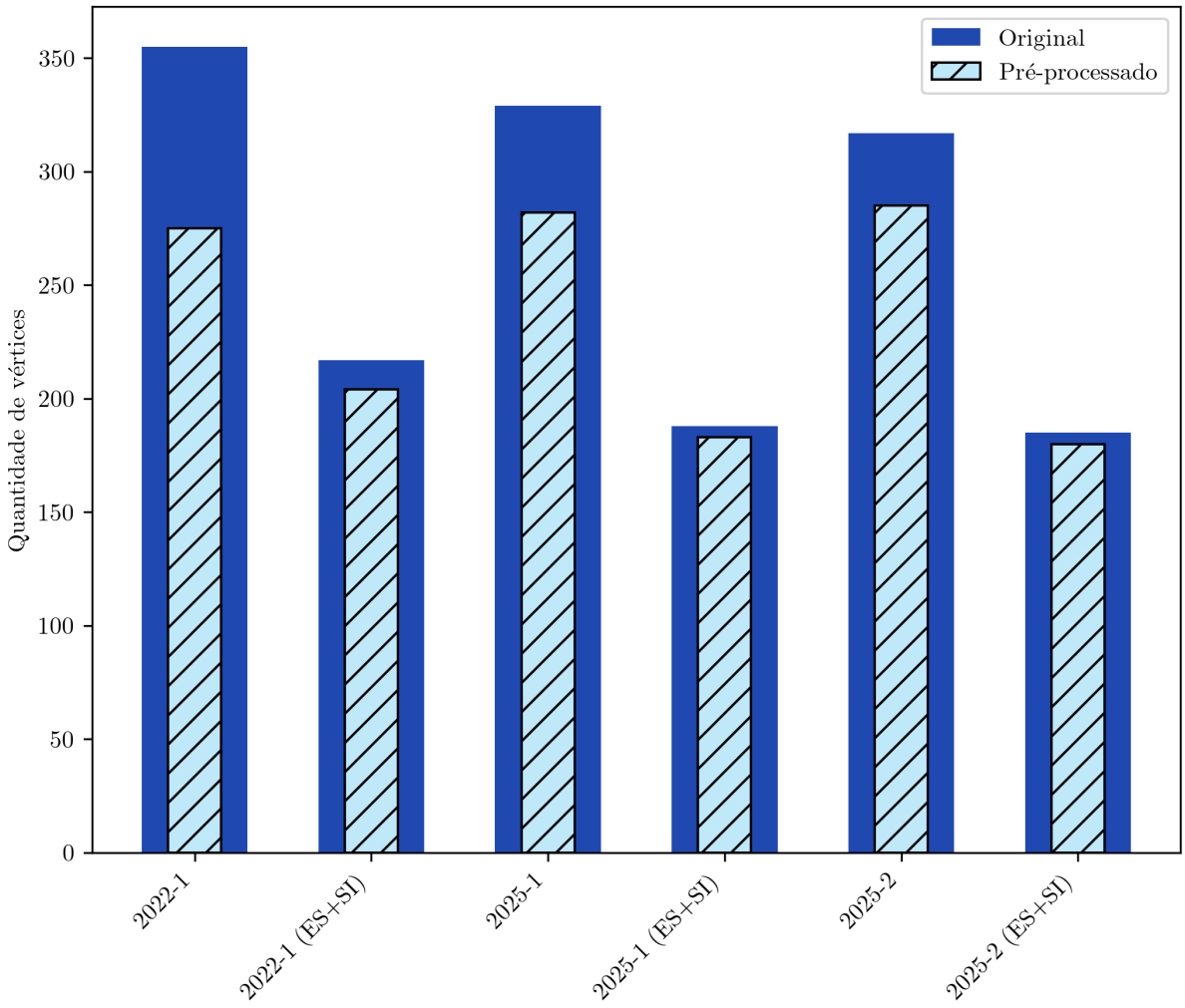


Figura 5.1: Quantidade de vértices antes e depois do pré-processamento do nó raiz

Temos evoluções semelhantes de cada instância em relação ao pré-processamento, indicando relações estruturais semelhantes nos grafos de semestres distintos.
A próxima figura aborda o pré-processamento de adição de arestas.

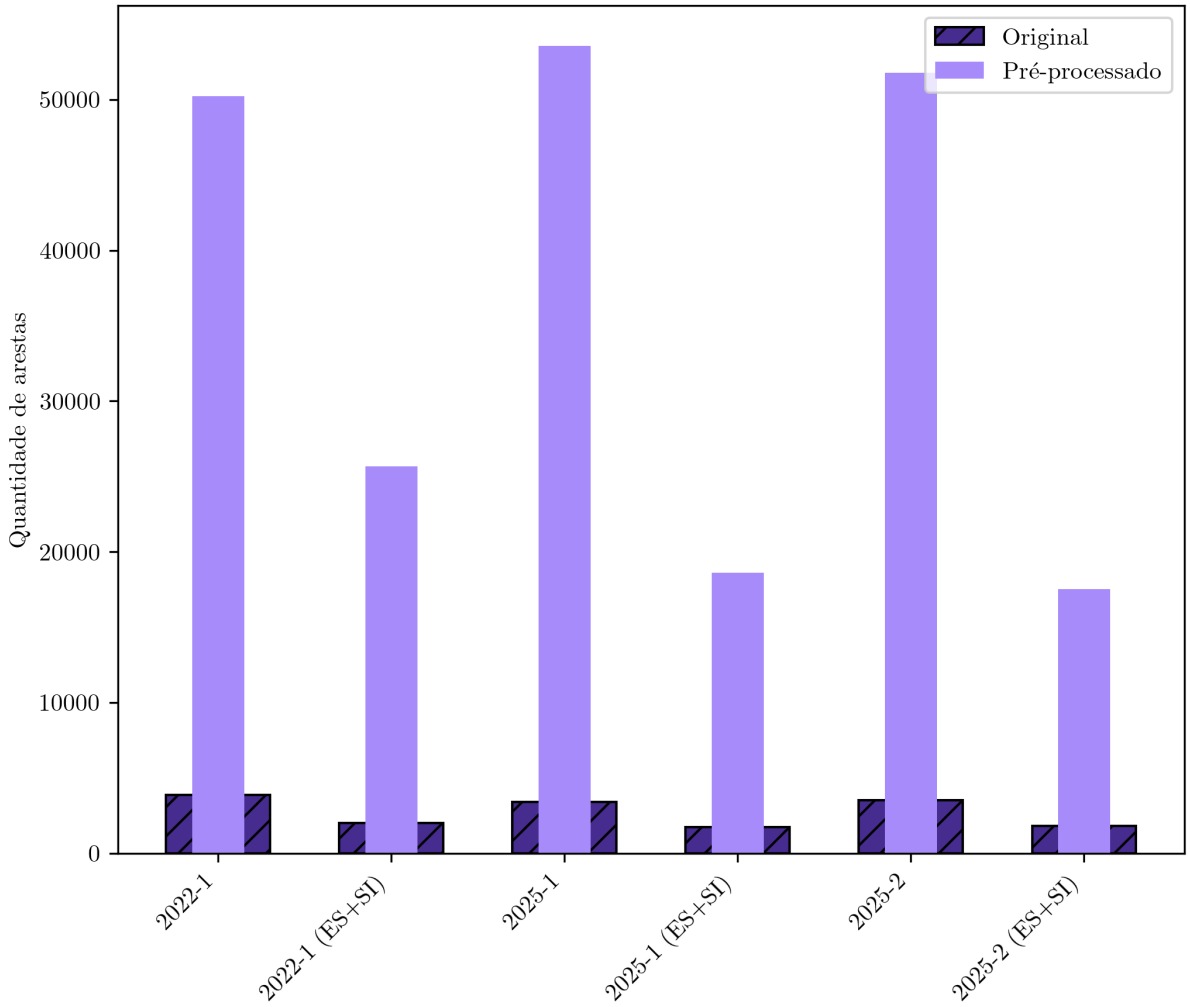


Figura 5.2: Quantidade de arestas antes e depois do pré-processamento do nó raiz

Há uma adição expressiva de arestas em todas as instâncias, esse processo se prova muito útil, uma vez que a contração de dois vértices conectados por essas novas arestas implicaria em uma coloração invalida por conta dos critérios de pré-processamento adotados.

5.4 Tempo de execução

A Figura 5.3 apresenta um diagrama de caixa (*boxplot*) que ilustra a distribuição dos tempos de execução para cada instância, considerando apenas os testes que encontraram solução dentro do limite de *timeout* de 120 segundos.

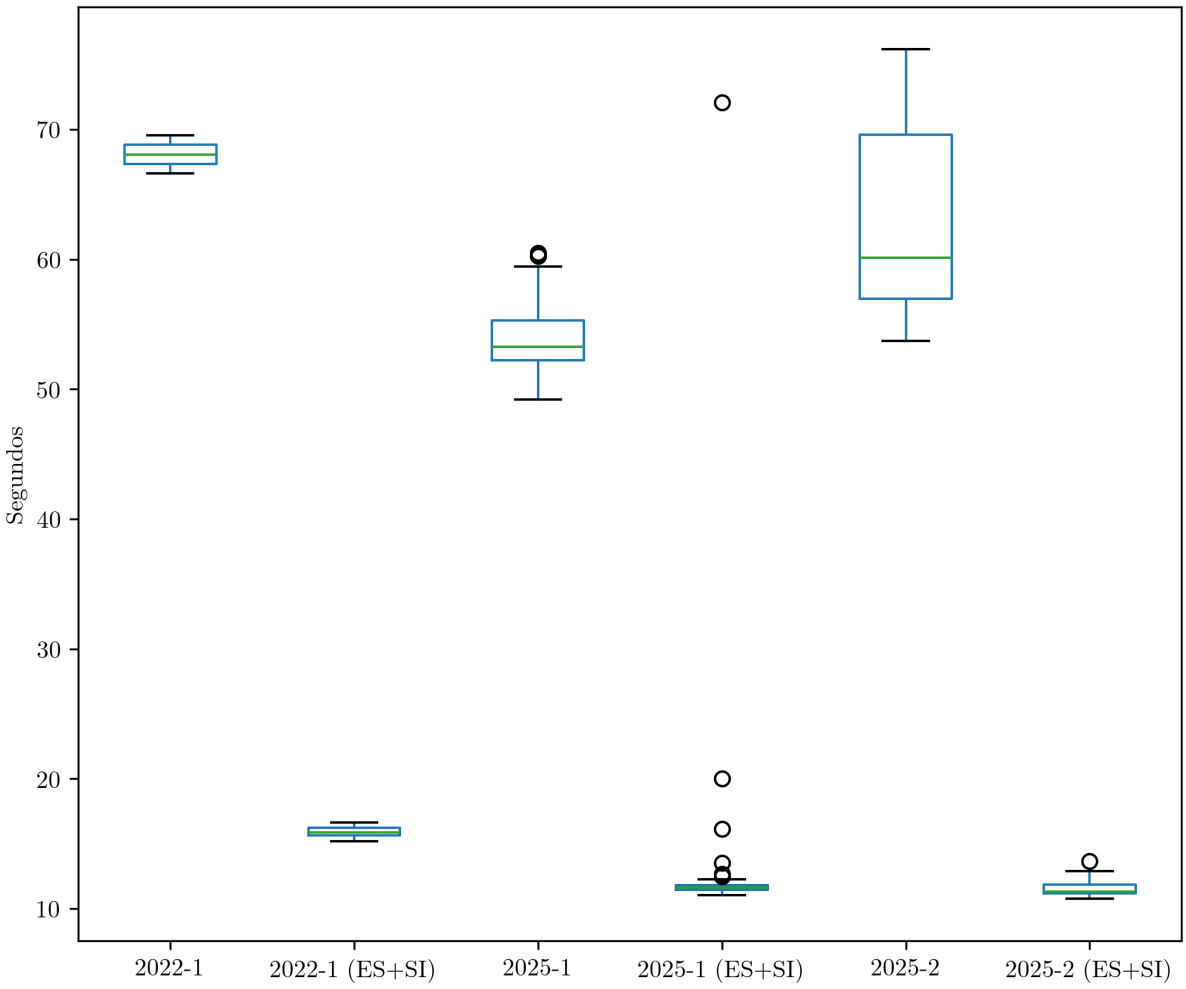


Figura 5.3: Distribuição dos tempos de execução por instância (testes resolvidos).

O desempenho da implementação nos casos em que se encontra uma solução atende aos requisitos de validação da instância, sendo uma possível etapa rápida de pré-validação para o *solver* de programação linear inteira, porém a dificuldade apresentada com as instâncias de 2022 inviabiliza uma aplicação imediata sem maiores polimentos da abordagem atual.

Os *outliers* observados correspondem a casos em que a execução do algoritmo permaneceu por muito tempo explorando um ramo pouco promissor. Como é extremamente improvável que tal ramo seja resolvido antes do *timeout*, o tempo de execução acaba extrapolando os valores típicos do resto da distribuição.

Podemos visualizar o quão improvável é esse caso com a ajuda da figura 5.4 a seguir:

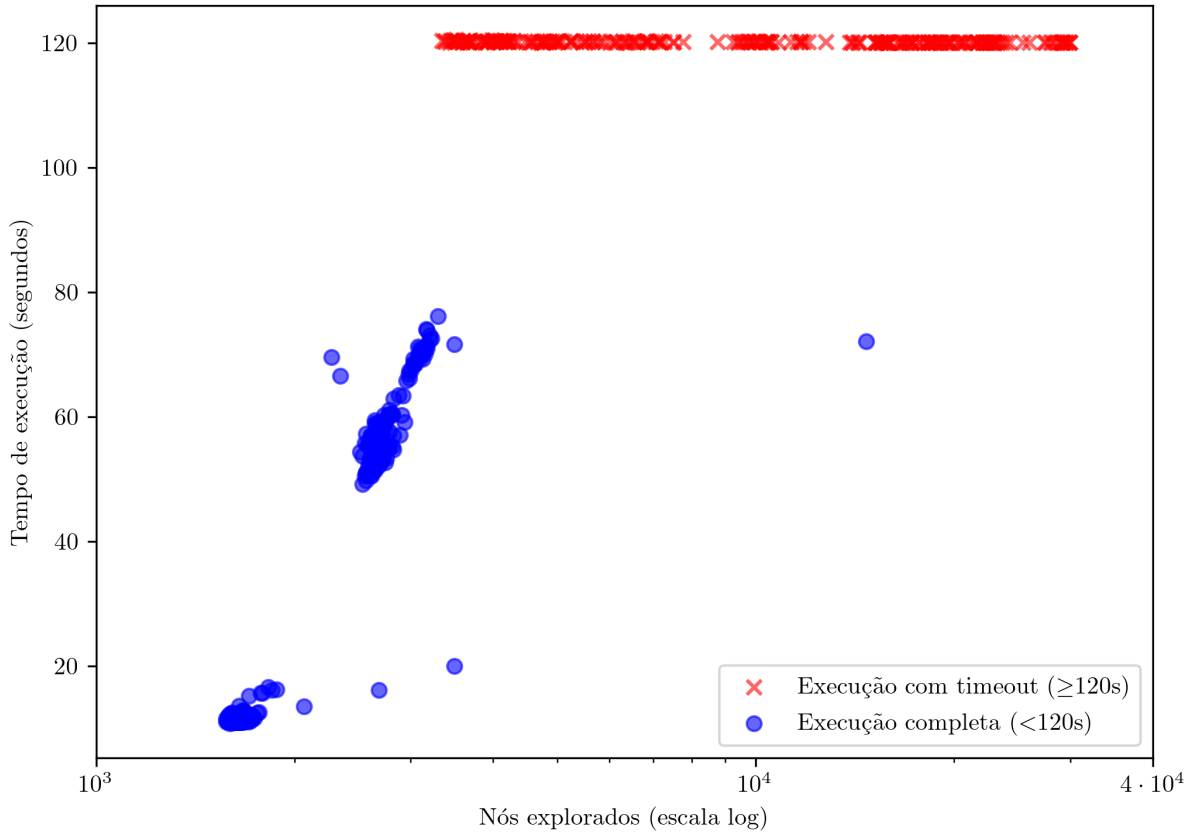


Figura 5.4: Distribuição unificada do tempo de execução de todos os testes

Observa-se que a figura apresenta três regiões bem definidas:

1. **Execuções rápidas das instâncias reduzidas**, com solução encontrada em menos de 20 segundos e baixo número de nós explorados;

2. **Execuções das instâncias completas resolvidas com sucesso**, tipicamente abaixo de 80 segundos, formando o agrupamento intermediário.

3. **Execuções que atingem o *timeout***, todas convergindo para a região superior, com número variado de nós explorados.

O grande afastamento entre essas três regiões ilustra a dificuldade encontrada com ramos da árvore de Zykov que são estruturalmente desfavoráveis, dos quais é difícil escapar.

Os mesmos *outliers* da figura 5.3, são representados aqui, mapeando casos excepcionais dentro do espaço de busca.

5.5 Podas

Finalmente, vamos visualizar as podas durante a execução do algoritmo. Embora todas as podas sejam por invalidez, podemos categorizá-las em duas

partes: poda por invalidez de *L-coloração* e poda por violação da restrição secundária do *ensalamento*.

A figura a seguir, ilustra as podas de todas as 600 execuções, cada execução é representada por dois pontos: um pela quantidade de violações da *L-coloração* e o outro pelo *ensalamento*.

Similarmente a figura 5.4: o eixo X distribui os testes pela quantidade de nós explorados.

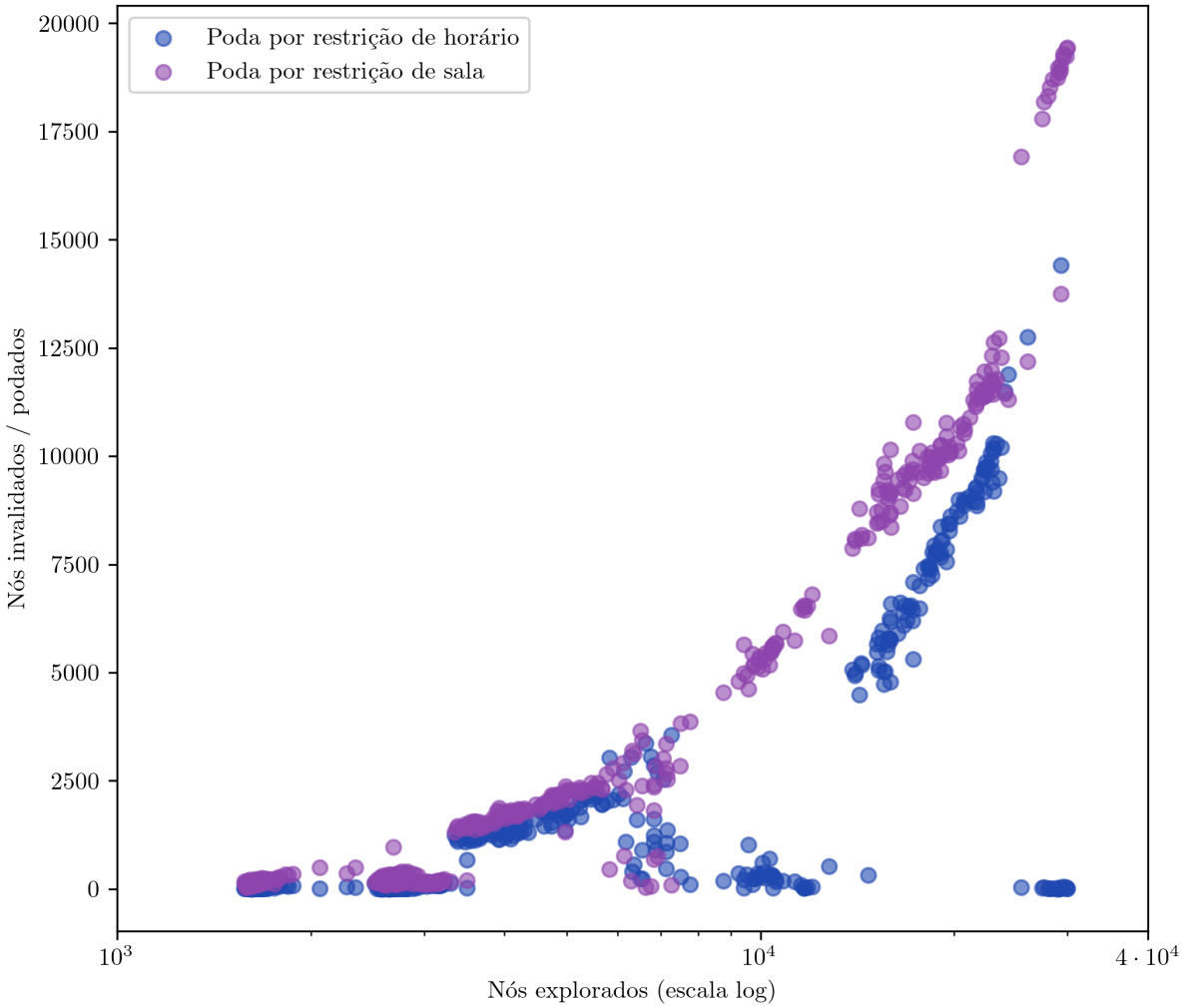


Figura 5.5: Distribuição unificada dos motivos de poda de todos os testes

Como uma das possíveis podas por ensalamento ocorre antes de um nó ser criado, é de se esperar que a curva do ensalamento seja maior na maioria dos casos.

Porém, em alguns casos extremos, a restrição do ensalamento é a única que está sendo repetidamente violada durante a busca, ilustrada pelos pares de pontos com uma grande distância vertical entre si. Surpreendentemente, em certos casos, a poda por *L-coloração* é muito próxima de zero.

CAPÍTULO

6

Contribuições e Conclusões

A partir dos resultados analisados, observamos que essa abordagem é promissora, porém incompleta. A possível incerteza sobre instâncias complexas, como a de 2022-1, serve de empecilho para a adoção da estratégia em sua presente formulação.

Como não foi possível testar instâncias invalidas em tempo hábil, uma análise importante das técnicas implementadas que almejam detectar esses padrões mais rapidamente não pode ser feita. Estes resultados seriam importantes para uma análise mais completa da efetividade de nossa modelagem.

6.1 Trabalhos futuros

Existem diversas frentes que podem ser exploradas, complementando o que foi feito. Todo o conjunto teórico da modelagem mostrou-se sólido, sugerindo que um polimento maior é alcançável por meio do refinamento das heurísticas, tanto na escolha da ordem de exploração da árvore, quanto na construção de cliques para validação.

Um aprofundamento sobre a heurística de seleção de vértices para a ramificação introduzida na seção 4.5 merece atenção especial. A adoção de pesos para os vértices contraídos foi descoberta incidentalmente durante a modelagem da heurística sem pesos, uma vez que os pesos são removidos no fator de ordenação, o desempenho da heurística é degradada, as causas ainda não são totalmente compreendidas.

Além disso, o levantamento de novas etapas de pré-processamento mostraria-se muito útil, sem necessidade de alterar o arcabouço do que foi construído.

Com essas frentes resolvidas, um estudo mais abrangente com dados de

outras unidades da universidade poderia ser feito. A inclusão de instâncias inviáveis seria de grande proveito.

Ademais, uma exploração mais aprofundada da detecção de viabilidade focaria na detecção e apresentação de um recorte do grafo que inviabiliza a instância. Isso auxiliaria o processo como um todo, reduzindo a intervenção manual necessária.

Finalmente, uma integração completa com o *solver* de instâncias que atua na FACOM, acoplada a uma interface gráfica amigável, seria um estado maduro ideal deste trabalho.

Referências

Carter, M. W. (1986). A survey of practical applications of examination timetabling algorithms. *Operations Research*, 34:193–202. Citado nas páginas 8 e 9.

de Lima, A. M. e Carmo, R. (2018). Exact algorithms for the graph coloring problem. *Revista de Informática Teórica e Aplicada*, 25(4):57–73. Citado nas páginas 14 e 23.

de Souza Oliveira, L. (2025a). Heurísticas para cálculo de grandes cliques em grafos. Atividade Orientada de Ensino. Citado na página 18.

de Souza Oliveira, L. (2025b). Validador de instâncias de alocação de disciplinas - facom/ufms. https://github.com/spoyis/TCC. Citado na página 16.

Even, S., Itai, A., e Shamir, A. (1976). On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703. Citado na página 4.

Ford Jr., L. R. e Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404. Citado na página 19.

Harris, J. M., Hirst, J. L., e Mossinghoff, M. J. (2008). *Combinatorics and Graph Theory*. Undergraduate Texts in Mathematics. Springer, New York, 2nd edition. Citado na página 9.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. e Thatcher, J. W., editors, *Complexity of Computer Computations*, pginas 85–103. Plenum Press. Citado na página 6.

Lawler, E. L. e Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719. Citado na página 14.

Mulvey, J. M. (1982). A classroom/time assignment model. *European Journal of Operational Research*, 5:64–70. Citado na página 8.

Neto, A. S. A. e Gomes, M. J. N. (2014). Problema e algoritmos de coloração em grafos - exatos e heurísticos. *Revista de Sistemas e Computação*, 4(2):101–115. Citado na página 7.

Stroustrup, B. (2013). *The C++ Programming Language*. Addison-Wesley, 4th edition. Citado na página 16.

Vizing, V. G. (1976). Coloring the vertices of a graph in prescribed colors. *Diskret. Analiz*, 29:3–10. Citado na página 6.

Welsh, D. J. A. e Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86. Citado na página 8.

Wu, K. e Otoo, E. (2005). A simpler proof of the average case complexity of union-find with path compression. Citado na página 16.

Zykov, A. A. (1949). On some properties of linear complexes. *Matematicheskii Sbornik (New Series)*, 66(2):163–188. Citado nas páginas 4, 6, e 13.