UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

IGOR ALAN ALBUQUERQUE DE SOUSA

ALGORITMOS DE RAMIFICAÇÃO E PODA APRIMORADOS POR APRENDIZADO PROFUNDO NA RESOLUÇÃO DO PROBLEMA DA CLIQUE MÁXIMA

QUIXADÁ

2025

IGOR ALAN ALBUQUERQUE DE SOUSA

ALGORITMOS DE RAMIFICAÇÃO E PODA APRIMORADOS POR APRENDIZADO PROFUNDO NA RESOLUÇÃO DO PROBLEMA DA CLIQUE MÁXIMA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Paulo Henrique Macedo de Araujo.
Coorientador: Prof. Dr. Wladimir Araujo Tavares.

QUIXADÁ

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S696a Sousa, Igor Alan Albuquerque de.
Algoritmos de ramificação e poda aprimorados por aprendizado profundo na resolução do problema da clique máxima / Igor Alan Albuquerque de Sousa. – 2025.
52 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2025.
Orientação: Prof. Dr. Paulo Henrique Macedo de Araújo.
Coorientação: Prof. Dr. Wladimir Araujo Tavares.

1. Neural networks. 2. Combinatorial optimization. 3. Branch-and-bound. I. Título.
CDD 004

IGOR ALAN ALBUQUERQUE DE SOUSA

ALGORITMOS DE RAMIFICAÇÃO E PODA APRIMORADOS POR APRENDIZADO PROFUNDO NA RESOLUÇÃO DO PROBLEMA DA CLIQUE MÁXIMA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: 11 de Julho de 2025.

BANCA EXAMINADORA

Prof. Dr. Paulo Henrique Macedo de Araujo (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Wladimir Araujo Tavares (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Fábio Carlos Sousa Dias
Universidade Federal do Ceará (UFC)

A meu pai e minha mãe.

AGRADECIMENTOS

Ao meu pai, Francisco Riso, que se foi no começo deste semestre e por isso não pôde me ver concluir este curso. Não teria conseguido sem o seu suporte financeiro ao longo de todos estes anos. À minha mãe, Zeina Maria, por ter me incentivado a continuar em todos os momentos.

À Instituição UFC, pelo apoio financeiro com as bolsas PIBIC, PID e P&D.

Ao Prof. Dr. Wladimir Araújo Tavares, por ter sido meu orientador durante a primeira metade do projeto. Ao Prof. Dr. Paulo Henrique Macedo de Araújo, por ter aceitado ser meu orientador com o projeto em andamento, após o professor Wladimir precisar se ausentar da UFC-Quixadá. Agradeço a ambos pela excelente orientação, suporte e paciência.

Aos professores participantes da banca examinadora, Dr. Paulo Henrique Macedo de Araújo, Dr. Fábio Carlos Sousa Dias e Dr. Wladimir Araújo Tavares, pelo tempo dedicado, bem como pelas valiosas colaborações e sugestões.

Aos vários amigos da UFC-Quixadá pelos momentos de lazer juntos, especialmente Lucas Ferreira, Eliabe, Romes, Patrick, Gustavo e Eduardo.

"O caminho de mil milhas começa com um único passo." (Lao Tzu)

RESUMO

Este trabalho aborda o Problema da Clique Máxima (PCM) em grafos, um desafio bem conhecido em teoria dos grafos e otimização combinatória. Os algoritmos exatos consomem muitos recursos computacionais e os algoritmos heurísticos projetados para o PCM dependem de boas soluções iniciais. Portanto, neste trabalho, é considerada a aplicação de métodos de *Machine Learn* (ML) para o resolver PCM. O objetivo principal deste trabalho é usar modelos de ML para decidir a ordem de ramificação e estimar limite superiores para algoritmos de *Branch and Bound* (B&B) que resolvem o PCM. No início, abordamos o PCM e os algoritmos de B&B. Posteriormente, apresentamos a rede neural como uma ferramenta poderosa para a aprendizagem de padrões de ramificação em árvores de busca. Por fim, são apresentados os resultados da integração de ML com o algoritmo de B&B que resolve o PCM.

Palavras-chave: grafos; clique máxima; aprendizado de máquina; redes neurais; branch and bound.

ABSTRACT

This work addresses the Maximum Clique Problem (MCP) in graphs, a well-known challenge in graph theory and combinatorial optimization. Exact algorithms are computationally expensive, and heuristic algorithms designed for MCP rely on good initial solutions. Therefore, the application of machine learning (ML) methods to solve the MCP is considered. The main goal is to use ML models to decide the branching order for Branch and Bound (B&B) algorithms that solve the MCP. Initially, the MCP and B&B algorithms are addressed. Subsequently, neural networks are presented as a powerful tool for learning branching patterns in search trees. Finally, the results of integrating ML with B&B algorithms that solve the MCP are presented.

Keywords: graphs; maximum clique; machine learning; neural networks; branch and bound.

LISTA DE FIGURAS

Figura 1 – Grafo G(V,E), onde V = {0,1,2,3,4,5} e E = {(0,2),(0,4),(0,5),(1,4),(1,5),(2,3),(2,4),(2,5),(4,5)}. 15

Figura 2 – O grafo H subgrafo de G da figura 1, onde V = {0,1,2,3,5} e E = {(0,5),(1,5),(2,3),(2,5)}. 16

Figura 3 – Exemplo de coloração de um grafo. 16

Figura 4 – A clique máxima C do grafo G da figura 1. 17

Figura 5 – Arquitetura *artificial neural network* (ANN) com duas camadas intermediárias. 20

Figura 6 – Representação do cálculo realizado por um neurônio. 21

Figura 7 – Fluxograma de etapas da metologia. 32

Figura 8 – Representação do nó inicial do da árvore de busca do grafo da figura 1. A matriz A é a matriz de adjacência, o vetor **c** indica que nenhum vértice pertence à clique inicial, e o vetor **y** mostra os vértices que podem ser adicionados para formar a clique máxima. 34

Figura 9 – Representação do nó seguinte da árvore de busca da figura 8, ao adicionar o vértice 0 à clique. A matriz A é a matriz de adjacência, preenchida com 0s para o próprio vértice adicionado e para os vértices que não são seus vizinhos. O vetor **c** contém 1 na posição do vértice 0, que foi adicionado à clique. O vetor **y** mostra os vértices que podem ser adicionados para formar a clique máxima. 35

Figura 10 – Distribuição de posições do vértice verdadeiro no ranking predito pelo modelo de *branch*. 39

Figura 11 – Dispersão entre valores reais e preditos pelo modelo de *bound*. 41

LISTA DE TABELAS

Tabela 1 – Numero de nós na árvore de busca e tempo de execução 28
Tabela 2 – Comparativo de desempenho entre o algoritmo MCBB_DLTS e BitClique. . 44

LISTA DE ABREVIATURAS E SIGLAS

ANN	*artificial neural network*
B&B	*Branch and Bound*
DFS	*depth first search*
DLTS	*Deep Learning assisted heuristic Tree Search*
DNN	*Deep Neural Networks*
ML	*Machine Learn*
PCM	Problema da Clique Máxima
PPMC	Problema do pré-manuseio de contêineres
SVM	*support vector machine*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Objetivo Geral 14
1.1.1 Objetivos Específicos 14
1.2 Estrutura do Trabalho 14
2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 Grafos 15
2.1.1 Clique 16
2.2 Otimização 17
2.2.1 Limites 18
2.2.2 Branch and Bound 18
2.2.3 Busca em Profundidade 18
2.3 Aprendizado de Máquina 19
2.3.1 Aprendizado Supervisionado 19
2.3.2 Redes Neurais Artificiais 19
2.3.3 Aprendizado Profundo 21
2.3.4 Métricas de Avaliação 22
3 TRABALHOS RELACIONADOS 25
3.1 Solução Exata do Problema da Clique Máxima 25
3.2 Deep Learning Assisted Heuristic Tree Search for the Container Pre-marshalling Problem 26
3.2.1 Algoritmo DLTS 26
3.3 Algoritmos para o problema da clique máxima com paralelismo de bits . 27
3.3.1 Branch and Bound com RoaringBitmap 28
4 METODOLOGIA 30
4.1 Dados de treinamento 30
4.2 Treinamento do Modelo 30
4.3 Teste de Eficiência 31
5 DLTS PARA O PROBLEMA DA CLIQUE MÁXIMA 33
5.1 Estrutura dos dados 33
5.2 Camada de entrada 34

5.3 **Camadas densas compartilhadas** 35
5.4 **Camadas ocultas** 36
5.5 **Camada de saída** 36
5.5.1 ***Camada de saída para a rede de branch*** 36
5.5.2 ***Camada de saída para a rede de bound*** 37
6 **RESULTADOS** 38
6.1 **Métricas do modelo de branch** 38
6.2 **Métricas do modelo de bound** 40
6.3 **MCBB_DLTS** 41
6.4 **Comparação entre algoritmos** 43
7 **CONCLUSÕES E TRABALHOS FUTUROS** 47
REFERÊNCIAS 49

1 INTRODUÇÃO

Um grafo é um par (V,E) em que V é um conjunto finito qualquer e $E \subseteq \{(u,v) \mid u,v \in V\}$. Os elementos de V são chamados **vértices** e os de E são chamados **arestas** (Feofiloff *et al.*, 2011). Uma clique de um grafo $G = (V,E)$, é um subconjunto C de V tal que todo par de elementos (u,v) de C pertence a E. Em outras palavras, em uma clique, todos os vértices estão interconectados, formando um grafo induzido completo. Uma clique máxima é uma clique ao qual não podem ser adicionados mais vértices sem quebrar a propriedade de uma clique. (West, 2000).

O problema da clique máxima (PCM) consiste em encontrar uma clique máxima em um grafo. Esse problema é NP-Completo, o que implica que não há uma solução em tempo polinomial para o problema, a menos que P = NP (Bomze *et al.*, 1999). Devido a essa característica, algoritmos exatos têm a garantia de retornar uma solução somente em um tempo que aumenta exponencialmente com o tamanho da entrada e e algoritmos heurísticos dependem de boas informações prévias, como por exemplo limites superiores ou inferiores.

Apesar do grande desafio que é sua complexidade computacional, o problema é amplamente estudado pois têm implicações abrangentes em diversas áreas do meio acadêmico e além. Por exemplo, segundo Tomita e Seki (2003), a redução no tempo necessário para resolver o PCM será de grande benefício para pesquisadores e profissionais em campos como bioinformática, processamento de imagens, design de circuitos quânticos e no projeto de sequências de DNA e RNA para a computação biomolecular. Além disso, de acordo com Reba *et al.* (2021), encontrar a clique máxima de maneira eficiente trará impacto direto na área de descoberta de medicamentos, onde a análise de estruturas de proteínas é essencial.

Dentre os diversos estudos com a clique máxima, avanços significativos têm sido alcançados na aplicação de técnicas de *Machine Learn* (ML) em problemas relacionados a clique máxima. Um exemplo notável é o estudo intitulado *Using Statistical Measures and Machine Learning for Graph Reduction to Solve Maximum Weight Clique Problems* (Sun *et al.*, 2021), que empregou a técnica de *support vector machine* (SVM) para redução de um programa linear binário gerado a partir de um grafo.

Outro trabalho relevante é o estudo conduzido por Hottung *et al.* (2017), que investigou a aplicação de técnicas de ML para otimizar algoritmos de resolução do Problema do pré-manuseio de contêineres (PPMC). Esse estudo demonstrou que redes neurais profundas podem ser treinadas para reconhecer padrões de ramificação e estimar limites em algoritmos de

Branch and Bound (B&B), resultando em melhorias significativas na eficiência desses algoritmos. Este trabalho, por sua vez, adapta essa abordagem ao contexto do PCM, aplicando as mesmas ideias para identificar ordens de ramificação e poda que otimizem a resolução do problema.

Apesar desses avanços, ainda há espaço para contribuições valiosas nessa área. A complexidade computacional inerente ao PCM continua a destacar a necessidade de novas abordagens que combinem eficiência e eficácia. Este trabalho, portanto, justifica-se pela busca de soluções que empreguem técnicas modernas de ML para otimizar algoritmos já existentes, com foco em reduzir o tempo de resolução do PCM e explorar novas possibilidades na aplicação de aprendizado profundo.

1.1 Objetivo Geral

O objetivo geral deste trabalho é integrar técnicas de aprendizado profundo a algoritmos de B&B para resolver o PCM. Redes neurais profundas serão utilizadas para a classificação de ramos mais promissores para uma clique máxima e estimar limites que auxiliem na redução do espaço de busca.

1.1.1 Objetivos Específicos

- Investigar a eficácia do modelo de ML em classificar ramos mais relevantes para a clique máxima e estimar limites superiores para os ramos da árvore de busca;
- Investigar a eficiência da abordagem em reduzir os tempos de execução para encontrar a clique máxima;

1.2 Estrutura do Trabalho

O Capítulo 2 apresenta a Fundamentação Teórica, dividida nas principais seções: Grafos, Otimização e Aprendizado de Máquina. O Capítulo 3 discute os Trabalhos Relacionados. O Capítulo 4 descreve os Procedimentos Metodológicos utilizados neste trabalho. O Capítulo 5 aprensenta a estrutura da rede neural usado neste trabalho. O Capítulo 6 expõe os Resultados obtidos. Finalmente, o Capítulo 7 trata da Conclusão e das Perspectivas para Trabalhos Futuros.

2 FUNDAMENTAÇÃO TEÓRICA

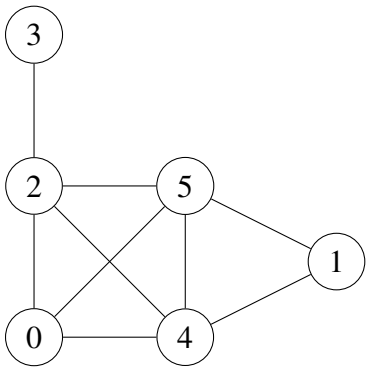
Neste capítulo, são apresentadas as bases teóricas e os conceitos fundamentais necessários para o desenvolvimento deste trabalho. As definições e explicações aqui reunidas fornecem suporte para compreender as técnicas, modelos e algoritmos utilizados ao longo da pesquisa.

2.1 Grafos

As definições apresentadas nesta seção tem como base as explicações do livro (West, 2000), onde são definidos os conceitos de grafos abordados neste trabalho.

Um grafo simples G é um par consistindo de um conjunto finito de vértices $V(G)$ e um conjunto de arestas $E(G)$ tal que $E \subseteq \{(u,v) : u,v \in V, u \neq v\}$. Neste trabalho trataremos apenas de grafos simples, não direcionados e não ponderados, isto significa que não podem existir múltiplas arestas para um mesmo par de vértices, $(u,v) = (v,u)$ e que todas as arestas possuem o mesmo peso, respectivamente. Na Figura 1, podemos ver uma representação visual de um exemplo desses grafos:

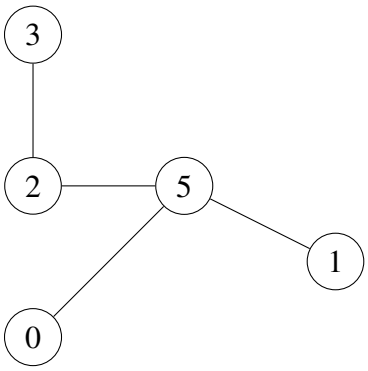
Figura 1 – Grafo G(V,E), onde V = {0,1,2,3,4,5} e E = {(0,2),(0,4),(0,5),(1,4),(1,5),(2,3),(2,4),(2,5),(4,5)}.



Fonte: Autor.

Podemos interpretar os vértices como os pontos no grafo e as arestas como as conexões entre os vértices. Consideremos um vértice $v \in V$ e uma aresta $e \in E$, se $v \in e$, então dizemos que e incide em v. Denotaremos por $\deg(v)$ o grau de um vértice v, isto é o número de arestas que indicidem neste vértice. Dois vértices $u,v \in V$ são adjacentes se $(u,v) \in E$. Um subgrafo de um grafo G é um grafo H de modo que $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Na Figura 2, temos um subgrafo H de G.

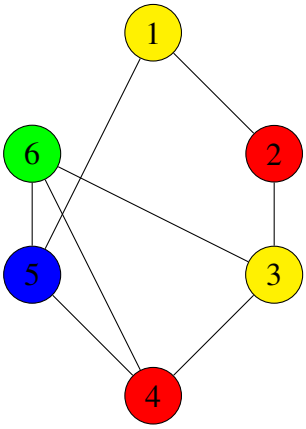
Figura 2 – O grafo H subgrafo de G da figura 1, onde V = {0,1,2,3,5} e E = {(0,5),(1,5),(2,3),(2,5)}.



Fonte: Autor.

S é um conjunto independente em G se, e somente se, $\forall u, v \in S$, $u, v \notin E(G)$. Uma k-coloração dos vértices de G é uma partição de V em k conjuntos independentes: $V = V_1, V_2, \ldots, V_k$. Na Figura 3, temos um exemplo de coloração de um grafo.

Figura 3 – Exemplo de coloração de um grafo.



Fonte: (Sousa; Tavares, 2024).

2.1.1 Clique

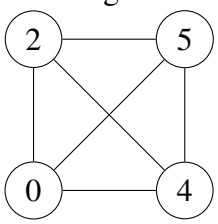
Definimos uma clique C de um grafo G como $C \subseteq V(G), \forall u, v \in C, u \neq v \Rightarrow (u, v) \in E(G)$. Em palavras, uma clique em um grafo é um conjunto de vértices onde todo par de vértices é adjacente. Uma clique C em um grafo G é chamada de clique máxima se não existe outra clique C' em G tal que $|C'| > |C|$, onde $|C|$ representa o número de vértices na clique C.

O PCM consiste em encontrar a maior clique em um grafo, ou seja, identificar o maior subconjunto de vértices no qual todos estão interconectados, formando um grafo induzido completo. Do ponto de vista computacional, isso pode ser um desafio, uma vez que é necessário

verificar todas as combinações possíveis de subconjuntos de vértices para determinar qual deles forma a clique máxima. Na Figura 4, é apresentado um exemplo de clique máxima em um grafo.

É importante destacar que o PCM é equivalente a encontrar o **conjunto independente máximo** no **grafo complementar**. O *grafo complementar* de um grafo G é um grafo G' onde dois vértices estão conectados por uma aresta em G' se, e somente se, não estiverem conectados por uma aresta em G. Um *conjunto independente* é um subconjunto de vértices em que nenhum par de vértices está conectado por uma aresta. Assim, encontrar o conjunto independente máximo no grafo complementar é equivalente a encontrar a clique máxima no grafo original, devido à relação inversa entre as arestas de G e G'.

Figura 4 – A clique máxima C do grafo G da figura 1.



Fonte: Autor.

2.2 Otimização

Otimização é uma área da matemática e da ciência da computação que se concentra na busca de soluções ótimas para problemas, sujeitas a determinadas restrições (Izmailov; Solodov, 2020). Ela desempenha um papel importante em uma ampla gama de aplicações do mundo real, desde o planejamento de rotas de transporte até o design de redes de comunicação.

Problemas de otimização combinatória em larga escala são comuns no mundo real (Sun *et al.*, 2021). Esses problemas são desafiadores de resolver, em parte devido ao grande espaço de busca e à complexidade NP. Solucionadores exatos, em muitos casos, não conseguem lidar com o grande tamanho do problema, e métodos heurísticos podem facilmente ficar presos em um ótimo local, resultando em um valor final ruim.

De acordo com Sun *et al.* (2021), uma maneira lógica de lidar com problemas de otimização em larga escala é usar a redução de problemas, ou seja, reduzir o espaço de busca de um problema. Isso pode ser feito removendo variáveis de decisão ou etapas computacionais. A clique máxima de um grande grafo geralmente consiste em uma pequena proporção de vértices. Os outros vértices são variáveis redundantes, que principalmente retardam o processo

de otimização.

2.2.1 Limites

Dado um problema de maximização P, um limite superior é um valor z tal que $z \geq \overline{z}$ para $\overline{z}$ um valor de uma solução ótima de P. Dado um problema de maximização P, um limite inferior é um valor z tal que $z \leq \overline{z}$ sendo que $\overline{z}$ é a solução ótima de P (Zuge, 2012).

Estratégias baseadas em limites superiores e inferiores se baseiam na restrição do espaço de busca, utilizando informações disponíveis sobre o intervalo onde a solução ótima pode ser encontrada (Zuge, 2012). Um exemplo prático dessa eficiência é observado em algoritmos de B&B, usados para resolver o PCM por meio da poda do espaço de busca.

2.2.2 Branch and Bound

Os algoritmos de B&B são técnicas de busca que dividem o espaço de busca do problema em subconjuntos menores, chamados de ramos, e aplicam estratégias para reduzir a quantidade de ramos a serem explorados (Zuge, 2012). No contexto do PCM, limites superiores como coloração são estrategias válidas para reduzir o número de ramos a serem explorados.

Em Hottung *et al.* (2017) foi apresentado o algoritmo conhecido como *Deep Learning assisted heuristic Tree Search* (DLTS), voltado para a resolução do desafio de PPMC. Se trata de um algoritmo de B&B de *depth first search* (DFS). Não fez uso de heurísticas tradicionais encontradas na literatura. Em vez disso, a tomada de decisões sobre a ordem de ramificação e poda dos ramos na árvore de busca foi realizada por meio de técnicas de aprendizado profundo.

2.2.3 Busca em Profundidade

O *depth first search* (DFS) é um algoritmo de busca utilizado em estruturas de dados como grafos e árvores. Ele explora tão profundamente quanto possível ao longo de cada ramificação antes de retroceder (Cormen *et al.*, 2012). Quando não é possível encontrar mais vértices não visitados se "regressa pelo mesmo caminho" para explorar vértices não visitados em outro ramo. Esse processo continua até descobrirmos todos os vértices que podem ser visitados a partir do vértice fonte inicial.

2.3 Aprendizado de Máquina

Machine Learn (ML) é uma subárea da inteligência artificial que se concentra no desenvolvimento de algoritmos e modelos que permitem que sistemas computacionais aprendam e melhorem seu desempenho em tarefas específicas à medida que são expostos a mais dados (Nilsson, 1996). Em vez de serem explicitamente programados para realizar uma tarefa, os sistemas de ML usam dados e experiência para automatizar a tomada de decisões, identificar padrões e fazer previsões.

As abordagens de ML, como aprendizado supervisionado, fornecerem boas soluções para muitos problemas, mas elas possuem diversas limitações (Bishop, 2006). Os modelos gerados por essas abordagens de ML não garantem a solução ótima, uma vez que se baseiam em previsões a partir dos dados. Em outras palavras, eles buscam encontrar soluções que sejam boas o suficiente com base nas informações disponíveis, mas não necessariamente a solução absolutamente ideal.

2.3.1 Aprendizado Supervisionado

O aprendizado supervisionado é uma abordagem de ML na qual um modelo de ML é treinado usando um conjunto de dados que contém exemplos rotulados, ou seja, pares de entrada e saída correspondentes (Nilsson, 1996). O objetivo do aprendizado supervisionado é aprender uma função que mapeia as entradas para as saídas desejadas, de modo que o modelo de ML possa fazer previsões precisas para novos dados não vistos. Para isso, emprega-se um algoritmo, como uma rede neural, que ajusta os parâmetros do modelo de ML por meio da comparação entre as previsões e os rótulos, visando à minimização dos erros associados.

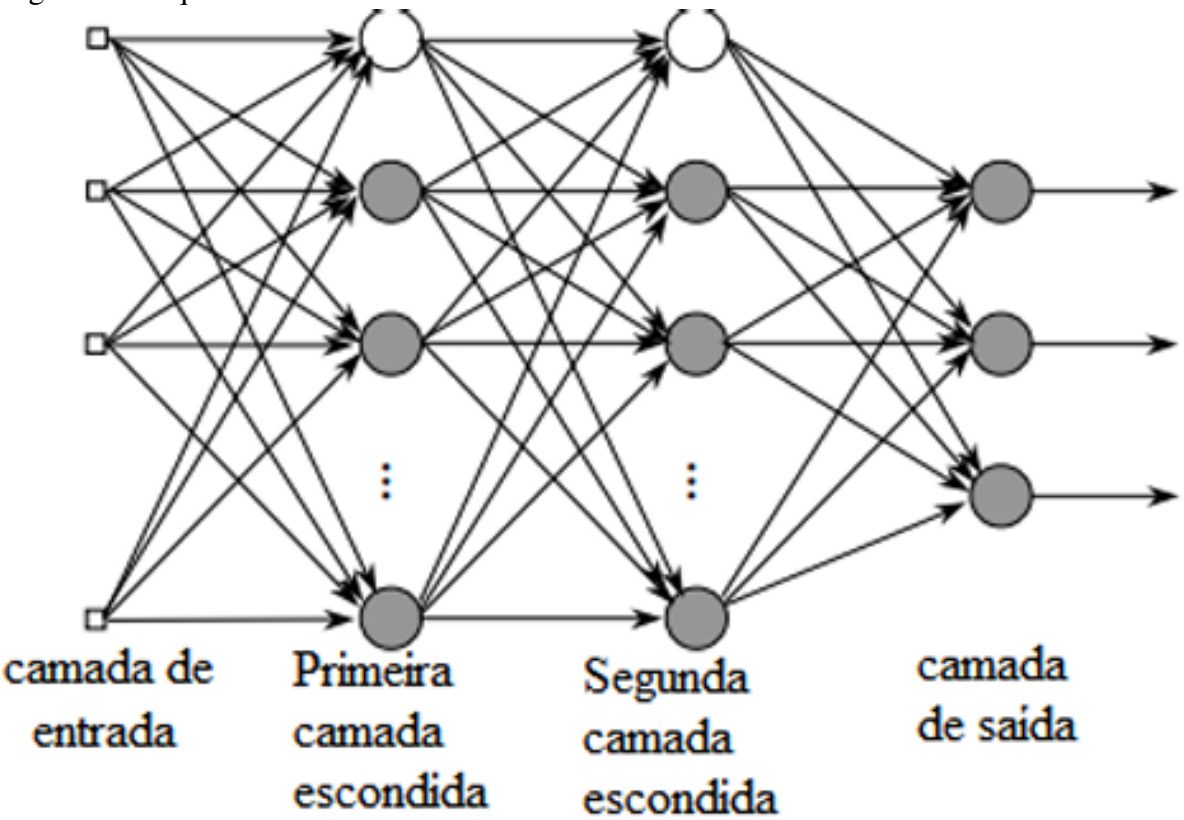
2.3.2 Redes Neurais Artificiais

Artificial Neural Network (ANN) são modelos de aprendizado de máquina supervisionado inspirados no funcionamento do cérebro humano (Fleck *et al.*, 2016). Elas consistem em um conjunto de unidades de processamento interconectadas, chamadas neurônios artificiais, que são organizadas em camadas. As ANN são projetadas para aprender a partir de dados por meio do ajuste de pesos nas conexões entre os neurônios (Fleck *et al.*, 2016).

Uma ANN típica é composta por três camadas: a camada de entrada, que recebe os dados de entrada; uma ou mais camadas ocultas, onde ocorre o processamento interno; e a

camada de saída, que produz os resultados ou previsões (Fleck *et al.*, 2016). Cada camada é constituída por um conjunto de neurônios. Na Figura 5 é possível observar uma arquitetura de ANN com duas camadas intermediárias.

Figura 5 – Arquitetura ANN com duas camadas intermediárias.



Fonte: (Fleck *et al.*, 2016).

De acordo com Fleck *et al.* (2016), supondo que temos n entradas, o neurônio k calcula a sua saída y_k dada pela fórmula:

$$y_k = AF\left(\sum_{i=1}^{n} w_{ki} \cdot x_i + b_k\right)$$

Onde: Yi é a saída gerada pelo neurônio i, enquanto Wki denota o peso sináptico entre os neurônios i e k. O valor bk representa o bias, que é um peso associado a uma constante diferente de zero e está ligado ao neurônio k. Caso o neurônio esteja na camada de entrada, o termo yi é substituído pela entrada correspondente.

O treinamento é processo de estimar os pesos sinápticos e o bias. Se trata de um processo iterativo onde os parâmetros iniciais são ajustados até a convergência do processo.

De acordo com Fleck *et al.* (2016), considerando a iteração j, o peso w_{ki} é atualizado

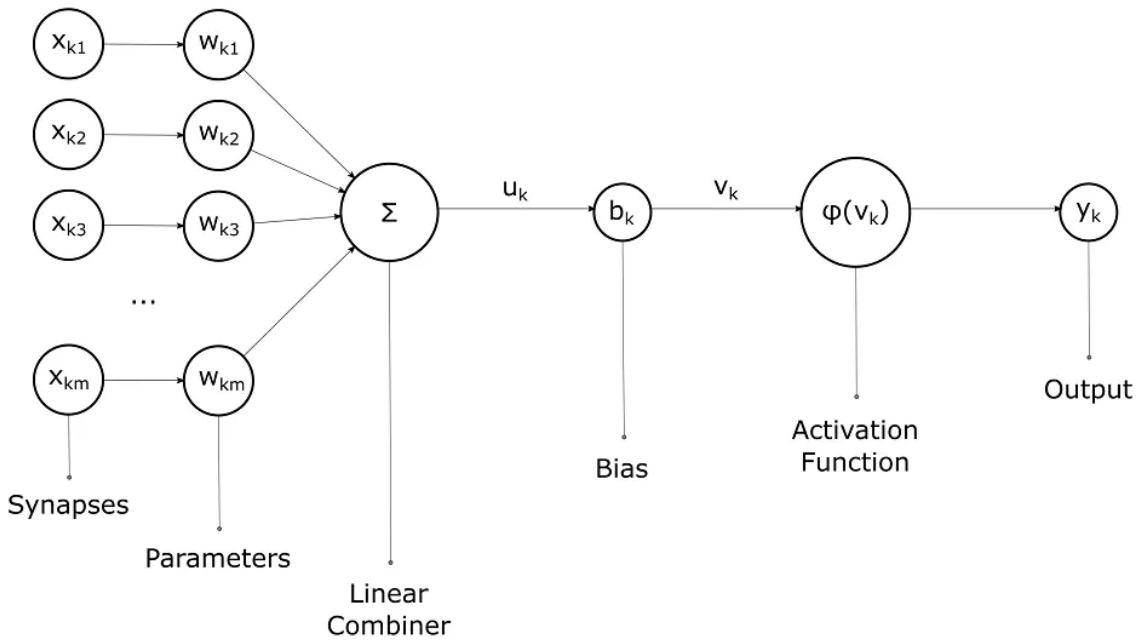
de acordo com a seguinte fórmula:

$$w_{ki}(j+1) = w_{ki}(j) + \Delta w_{ki}(j)$$

onde $\Delta w_{ki}(j)$ é o vetor de correção para o parâmetro w_{ki} na iteração j.

Após calcular o somatório, aplica-se uma função de ativação AF para produzir uma saída. A função de ativação determina se o neurônio será ativado ou não com base no valor de y_k. Uma função de ativação comum é a Rectified Linear Unit (ReLU), dada por $f(z) = max(0,z)$. A Figura 6 representa o cálculo realizado por um neurônio.

Figura 6 – Representação do cálculo realizado por um neurônio.



Fonte: Imagem extraída de Rowe (2019), "Introduction to Neurons in Neural Networks". Medium. Disponível em: https://medium.com/artificial-neural-networks/introduction-to-neurons-in-neural-networks-71828d040a65. Acesso em: 30 dez. 2024.

2.3.3 Aprendizado Profundo

O *Deep Learning* (Aprendizado Profundo) é uma subárea do aprendizado de máquina que utiliza ANN com múltiplas camadas ocultas para extrair representações hierárquicas e complexas dos dados (Goodfellow *et al.*, 2016). Essas redes, chamadas de *Deep Neural Networks* (DNN), são capazes de modelar relações altamente não-lineares entre as variáveis de entrada e saída.

Matematicamente, uma DNN com L camadas pode ser descrita como uma composição de funções:

$$f(x) = f^{(L)}\left(f^{(L-1)}\left(\dots f^{(1)}(x)\right)\right)$$

Onde cada função $f^{(l)}$, para $l = 1, \dots, L$, representa a operação realizada pela camada l, incluindo transformações lineares seguidas de funções de ativação não-lineares. Para uma camada l, com vetor de entrada $\mathbf{h}^{(l-1)}$, os pesos $\mathbf{W}^{(l)}$ e bias $\mathbf{b}^{(l)}$, o cálculo da saída $\mathbf{h}^{(l)}$ é dado por:

$$\mathbf{h}^{(l)} = AF^{(l)}\left(\mathbf{W}^{(l)} \cdot \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right)$$

A camada de entrada corresponde a $\mathbf{h}^{(0)} = \mathbf{x}$, onde $\mathbf{x}$ é o vetor de entrada da rede. A camada final $\mathbf{h}^{(L)}$ fornece a saída do modelo.

Durante o treinamento, utiliza-se o algoritmo de retropropagação para calcular o gradiente do erro em relação aos pesos da rede. Esse erro é então utilizado por um otimizador, como o Gradiente Descendente, para atualizar os parâmetros e minimizar uma função de custo $J(\theta)$:

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta J(\theta)$$

Onde θ representa o conjunto de todos os pesos e bias da rede, η é a taxa de aprendizado e $\nabla_\theta J(\theta)$ é o gradiente da função de custo em relação aos parâmetros.

As funções de ativação não-lineares, como ReLU, sigmoid ou *tanh*, são essenciais para que a rede possa aprender representações complexas. Sem elas, a composição de várias camadas se comportaria como uma transformação linear, o que limitaria a expressividade do modelo.

2.3.4 Métricas de Avaliação

Para avaliar o desempenho de modelos de ML, é comum a utilização de diferentes métricas, dependendo da natureza da tarefa. Essas métricas permitem quantificar, de forma objetiva, a precisão, a robustez e a qualidade das predições realizadas pelos modelos. A seguir, são apresentadas as definições formais das principais métricas adotadas neste trabalho, bem como os tipos de visualizações empregados para interpretar os resultados.

Top-k Accuracy

A métrica **Top-k Accuracy** indica a proporção de vezes em que a classe correta está entre as k primeiras posições do ranking predito. Seja y_i o índice da classe verdadeira na i-ésima amostra e $\hat{r}_i$ o ranking de predições para essa amostra, definimos:

$$\text{Accuracy@}k = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\left(y_i \in \text{Top-}k\left(\hat{r}_i\right)\right) \tag{2.1}$$

onde $\mathbb{1}(\cdot)$ é a função indicadora e n é o número total de amostras.

Mean Reciprocal Rank (MRR)

A métrica **MRR** calcula o inverso da posição do item correto no ranking predito e tira a média sobre todas as amostras:

$$\text{MRR} = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\text{rank}_i} \tag{2.2}$$

onde rank_i é a posição (baseada em 1) da classe correta no ranking da i-ésima amostra.

Rank Relativo Médio

O **Rank Relativo Médio** considera a posição da classe correta normalizada em relação ao número total de classes. Dado N como o número de classes, temos:

$$\text{Rank Relativo}_i = \frac{N - \text{rank}_i}{N - 1} \tag{2.3}$$

$$\text{Rank Relativo Médio} = \frac{1}{n}\sum_{i=1}^{n} \text{Rank Relativo}_i \tag{2.4}$$

Valores próximos de 1 indicam melhores posições relativas.

Erro Absoluto Médio (MAE)

O **MAE** mede a média dos erros absolutos entre os valores preditos $\hat{y}_i$ e os valores reais y_i:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} \left|y_i - \hat{y}_i\right|. \tag{2.5}$$

Erro Quadrático Médio (MSE)

O **MSE** penaliza erros maiores ao elevar a diferença ao quadrado:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 . \tag{2.6}$$

Raiz do Erro Quadrático Médio (RMSE)

O **RMSE** é a raiz quadrada do MSE, mantendo a unidade original dos dados:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}. \tag{2.7}$$

Coeficiente de Determinação (R^2)

O R^2 mede a proporção da variância dos valores reais explicada pelo modelo:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2.8}$$

em que $\bar{y}$ é a média dos valores reais. Um valor de R^2 próximo de 1 indica boa capacidade preditiva.

Gráfico de Dispersão

Um **gráfico de dispersão** é uma representação visual que exibe os pares $(y_i, \hat{y}_i)$ para cada amostra, permitindo verificar quão próximas estão as predições dos valores reais. Uma reta $y = \hat{y}$ é frequentemente incluída como referência para predição perfeita.

Gráfico de Distribuição de Posições

O **gráfico de distribuição de posições** mostra a frequência com que a classe correta aparece em cada posição do ranking predito. Essa visualização permite observar a tendência do modelo em priorizar corretamente os melhores candidatos.

3 TRABALHOS RELACIONADOS

Esta seção tem como objetivo apresentar e discutir pesquisas anteriores que são relevantes para este trabalho.

3.1 Solução Exata do Problema da Clique Máxima

O estudo de (Zuge, 2012) aborda uma série de testes em algoritmos de B&B para o PCM. É apresentando um algoritmo base para a resolução do PCM, que serve como alicerce para várias variações discutidas no texto, cada uma adaptada a diferentes tipos de grafos.

A implementação tanto do algoritmo base quanto de suas variações permitiu uma avaliação empírica robusta, resultando em dados experimentais que facilitam a comparação direta do desempenho entre as diferentes abordagens. Essa análise empírica não apenas confirma a eficácia do algoritmo base, mas também fornece insights valiosos sobre as estratégias que podem ser adotadas para enfrentar o PCM em uma variedade de contextos.

Algoritmo 1: Algoritmo base para o PCM - fonte: (Zuge, 2012)

```
function MCBB(G):
    (C,S) ← pre_process_instance(G)
    while S ≠ ∅ do
        (Q,K) ← pop(S)
        pre_process_state(G,Q,K,C)
        while K ≠ ∅ e |C| < |Q| + upper_bound(G,K,C) do
            v ← remove_vertex(G,K)
            push(S,(Q,K))
            (Q,K) ← (Q ∪ {v}, K ∩ Γ_G(v))
            pre_process_state(G,Q,K,C)
        end
        if |C| < |Q| then
            C ← Q
        end
    end
    return post_process_instance(G,C)
```

O algoritmo MCBB começa com um pré-processamento da instância G, retornando os conjuntos C e S. O conjunto S contém estados a serem processados, e o conjunto C representa a melhor solução encontrada até o momento.

A partir daí, o algoritmo itera sobre os estados em S, onde para cada estado (Q,K), Q é o conjunto de vértices da solução parcial e K é o conjunto de vértices restantes que podem

ser adicionados à solução. Em cada iteração, ele verifica se o tamanho da solução C pode ser melhorado, realizando a remoção de vértices do grafo G e fazendo uma poda baseada em uma estimativa do limite superior (`upper_bound`). O conjunto K é atualizado conforme novos vértices são adicionados à solução parcial Q.

Se a solução parcial Q for maior que C, C é atualizada. Esse processo é repetido até que todos os estados sejam processados ou que a solução ótima seja encontrada. Ao final, o algoritmo retorna a melhor solução C após o pós-processamento da instância de entrada G.

A contribuição desse estudo esta na definição de um algoritmo base testado empiricamente que oferece um ponto de partida sólido. Neste trabalho, a proposta é avançar além desse arcabouço estabelecido, incorporando elementos de aprendizado de máquina para aprimorar a eficiência e a precisão das decisões durante o processo de poda.

3.2 Deep Learning Assisted Heuristic Tree Search for the Container Pre-marshalling Problem

O estudo de Hottung *et al.* (2017) apresenta o algoritmo DLTS, aplicado ao PPMC, um problema de otimização relacionado à disposição de contêineres em um terminal portuário antes de sua transferência para navios de carga. O objetivo é organizar os contêineres com base nas prioridades de embarque, ou seja, contêineres a serem retirados primeiro devem ser posicionados acima dos que serão retirados depois.

O DLTS é uma abordagem de B&B que incorpora redes neurais profundas para aprimorar a tomada de decisão durante a busca. A rede neural é utilizada para refinar a ordem de ramificação e para estimar os limites das soluções em cada ramo da árvore de busca. Após ser treinado com soluções conhecidas, o DLTS encontra soluções heurísticas de alta qualidade em tempo significativamente menor do que abordagens exatas.

Os resultados demonstraram um desempenho superior, com diferenças de otimalidade entre 0% e 2% em instâncias reais, em contraste com meta-heurísticas modernas, que apresentaram variações entre 6% e 15%.

3.2.1 Algoritmo DLTS

O DLTS-DFS é uma estratégia de B&B baseada em DFS, na qual redes neurais profundas são usadas para melhorar a ordenação de ramificação e a poda dos ramos da árvore de

Algoritmo 2: Algoritmo de busca em profundidade assistido por aprendizado profundo - fonte:(Hottung *et al.*, 2017)

Input : A node n of the search tree; bounding network query frequency k; lower bound uncertainty adjustment d; branch pruning adjustment parameter p.
Output : Node representing the best complete solution found (with costs below ub); $\emptyset$ otherwise.
function `DLTS-DFS(n, k, d, p)`:
  if *Complete(n)* ***and*** *Cost(n)* $< ub$ **then**
    ub $\leftarrow$ Cost(n) **return** n;
  end
  hlb $\leftarrow -\infty$;
  if *Depth(n) mod k = 0* **then**
    hlb $\leftarrow$ Cost(n) + DNN-Bounding(n) $\cdot\, d$;
  end
  if *Cost(n)* $\geq ub$ ***or*** *hlb* $\geq ub$ ***or*** *CPU time exceeded* **then**
    return $\emptyset$;
  end
  $r \leftarrow \max_{n_0 \in \text{Successors}(n)} \text{DNN-Branching}(n, n_0)$;
  $B \leftarrow \{n_0 \in \text{Successors}(n) \,|\, \text{DNN-Branching}(n, n_0) \geq \text{MP}(p, r, \text{Cost}(n), ub)\}$;
  Sort B by DNN-Branching(n, n_0) for each $n_0 \in B$, descending;
  return arg $\min_{n_0 \in B} \text{Cost}(\text{DLTS-DFS}(n_0, k, d, p))$;

busca.

Dado um nó n da árvore de busca, o algoritmo avalia a solução parcial associada a n, calculando o custo $\text{Cost}(n)$. O algoritmo calcula um limite inferior $\text{hlb}(n)$ ajustado pela rede neural, considerando a profundidade do nó. Se o custo de n ou o limite inferior excederem o limite superior ub, o nó é podado. Para os sucessores n_0 de n, o algoritmo aplica a função de ramificação $\text{DNN-Branching}(n, n_0)$, selecionando os sucessores com base na função de poda $\text{MP}(p, r, \text{Cost}(n), ub)$. Os sucessores selecionados são ordenados e o algoritmo recursivamente expande o nó com o menor custo até encontrar a solução ótima.

Neste trabalho, será aplicado os métodos utilizados nesse estudo, adaptando-os para otimizar a tomada de decisão sobre ordem de ramificação e poda em algoritmos de B&B, agora direcionados ao PCM.

3.3 Algoritmos para o problema da clique máxima com paralelismo de bits

O estudo de Sousa e Tavares (2024) apresenta uma implementação de algoritmos para o PCM utilizando uma estrutura de dados baseada em paralelismo de bits. Nesse trabalho, os algoritmos propostos foram comparados ao algoritmo MCS, pertencente ao framework unificado (Carmo; Züge, 2012), que é reconhecido por obter soluções de alta qualidade para o PCM.

Os resultados demonstraram a eficiência da estrutura de dados RoaringBitmap utilizada para a implementação dos dois algoritmos apresentados. Ambos alcançaram um

desempenho significativamente superior ao do algoritmo MCS, especialmente em instâncias de grande escala. A Tabela 1 compara os tempos médios de execução (em segundos) e a quantidade média de nós explorados na árvore de busca pelos algoritmos B&B com paralelismo de bits, MCS e Bonecas Russas.

Tabela 1 – Numero de nós na árvore de busca e tempo de execução

Vértices	Densidade	B&B	Bonecas Russas	MCS	B&B Nós	Boneca Russas nós
2000	0.25	54.32	61.33	235.88	2,19E+06	2,45E+06
500	0.5	17.51	19.24	31.52	9,60E+05	9,65E+05
250	0.75	85.91	93.79	111.5	3,48E+06	3,40E+06
150	0.9	18.16	15.58	21.05	5,04E+05	3,77E+05

Fonte: (Sousa; Tavares, 2024)

3.3.1 Branch and Bound com RoaringBitmap

A heurística de coloração relaxada apresentada no Algoritmo 3 recebe um inteiro k, que representa o número máximo de classes de cores a serem construídas. A heurística devolve dois conjuntos de vértices: L, que contém os vértices coloridos nas k classes geradas, e R, que contém os vértices não coloridos. O conjunto R será utilizado para a etapa de ramificação. A ideia principal deste método é reduzir o tempo de coloração, permitindo uma transição mais rápida para a fase de geração de subproblemas.

Algoritmo 3: Algoritmo separa - fonte: (Sousa; Tavares, 2024)

```
Function separa(G, U, k):
    R ← U.copy();
    L ← Bitmap([]);
    l ← 1;
    while |R| > 0 and l ≤ k do
        S ← R.copy();
        while |S| > 0 do
            v ← S.min();
            S ← S \ G.N(v);
            S ← S \ {v};
            R ← R \ {v};
            L ← L ∪ {v};
        end
        l ← l + 1;
    end
    return L, R;
```

Algoritmo 4: Algoritmo BITCLIQUERELAXED - fonte: (Sousa; Tavares, 2024)

```
Function BITCLIQUERELAXED(G, U, S, S_max):
    if U = ∅ then
        if |S| > |S_max| then
            S_max ← S;
        end
    end
    else
        k ← |S_max| − |S|;
        L, R ← separa(G, U, k) ;
        while R ≠ ∅ do
            v ← R.max();
            S ← S ∪ {v};
            U_v ← U ∩ N(v);
            BITCLIQUERELAXED(G, U_v, S, S_max);
            U ← U \ {v};
            S ← S \ {v};
        end
    end
```

O Algoritmo 4 descreve o B&B utilizando a heurística de coloração relaxada para o PCM). Cada subproblema é representado por uma tripla (U, S, S_{max}), onde U é o conjunto de vértices candidatos a serem adicionados à clique atual, S é a clique parcial atual e S_{max} é a maior clique encontrada até o momento. Em cada subproblema (U, S, S_{max}), aplica-se uma heurística de coloração relaxada sobre o conjunto U, resultando em dois subconjuntos: L, contendo os vértices que foram coloridos, e R, contendo os vértices não coloridos, de forma que o número de cores $\omega(G[L])$ seja limitado a k.

Os vértices do conjunto R são escolhidos para a geração de novos subproblemas. O vértice v de maior índice, de acordo com a ordenação inicial dos vértices, é selecionado para ser adicionado à clique atual S. Em seguida, o conjunto candidato é atualizado. O novo subproblema gerado consiste em $(U \cap N(v), S \cup \{v\}, S_{max})$, onde a busca prossegue de forma recursiva. Após resolver esse subproblema, o vértice v é removido do conjunto candidato U e também da clique atual S, permitindo que outros vértices de R sejam considerados.

A principal contribuição desse estudo está na implementação de um algoritmo de B&B, que será utilizado como referência para a comparação com o algoritmo de B&B desenvolvido neste trabalho, o qual integra elementos de aprendizado de máquina.

4 METODOLOGIA

Nesta seção, detalha-se as etapas envolvidas no processo de criação, avaliação, validação e na análise da eficiência dos modelos em reduzir os tempos para calcular a clique máxima.

4.1 Dados de treinamento

O treinamento do modelo de ML exige um conjunto de dados robusto e diversificado. Para isso, foi gerado um total de 50.000 instâncias de grafos para serem usados para o treinamento. A geração de cada grafo foi parametrizada para garantir variedade em sua estrutura. Especificamente, o número de vértices de cada grafo foi definido aleatoriamente em um intervalo de 15 a 30. Além do tamanho, a densidade dos grafos também foi variada. A probabilidade de existência de uma aresta entre quaisquer dois vértices foi distribuída uniformemente entre 0.3 e 0.9. Para a geração, foram utilizadas bibliotecas da linguagem Python, como random para a aleatorização dos parâmetros e networkx para a criação dos grafos.

Os dados gerados passaram por um processo de pré-processamento para padronizar sua representação, garantindo que todos os grafos estejam em um formato uniforme e adequado para o modelo de ML a ser utilizado. Esses dados foram usados no treinamento do modelo para aprender ordens de ramificação e poda de B&B.

O Algoritmo base para o PCM apresentado em (Zuge, 2012) foi utilizado para resolver o PCM para as instâncias geradas. Durante o processo de resolução foi registrado o ramo de nós que conduz da configuração inicial do grafo até a clique máxima. Cada nó no ramo é uma configuração do grafo, representada por três elementos: a matriz de adjacência atual, a clique atual e os rótulos dos vértices que indicam se levam ou não a uma solução ótima.

4.2 Treinamento do Modelo

O conjunto de dados foi dividido em subconjuntos de treino e teste. Essa divisão é fundamental para avaliar o desempenho do modelo em dados que não foram utilizados durante o treinamento, promovendo assim a generalização. Os dados foram divididos, destinando 80% deles para o conjunto de treinamento e reservando os 20% restantes para o conjunto de teste. Uma divisão comumente padrão em trabalhos envolvendo ML.

Um modelo de ML foi treinado usando os dados de treinamento. Esse modelo de

ML foi projetado para cumprir os objetivos da pesquisa, que incluem estimar a probabilidade de um ramo conter uma solução ótima e os limites superiores sobre quanto um ramo ainda pode aumentar a solução.

Durante o treinamento do modelo, a etapa de avaliação mediu o quão eficaz o modelo é no cumprimento de seus objetivos. Diversas métricas de desempenho foram aplicadas para fornecer uma visão abrangente de sua qualidade. Algumas das principais métricas incluem: Acurácia, precisão, recall, erro médio absoluto (MAE), erro médio quadrático (MSE) e coeficiente de determinação (R^2).

Os resultados finais obtidos a partir do modelo treinado foram documentados, incluindo a análise de descobertas e conclusões relevantes. Para documentar esses resultados, foram utilizadas tabelas e gráficos que mostrem métricas de desempenho.

4.3 Teste de Eficiência

Após a conclusão do modelo de ML com uma acurácia satisfatória,realizamos uma série de testes para avaliar a eficácia deste método na redução dos tempos computacionais. O modelo foi empregado em grafos previamente não observados, a fim de estimar ordens de ramificação e limites para poda da árvore de busca.

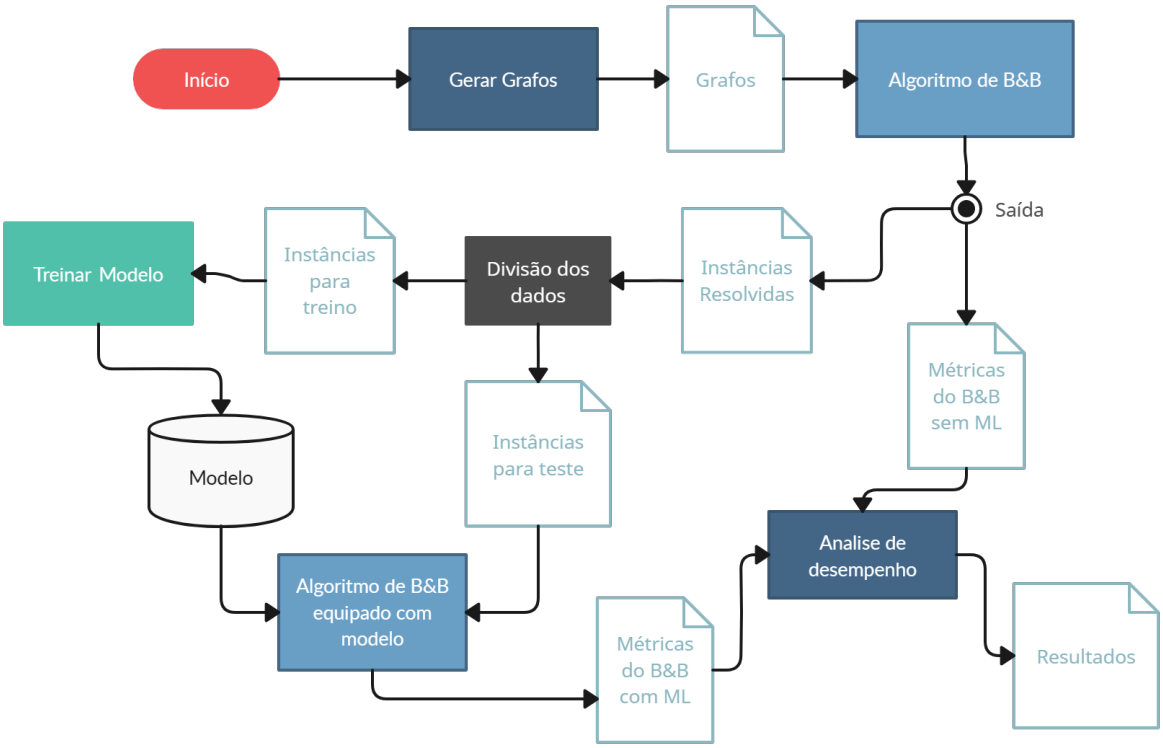
Comparamos o tempo de execução e a precisão do algoritmo de B&B com DLTS, desenvolvido neste trabalho, com o algoritmo de B&B baseado em paralelismo de bits apresentado por (Sousa; Tavares, 2024).

Nos casos em que o algoritmo de B&B com DLTS encontra corretamente a clique máxima, calculamos o ganho de performance relativo entre as duas abordagens, verificando a frequência com que essa relação se mantém: $T_r(g) < T_l(g)$.

Onde $T_r(g)$ representa o tempo necessário para encontrar a clique máxima utilizado DLTS para o grafo g, e $T_l(g)$ denota o tempo necessário para encontrar a clique máxima no grafo g utilizado o algoritmo de B&B com paralismo de bits.

Nos casos em que a clique encontrada pela abordagem que usa DLTS não for a máxima, calculamos: a diferença média entre o tamanho da clique máxima encontrada e o tamanho da clique máxima correta; a porcentagem de vértices corretos; e a diferença de tempo entre as soluções. A Figura 7 é a representação visual do fluxograma descrito nesta seção.

Figura 7 – Fluxograma de etapas da metologia.



Fonte: Autor.

5 DLTS PARA O PROBLEMA DA CLIQUE MÁXIMA

Nesta seção, será apresentada a estrutura da rede neural utilizada neste trabalho para o treinamento do modelo DLTS. O objetivo é detalhar como o modelo foi projetado para aprender heurísticas que orientem a ordem de ramificação e poda no contexto do algoritmo B&B.

Inicialmente, será abordado como os dados de entrada são estruturados para representar os nós da árvore de busca e como as saídas do modelo são interpretadas para guiar as decisões no PCM. Em seguida, será descrita a arquitetura da rede neural, destacando as camadas utilizadas e suas respectivas funções.

A implementação da rede neural, desenvolvida em Python, está documentada e disponível no repositório de código associado a este trabalho (Sousa, 2025), permitindo a reprodução e análise detalhada do modelo.

5.1 Estrutura dos dados

Os dados utilizados para treinar o modelo são estruturados de forma a capturar o caminho percorrido na árvore de busca do algoritmo B&B, desde o nó inicial até alcançar a clique máxima. Cada nó da árvore de busca é representado a matriz de adjacência atual, o vetor que representa a clique atual e os rótulos que variam de acordo com o modelo de *branch* e modelo de *bound*.

No modelo de *branch*, os rótulos indicam os vértices que podem ser adicionados para alcançar a clique máxima, mas que não necessariamente serão todos adicionados. Já no modelo de *bound*, o rótulo representa o número de passos necessários para ir do nó atual até a clique máxima.

A matriz de adjacência é uma matriz quadrada onde cada linha e coluna representam um vértice do grafo. A entrada A_{ij} na matriz de adjacência é definida da seguinte forma:

$$A_{ij} = \begin{cases} 1 & \text{se existe uma aresta entre os vértices } i \text{ e } j, \\ 0 & \text{caso contrário} \end{cases}$$

O vetor de clique, denotado por $\mathbf{c}$, é um vetor binário onde cada posição i indica se o vértice i pertence à clique atual, conforme a definição abaixo:

$$\mathbf{c}_i = \begin{cases} 1 & \text{se o vértice } i \text{ faz parte da clique atual,} \\ 0 & \text{caso contrário} \end{cases}$$

O vetor de rótulos, denotado por $\mathbf{y}$, é um vetor binário onde cada posição i indica se

o vértice i leva a uma solução ótima, conforme a definição abaixo:

$$\mathbf{y}_i = \begin{cases} 1 & \text{se o vértice } i \text{ faz parte da clique máxima,} \\ 0 & \text{caso contrário} \end{cases}$$

Inicialmente, o nó raiz contém a matriz de adjacência completa do grafo, o vetor de clique inicializado com zeros (indicando que nenhum vértice foi adicionado à clique) e um vetor de rótulos. O vetor de rótulos é binário, onde cada posição indica se adicionar o vértice correspondente à clique atual leva à clique máxima (1) ou não (0). A Figura 8 representa o estado inicial para do grafo da Figura 1.

Figura 8 – Representação do nó inicial do da árvore de busca do grafo da figura 1. A matriz A é a matriz de adjacência, o vetor $\mathbf{c}$ indica que nenhum vértice pertence à clique inicial, e o vetor $\mathbf{y}$ mostra os vértices que podem ser adicionados para formar a clique máxima.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Fonte: Autor.

Conforme um vértice é adicionado à clique, os dados para o próximo nó da árvore são ajustados. A matriz de adjacência é reduzida para conter apenas os vértices que são vizinhos do vértice adicionado, excluindo o próprio vértice. O vetor de clique é atualizado para refletir a inclusão do vértice adicionado, colocando 1 na posição correspondente. Esse processo se repete até que chegar ao nó que contém a clique máxima.

A Figura 9 representa o nó seguinte no caminho da árvore de busca do inicial até a clique máxima, após adicionar o vértice 0, para o grafo da figura 1. Esse método permite que o modelo aprenda as transições entre estados na árvore de busca e identifique quais vértices devem ser escolhidos para alcançar a solução ótima.

5.2 Camada de entrada

Primeiramente, é importante definir o conceito de tensor. Um tensor é uma entidade matemática que pode ser representada por uma estrutura com um número específico de índices,

Figura 9 – Representação do nó seguinte da árvore de busca da figura 8, ao adicionar o vértice 0 à clique. A matriz A é a matriz de adjacência, preenchida com 0s para o próprio vértice adicionado e para os vértices que não são seus vizinhos. O vetor **c** contém 1 na posição do vértice 0, que foi adicionado à clique. O vetor **y** mostra os vértices que podem ser adicionados para formar a clique máxima.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Fonte: Autor.

os quais determinam sua ordem ou dimensionalidade. Além disso, os tensores obedecem a leis de transformação específicas quando as coordenadas do sistema de referência são alteradas (Grinfeld *et al.*, 2013). De forma geral, os tensores englobam e generalizam conceitos como escalares, vetores e matrizes, estendendo-os para quantidades de ordem superior.

A camada de entrada do modelo recebe uma lista $x+1$ tensores, onde x é o número de vértices no grafo. Os primeiros x tensores correspondem às linhas da matriz de adjacência reduzida no nó atual, cada um com dimensão x. Essa abordagem separa a matriz de adjacência em vetores individuais, facilitando o processamento paralelo de cada linha. O último tensor da lista é um vetor binário de dimensão x, que representa a clique atual, indicando quais vértices já foram adicionados à clique.

5.3 Camadas densas compartilhadas

As camadas densas compartilhadas processam cada vetor de entrada separadamente. Essas camadas são projetadas para capturar padrões locais específicos de cada entrada, com o objetivo de preservar a independência inicial dos vetores antes da concatenação.

Cada camada densa compartilhada utiliza a função de ativação ReLU para capturar relações não lineares nos dados. A dimensão de saída de cada camada densa compartilhada é escalonada em função de uma lista de parâmetros $p1$, que define a quantidade de neurônios na camada densa i como $x * p1_i$. O número de camadas densas compartilhadas vai ser dado pelo tamanho da lista $p1$.

5.4 Camadas ocultas

Após o processamento nas camadas densas compartilhadas, os vetores de saída são concatenados em um único vetor unificado. Essa etapa combina as informações extraídas de cada entrada individual, permitindo que as camadas subsequentes processem o estado completo do nó da árvore de busca.

As camadas ocultas subsequentes processam o vetor concatenado para aprender representações mais complexas que conectem os dados de entrada às decisões de saída.

Essas camadas utilizam a função de ativação ReLU e possuem dimensões definidas pelo parâmetro $p2$, que controla o número de neurônios em cada camada. A dimensão de cada camada i é ajustada em função de $x * (x+1) * p2_i$, considerando que há entrada incialmente consiste $x+1$ tensores de dimensão x. O número de camadas ocultas é determinado pelo tamanho da lista $p2$.

5.5 Camada de saída

A camada de saída desempenha um papel crucial em ambos os modelos, pois define o formato final das previsões e traduz as informações extraídas pelas camadas ocultas em resultados específicos para cada tarefa. No modelo de *branch*, a saída consiste em uma distribuição de probabilidades sobre os vértices, indicando aqueles que devem ser priorizados para a expansão da clique. Já no modelo de *bound*, a saída é um valor escalar contínuo, que estima o número de passos necessários para alcançar a clique máxima.

5.5.1 Camada de saída para a rede de branch

A camada de saída para o modelo de *branch* é uma camada densa que utiliza a função de ativação *softmax*. Sua função é produzir probabilidades para cada vértice, indicando a probabilidade de que ele deve ser adicionado à clique atual para alcançar a clique máxima.

A dimensão da saída é x, correspondente ao número total de vértices no grafo. A camada considera todos os vértices do grafo, mesmo aqueles que não podem ser adicionados. A remoção dos vértices inválidos para ramificação no nó atual é realizada em uma etapa de pós-processamento.

A função *softmax* normaliza os valores de saída para formar uma distribuição de

probabilidades:

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{x} \exp(z_j)}$$

Onde z_i é o valor bruto (*logit*) produzido para o vértice i.

O modelo é compilado com o otimizador Adam, utilizando a função de perda *categorical crossentropy*, ideal para tarefas de classificação multiclasse, e a métrica de avaliação baseada na acurácia.

5.5.2 Camada de saída para a rede de bound

A camada de saída do modelo de *bound* é uma camada densa que não utiliza função de ativação, produzindo um único valor contínuo como saída.

A dimensão da saída é 1, representando a estimativa do número de passos necessários para alcançar a clique máxima a partir do nó atual na árvore de busca. Esse valor contínuo permite ao modelo realizar uma previsão regressiva diretamente, sem restrições de classificação ou normalização, uma vez que o problema de *bound* exige prever valores escalares.

Além disso, o modelo é compilado com o otimizador Adam. A perda utilizada é o erro quadrático médio, adequado para problemas de regressão, e a métrica de avaliação é o erro absoluto médio.

6 RESULTADOS

Nesta seção, serão apresentados os resultados obtidos com a aplicação dos modelos de aprendizado de máquina desenvolvidos para o PCM. A principal barreira encontrada para a escalabilidade da solução foi o consumo de memória, que cresce de forma acentuadamente não linear com o número de vértices. A exemplo enquanto os testes no ambiente principal, com 6 GB de VRAM, foram possíveis para grafos de até 30 vértices, um teste complementar na plataforma Google Colab verificou que instâncias com 100 vértices já demandavam mais de 100 GB de VRAM. Diante da impossibilidade de escalar os testes no hardware local, a análise apresentada se concentrou em instâncias menores de até 30 vétices. Os testes principais foram realizados em um notebook com processador Intel Core i7-9750H, 32 GB de memória RAM, GPU Nvidia RTX 2060 com 6 GB de VRAM e sistema operacional Windows Subsystem for Linux 2 (WSL 2).

Inicialmente, nesta seção, serão detalhadas as métricas de desempenho do modelo de *branch*. Em seguida, serão analisadas as métricas do modelo de *bound*. Por fim, será realizada uma comparação entre o desempenho do algoritmo baseado no DLTS e o algoritmo de B&B com paralelismo de bits apresentado no trabalho (Sousa; Tavares, 2024).

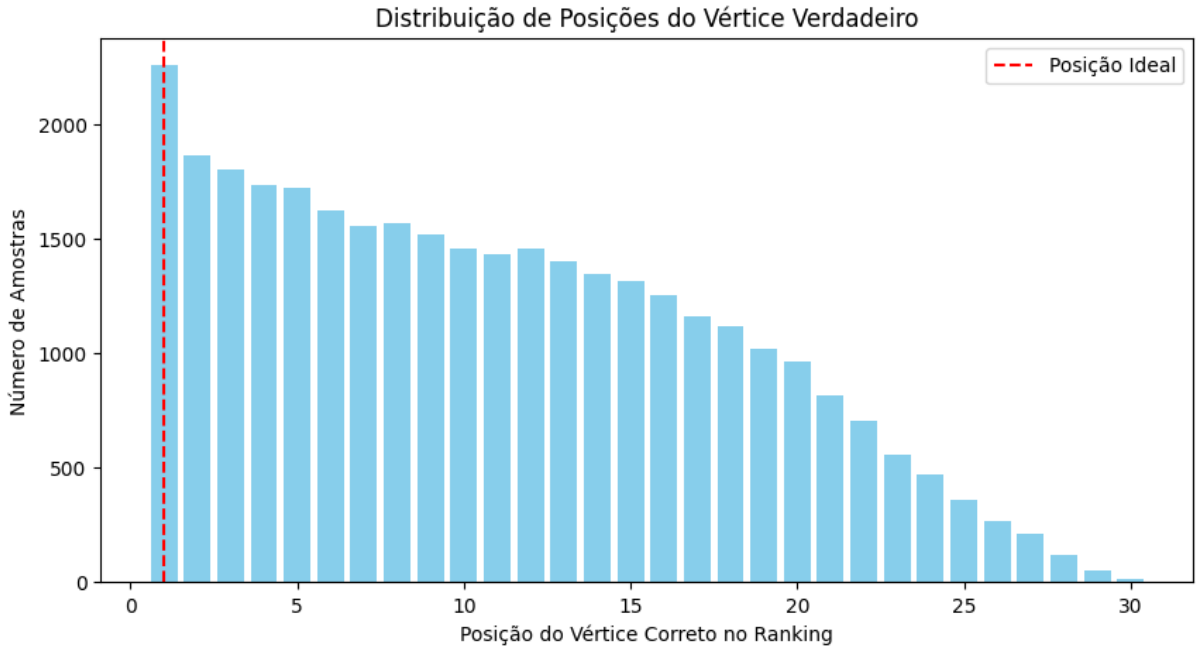
6.1 Métricas do modelo de branch

Para avaliar o desempenho do modelo de *branch*, foram utilizadas métricas que capturam a capacidade da rede DLTS de priorizar corretamente o vértice ideal no ranking de candidatos em cada estado da árvore de busca. As principais métricas computadas foram:

- **Accuracy@k** (2.1): percentual de amostras em que o melhor vértice para ramificar aparece entre os k primeiros do ranking predito. Valores reportados:
  - Accuracy@1: 6,82%
  - Accuracy@2: 12,44%
  - Accuracy@3: 17,88%
- **MRR (Mean Reciprocal Rank)** (2.2): média do inverso da posição do vértice correto no ranking. O valor obtido foi de 0,1986, o que indica que, em média, o vértice ideal aparece entre as primeiras posições.
- **Rank Relativo Médio** (2.4): normaliza a posição do vértice correto no ranking considerando o total de vértices disponíveis. O valor médio foi de 0,6593, sendo mais próximo de 1 quanto melhor for a posição relativa do vértice ideal.

A Figura 10 ilustra a distribuição das posições reais do vértice verdadeiro no ranking gerado pelo modelo. Observa-se que a maioria dos vértices corretos aparece nas primeiras posições, com um decréscimo gradual, o que confirma a efetividade da rede em priorizar bons candidatos. O traço vertical em vermelho indica a posição ideal (primeiro lugar no ranking).

Figura 10 – Distribuição de posições do vértice verdadeiro no ranking predito pelo modelo de *branch*.



Fonte: Autor.

Embora a Accuracy@1 de 6,82% revele que o vértice ótimo isoladamente nem sempre ocupa a primeira posição no ranking, a progressão para Accuracy@2 (12,44%) e Accuracy@3 (17,88%) demonstra uma capacidade significativa do modelo em posicionar o vértice correto entre as principais escolhas. Mais importante ainda, o MRR de 0,1986, que corresponde a uma posição média recíproca de aproximadamente 5 ($1/0{,}1986 \approx 5{,}03$), sugere que, na média, o vértice ideal é encontrado relativamente cedo na lista de candidatos. Este achado é crucial, pois mesmo não acertando sempre o primeiro, ao consistentemente colocar o vértice ideal entre os primeiros, o modelo efetivamente direciona o algoritmo B&B para regiões mais promissoras da árvore de busca.

Esses resultados indicam que o modelo de *branch* aprendeu padrões relevantes para ranquear os vértices candidatos, contribuindo para uma busca mais eficiente na árvore de decisão do algoritmo B&B.

6.2 Métricas do modelo de bound

O modelo de *bound* foi avaliado com métricas de regressão clássicas, visando medir a capacidade da rede neural em prever o número de passos restantes até a solução ótima do problema. As métricas utilizadas foram:

- **MAE (Mean Absolute Error)** (2.5): erro absoluto médio entre valores reais e preditos. Valor obtido: 0,9292.
- **MSE (Mean Squared Error)** (2.6): média dos erros quadráticos. Valor obtido: 1,5241.
- **RMSE (Root Mean Squared Error)** (2.7): raiz quadrada do erro quadrático médio. Valor obtido: 1,2346.
- **R^2 (Coeficiente de Determinação)** (2.8): mede a proporção da variância dos dados explicada pelo modelo. O valor obtido foi 1, indicando que o modelo explica quase perfeitamente os dados de teste.

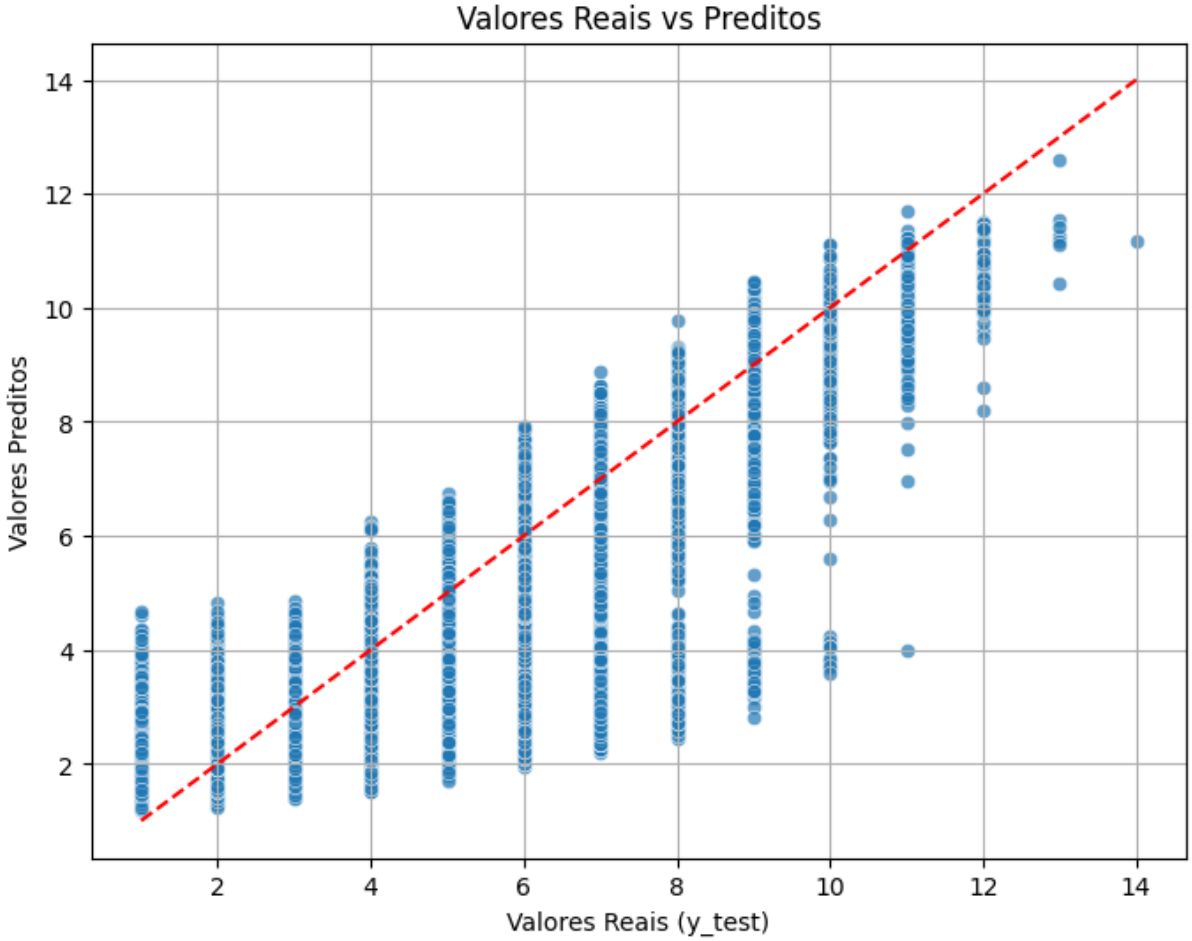
A Figura 11 apresenta o gráfico de dispersão entre os valores reais e os valores preditos pela rede. A linha tracejada em vermelho representa a reta ideal, onde todas as predições coincidiriam exatamente com os valores reais.

Observa-se que os pontos concentram-se fortemente ao redor da linha ideal, o que reforça o valor elevado do R^2 e demonstra a eficácia do modelo em prever corretamente o número de passos restantes para alcançar a clique máxima.

As métricas de erro, como o MAE de 0,9292, MSE de 1,5241 e o RMSE de 1,2346, são particularmente encorajadoras. Um MAE próximo de 1 indica que, em média, a previsão do modelo para o número de passos restantes desvia-se em menos de um passo do valor real. Essa precisão é vital para a função de *bound* em um algoritmo B&B.

O Coeficiente de Determinação (R^2) de 1 é um resultado notável, sugerindo que, para o conjunto de dados de teste e dentro da limitação de 30 vértices, o modelo foi capaz de explicar praticamente toda a variabilidade no número de passos restantes. Esta predição acurada do "custo para chegar à solução" permite ao algoritmo B&B realizar podas de forma muito mais confiante. Se o número de passos predito, somado ao tamanho da clique atual, não supera o melhor limitante inferior conhecido, o ramo pode ser descartado precocemente, economizando recursos computacionais significativos.

Figura 11 – Dispersão entre valores reais e preditos pelo modelo de *bound*.



Fonte: Autor.

6.3 MCBB_DLTS

O algoritmo `MCBB_DLTS` (Max-Clique Branch and Bound with Deep Learning for Tree Search) é detalhado no Algoritmo 5. Ele emprega uma estratégia de ramificação e poda (Branch and Bound) onde modelos de aprendizado profundo são utilizados para guiar a seleção de vértices para ramificação e para estimar limites superiores que permitem a poda de ramos não promissores da árvore de busca.

No que tange à estratégia de *bounding*, o `MCBB_DLTS` evita invocar a respectiva rede neural (`dnn_bounding`) a cada nó da árvore de busca. A consulta à DNN para obter uma estimativa mais precisa do limite superior ocorre apenas quando o tamanho da clique parcial atual atinge um múltiplo do parâmetro k_{bound}. Fora dessas consultas periódicas, o algoritmo recorre a um limite mais simples e rápido, como o tamanho do conjunto de candidatos $|K|$. Essa tática de uso intermitente da DNN de *bounding* é crucial para mitigar o alto custo computacional

Algoritmo 5: Algoritmo MCBB_DLTS

```
Função MCBB_DLTS(G, k_branch, k_bound):
    (C, S) ← preprocess(G);
    branch_sc ← nulo;
    while S ≠ ∅ do
        (Q, K) ← pop(S);
        iter_refr ← 0;
        branch_sc ← nulo;
        while K ≠ ∅ do
            est_ext ← |K|;
            if |Q| (mod k_bound) = 0 then
                est_ext ← dnn_bounding(Q, K, G)
            end
            if |C| ≥ |Q| + est_ext then
                break;
            end
            if iter_refr (mod k_branch) = 0 then
                branch_sc ← get_dnn_scores(Q, K, G);
            end
            iter_refr ← iter_refr + 1;
            v ← nulo;
            if branch_sc ≠ nulo then
                cand_sc_K ← {i ↦ branch_sc[i] | i ∈ K};
                if cand_sc_K ≠ ∅ then
                    v ← argmax_{i∈cand_sc_K}(cand_sc_K[i]);
                end
            end
            if v = nulo then
                break;
            end
            K_no_v ← K.remove(v);
            if K_no_v ≠ ∅ then
                if should_expand(Q, K_no_v, C, G) then
                    push(S, (list(Q), K_no_v));
                end
            end
            Q.append(v);
            K ← K_no_v ∩ neighbors(G, v);
        end
        if |C| < |Q| then
            C ← list(Q);
        end
    end
    return C;
```

da rede de *bounding*.

De maneira análoga, a operação de *branching* é otimizada. Uma rede neural é empregada para gerar pontuações que indicam a relevância de cada vértice candidato para a formação da clique máxima. Contudo, visando a eficiência, esses scores não são recalculados a cada decisão de qual vértice adicionar à clique. Em vez disso, os scores são atualizados apenas periodicamente: a cada k_{branch} seleções de vértice dentro do mesmo caminho de exploração ou quando um novo caminho é iniciado a partir da pilha principal (conforme detalhado no Algoritmo 5). Entre essas atualizações, o algoritmo reutiliza os scores da última consulta à rede, aplicando-os aos candidatos ainda disponíveis. Essa reutilização permite que a orientação da DNN influencie múltiplas decisões de ramificação com o custo de uma única inferência, balanceando a qualidade da orientação com a eficiência.

A adoção desta estratégia de consulta periódica às redes neurais, controlada pelos parâmetros k_{bound} e k_{branch}, foi uma decisão de projeto fundamentada em análises de desempenho, realizadas através de *profiling* de código. Em uma versão preliminar que consultava as redes neurais a cada oportunidade, observou-se que as chamadas de inferência eram um gargalo crítico, consumindo em média 97% do tempo total de execução. Este resultado evidenciou que a frequência das consultas aos modelos de aprendizado profundo era o principal fator limitante para a eficiência do processo.

Diante dessa constatação, foi implementada uma estratégia de consulta periódica, controlada pelos parâmetros k_{bound} e k_{branch}. A adoção deste mecanismo resultou em uma redução expressiva de aproximadamente 29% no tempo total de execução. É notável que, embora a contribuição percentual das consultas para o novo tempo total tenha permanecido alta (cerca de 97%), a estratégia se provou eficaz ao diminuir o número absoluto de chamadas de inferência, melhorando significativamente a performance geral do algoritmo.

6.4 Comparação entre algoritmos

Nesta seção, é realizada uma comparação de desempenho entre o algoritmo proposto, `MCBB_DLTS` (baseado em DLTS), e o algoritmo `BitClique`, uma implementação de *Branch and Bound* com paralelismo de bits conforme apresentado por Albuquerque et al. (Sousa; Tavares, 2024). A comparação foca no tempo médio de execução, no número médio de nós percorridos durante a exploração da árvore de busca e na diferença percentual de tamanho entre a clique máxima encontrada pelo `BitClique` (considerado exato) e a clique encontrada pelo

MCBB_DLTS. Os testes foram conduzidos para grafos com diferentes números de nós e densidades. Os resultados estão compilados na Tabela 2.

Na Tabela 2 as colunas 'Nós' e 'Densidade' descrevem as características dos grafos utilizados nos testes. As colunas 'Tempo DLTS (s)' e 'Tempo BitClique (s)' apresentam o tempo médio de execução em segundos para cada algoritmo. A eficiência da busca é comparada através das colunas 'Nós DLTS' e 'Nós BitClique', que indicam o número de nós da árvore de busca explorados. Finalmente, a coluna 'Tam Dif. (%)' mede a diferença percentual de tamanho da clique encontrada pelo MCBB_DLTS em comparação com a solução exata do BitClique, servindo como uma medida de acurácia da heurística.

Tabela 2 – Comparativo de desempenho entre o algoritmo MCBB_DLTS e BitClique.

Nós	Densidade	Tempo DLTS (s)	Tempo BitClique (s)	Nós DLTS	Nós BitClique	Tam Dif. (%)
10	0.25	7.18	2.48×10^{-4}	13	13	0
10	0.50	7.77	2.82×10^{-4}	17	17	0
10	0.75	5.83	3.37×10^{-4}	13	25	2
10	0.90	5.25	3.72×10^{-4}	11	35	3
15	0.25	1.10×10^{1}	2.91×10^{-4}	25	19	0
15	0.50	9.05	4.18×10^{-4}	20	29	3
15	0.75	1.02×10^{1}	6.03×10^{-4}	19	44	3
15	0.90	7.38	7.55×10^{-4}	12	66	3
20	0.25	1.70×10^{1}	3.91×10^{-4}	39	26	0
20	0.50	1.68×10^{1}	7.07×10^{-4}	36	34	0
20	0.75	9.06	1.31×10^{-3}	16	65	5
20	0.90	9.83	1.42×10^{-3}	16	96	6
30	0.25	3.92×10^{1}	1.10×10^{-3}	94	40	0
30	0.50	2.62×10^{1}	1.60×10^{-3}	50	57	8
30	0.75	1.21×10^{1}	2.39×10^{-3}	21	104	10
30	0.90	1.61×10^{1}	2.74×10^{-3}	25	174	4

Fonte: Autor.

Analisando os resultados apresentados na Tabela 2, algumas observações podem ser feitas:

- **Tempo de Execução**: O algoritmo BitClique demonstrou ser consistentemente e significativamente mais rápido que o algoritmo MCBB_DLTS em todas as configurações testadas. Em média, o tempo de execução do MCBB_DLTS foi de aproximadamente 1.31×10^{1} segundos, enquanto o BitClique levou em média 9.35×10^{-4} segundos. Essa diferença representa várias ordens de magnitude, destacando a superior eficiência computacional do BitClique.
- **Nós Explorados (Eficiência da Busca)**: As colunas "Nós DLTS"e "Nós BitClique"indicam

o número médio de nós da árvore de busca explorados por cada algoritmo para cada tipo de instância (média sobre 10 instâncias).

  - Em média geral, considerando os 16 tipos de instância, o `MCBB_DLTS` explorou aproximadamente 26.7 nós, enquanto o `BitClique` explorou cerca de 52.8 nós. Isso sugere que, embora mais lento em tempo de execução total, o `MCBB_DLTS` pode estar realizando uma poda mais agressiva ou seguindo um caminho de busca mais direto.
  - Em alguns tipos de instância, o `BitClique` explorou, em média, menos nós que o `MCBB_DLTS`. No entanto, para outros tipos, especialmente em cenários de maior densidade, o `BitClique` tende a explorar um número médio de nós significativamente maior.

- **Qualidade da Solução (Tam Dif. (%))**: Esta coluna na Tabela 2 representa a diferença percentual média (sobre 10 instâncias para cada tipo de grafo) entre o tamanho da clique encontrada pelo `BitClique` (considerado exato) e a clique encontrada pelo `MCBB_DLTS`. A diferença é calculada como $((\text{tam_exato} - \text{tam_dlts})/\text{tam_exato}) \times 100\%$. Um valor de 0% na tabela indica que encontrou uma clique de tamanho ótimo.
  - Para 6 dos 16 tipos de grafos analisados (representando 60 instâncias individuais), a Tabela 2 mostra uma "Tam Dif. (%)"de 0%. O `MCBB_DLTS` encontrou a solução ótima em todas as 10 instâncias para cada um desses 6 tipos, isso corresponde a aproximadamente 37.5% dos tipos de problema onde o `MCBB_DLTS` demonstrou ser consistentemente ótimo.
  - Para os demais 10 tipos de grafos, a clique encontrada pelo `MCBB_DLTS` foi, em média, menor que a ótima, com a diferença percentual média (conforme a tabela) variando de 2% a 10% para esses tipos.
  - A maior diferença percentual média observada foi de 10% para o tipo de grafo com 30 nós e densidade 0.75, causado por um outlier.
  - A média das diferenças percentuais médias listadas na tabela é de aproximadamente 2.94%. Isso indica que, embora não seja sempre exato em todas as instâncias, o `MCBB_DLTS` geralmente encontra soluções muito próximas da ótima no geral, considerando a média de desempenho para cada tipo de problema.

Em resumo, o algoritmo `BitClique` destaca-se pela sua excepcional velocidade de execução, fornecendo a solução exata em tempo significativamente menor. Por outro lado, o `MCBB_DLTS`, apesar de mais lento, demonstrou explorar, em média, um número menor de nós

na árvore de busca, o que pode indicar uma heurística de busca ou poda potencialmente mais direcionada em certos aspectos. Em termos de qualidade da solução, o `MCBB_DLTS` obteve a clique ótima em uma parcela considerável dos casos e, nos demais, a diferença para a solução exata foi geralmente pequena, com uma média de desvio inferior a 3%.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho se dedicou ao desenvolvimento e avaliação de modelos de aprendizado de máquina aplicados à resolução do problema da clique máxima, culminando na proposição do algoritmo `MCBB_DLTS` (Max-Clique Branch and Bound with Deep Learning for Tree Search). Foram investigados modelos de *branch*, para guiar a escolha de vértices na árvore de busca, e de *bound*, para estimar o tamanho da solução e permitir podas eficientes.

Os resultados individuais dos modelos de aprendizado profundo foram promissores. O modelo de *branch* demonstrou capacidade de priorizar vértices relevantes, com um MRR de 0,1986, indicando que, em média, o vértice ideal se encontra entre as primeiras cinco posições do ranking predito. Embora a Accuracy@1 tenha sido modesta (6,82%), a progressão para Accuracy@2 (12,44%) e Accuracy@3 (17,88%) sugere que o modelo efetivamente direciona a busca para regiões promissoras. O modelo de *bound* exibiu alta precisão, com um R^2 de 1 e um MAE de 0,9292 para instâncias de até 30 vértices, o que significa que, em média, a previsão do número de passos restantes até a solução desvia-se em menos de um passo do valor real, sendo crucial para a eficácia da poda no algoritmo B&B. Foi identificado que as inferências das redes neurais representavam um gargalo de desempenho, levando à estratégia de chamadas periódicas controladas pelos parâmetros k_{branch} e k_{bound}, que se mostrou fundamental para a viabilidade do `MCBB_DLTS`.

Na comparação com o algoritmo `BitClique`, uma abordagem exata baseada em paralelismo de bits, o `MCBB_DLTS` apresentou um tempo de execução consideravelmente maior. O `BitClique` foi, em média, ordens de magnitude mais rápido. Contudo, o `MCBB_DLTS` explorou, em média, um número menor de nós na árvore de busca (26,7 nós contra 52,8 nós do `BitClique`), sugerindo que as heurísticas aprendidas pelos modelos de ML podem, de fato, guiar a busca de forma mais direcionada ou realizar podas mais agressivas. Em termos de qualidade da solução, para os grafos testados (até 30 vértices), o `MCBB_DLTS` encontrou a solução ótima em aproximadamente 37,5% dos tipos de instâncias e, nos demais casos, a diferença percentual média para a clique ótima foi de cerca de 2,94%, indicando uma boa aproximação da solução exata.

As limitações deste estudo incluem o foco em grafos relativamente pequenos (até 30 vértices), devido a restrições de memória e capacidade de processamento para o treinamento e execução dos modelos de aprendizado profundo com a representação de estado via matriz de adjacência completa.

Para trabalhos futuros, diversas avenidas de pesquisa se apresentam. Primeiramente, a exploração de arquiteturas de redes neurais mais eficientes para processamento de grafos, como as Redes Neurais de Grafos (GCNs), poderia resultar em modelos com menor custo de inferência e melhor capacidade de aprendizado das topologias dos grafos. Em segundo lugar, investigar representações de estado mais compactas e eficientes em termos de memória seria crucial para escalar a abordagem para grafos maiores, a exemplo a exploração da simetria inerente à matriz de adjacência em grafos não direcionados, uma técnica que pode reduzir significativamente o consumo de memória. Seria interessante também analisar como o tempo de inferência das redes neurais escala com o aumento do tamanho do grafo; se o crescimento não for exponencial, a abordagem baseada em DLTS poderia se tornar competitiva ou até superior ao `BitClique` para instâncias muito grandes, onde os algoritmos exatos podem enfrentar explosão combinatória. Adicionalmente, uma estratégia promissora é a aplicação de abordagens híbridas que integram ML e algoritmos exatos. Nesse modelo, o processo de ramificação, característico de um algoritmo de coloração exato, seria enriquecido por uma poda da árvore de busca direcionada por ML. Por fim, a metodologia DLTS aqui explorada para o problema da clique máxima possui potencial para ser adaptada e aplicada a outros problemas de otimização combinatória NP-difíceis.

Em suma, este trabalho demonstrou o potencial da integração de aprendizado profundo com algoritmos de busca clássicos para o problema da clique máxima, oferecendo uma nova perspectiva e abrindo caminhos para futuras otimizações e aplicações.

REFERÊNCIAS

BISHOP, C. M. **Pattern Recognition and Machine Learning**. 1st ed. 2006. corr. 2nd printing. ed. Nova York, EUA: Springer, 2006. (Information science and statistics). ISBN 9780387310732; 0387310738.

BOMZE, I. M.; BUDINICH, M.; PARDALOS, P. M.; PELILLO, M. The maximum clique problem. In: DU, D.-Z.; PARDALOS, P. M. (Ed.). **Handbook of Combinatorial Optimization:** Supplement Volume A. New York: Springer, 1999. p. 1–74.

CARMO, R.; ZÜGE, A. Branch and bound algorithms for the maximum clique problem under a unified framework. **Journal of the Brazilian Computer Society**, Springer, v. 18, p. 137–151, 2012.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos:** Teoria e Prática. 3. ed. Rio de Janeiro: Elsevier, 2012. Tradução da 3ª edição de: Introduction to algorithms.

FEOFILOFF, P.; KOHAYAKAWA, Y.; WAKABAYASHI, Y. **Uma Introdução Sucinta à Teoria dos Grafos**. [*S. l.*]: IME-USP, 2011.

FLECK, L.; TAVARES, M. H. F.; EYNG, E.; HELMANN, A. C.; ANDRADE, M. A. d. M. Redes neurais artificiais: Princípios básicos. **Revista Eletrônica Científica Inovação e Tecnologia**, v. 1, n. 13, p. 47–57, 2016.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. MIT Press, 2016. Disponível em: https://www.deeplearningbook.org/. Acesso em: 14 dez. 2024.

GRINFELD, P. *et al.* **Introduction to tensor analysis and the calculus of moving surfaces**. [*S. l.*]: Springer, 2013.

HOTTUNG, A.; TANAKA, S.; TIERNEY, K. Deep learning assisted heuristic tree search for the container pre-marshalling problem. **arXiv preprint arXiv:1709.09972**, 2017.

IZMAILOV, A.; SOLODOV, M. **Otimização Vol. 1 - Condições de otimalidade, elementos de análise convexa e de dualidade**. Rio de Janeiro: IMPA, 2020. ISBN 978-65990528-0-4.

NILSSON, N. J. **Introduction to Machine Learning**. Palo Alto, EUA: Stanford University, 1996. Disponível em: https://ai.stanford.edu/~nilsson/MLBOOK.pdf. Acesso em: 14 dez. 2024.

REBA, K.; GUID, M.; ROZMAN, K.; JANEŽIČ, D.; KONC, J. Exact maximum clique algorithm for different graph types using machine learning. **Mathematics**, MDPI, v. 10, n. 1, p. 97, 2021.

SOUSA, I. A. A. **TCC - DLTS para clique máxima**. GitHub, 2025. Disponível em: https://github.com/IgorAlanAlbuquerque/TCC---DLTS-para-clique-maxima. Acesso em: 4 jan. 2025.

SOUSA, I. A. A.; TAVARES, W. A. Algoritmos para o problema da clique máxima com paralelismo de bits. In: SIMPóSIO BRASILEIRO DE PESQUISA OPERACIONAL, 56., Fortaleza. **Anais [...]**. SBPO, 2024. Disponível em: https://proceedings.science/sbpo/sbpo-2024/trabalhos/algoritmos-para-o-problema-da-clique-maxima-com-paralelismo-de-bits?lang=pt-br. Acesso em: 30 dez. 2024.

SUN, Y.; LI, X.; ERNST, A. Using statistical measures and machine learning for graph reduction to solve maximum weight clique problems. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 43, n. 5, p. 1746–1760, 2021.

TOMITA, E.; SEKI, T. An efficient branch-and-bound algorithm for finding a maximum clique. In: **Proceedings of the 4th** INTERNATIONAL CONFERENCE ON DISCRETE MATHEMATICS AND THEORETICAL COMPUTER SCIENCE. Dijon, França: Springer, 2003. p. 278–289.

WEST, D. B. **Introduction to Graph Theory**. 2nd. ed. Nova York, EUA: Pearson, 2000.

ZUGE, A. P. **Solução exata do problema da clique máxima**. Dissertação (Mestrado em Informática) – Setor de Ciências Exatas, Universidade Federal do Paraná, Curitiba, 2012.