UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Sarah Rebecca Dias Luiz

Problema do Empacotamento Aberto em Classes de Grafos

Belo Horizonte
2023

Sarah Rebecca Dias Luiz

Problema do Empacotamento Aberto em Classes de Grafos

Versão Final

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Vinícius Fernandes dos Santos

Belo Horizonte
2023

Luiz, Sarah Rebecca Dias.

L953p Problema do empacotamento aberto em classes de grafos [recurso eletrônico] / Sarah Rebecca Dias Luiz – 2023.
1 recurso online (52 f. il., color.) : pdf.

Orientador: Vinícius Fernandes dos Santos.

Dissertação (Mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f. 51-52

1. Computação – Teses. 2. Teoria dos grafos – Teses. 3. Complexidade Computacional – Teses. 4. Empacotamento e cobertura combinatória – Teses. I. Santos, Vinícius Fernandes dos Santos. I. Universidade Federal de Minas Gerais Exatas, Departamento de Ciência da Computação. III. Título.

CDU 519.6*52(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais - ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Problema do Empacotamento Aberto em Classes de Grafos

SARAH REBECCA DIAS LUIZ

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores(a):

Vinícius Fernandes dos Santos

PROF. VINÍCIUS FERNANDES DOS SANTOS - Orientador
Departamento de Ciência da Computação - UFMG

PROF. GUILHERME DE CASTRO MENDES GOMES
Departamento de Ciência da Computação - UFMG

PROFA. ANA KAROLINNA MAIA DE OLIVEIRA
Departamento de Computação - Universidade Federal do Ceará

Belo Horizonte, 17 de outubro de 2023.

Dedico esse trabalho a minha mãe.

Agradecimentos

Agradeço inicialmente aos meus pais, que sempre me apoiaram e me impeliram a estudar, para que eu fosse o mais longe que pudesse. Muito obrigada, mãe. A senhora sempre disse que pais são como arcos e filhos como flechas. Sei que a senhora me lançou o mais distante que pôde. Muito obrigada, pai, por ter se aproximado tanto depois que ela não pôde mais estar aqui. Gostaria de agradecer também à minha irmã Esther por sempre estar ao meu lado em todos os momentos e me incentivar a continuar quando eu achei que não ia mais conseguir. Agradeço também à minha tia Alessandra, por todas as conversas de madrugada quando eu não conseguia dormir por não saber que rumo tomar na minha vida. Deixo meu muito obrigada também ao meu namorado (ou noivo, como preferir), Adriano, por acreditar em mim quando nem eu mesma acreditei e por também estar ao meu lado nos bons e nos maus momentos.

Agradeço também imensamente ao meu orientador. Se não fosse pelo Vinícius eu não teria descoberto minha paixão por grafos, não teria pesquisado dentro dessa área e teria perdido a data para entrar no mestrado. Agradeço também pela paciência e pelo carinho ao longo de todo o percurso e por não ter desistido de mim, mesmo quando eu desapareci por um tempo.

Gostaria de poder agradecer nominalmente a todos os meus amigos que me ouviram desabafar e que me impulsionaram a não desistir, mas sinto que a lista ficaria gigantesca e mesmo assim eu acaberia esquecendo algum nome importante, então você, amigo que talvez leia essa mensagem, saiba que sou imensamente grata, mas não posso deixar de agradecer a alguns nomes especiais: obrigada, Diego! Você chegou um semestre antes e guiou o meu caminho com muito carinho. Obrigada também Eder e Pedro Henrique por lerem essa dissertação e me ajudarem com as revisões. Por último, gostaria de agradecer a Lorenna por me ouvir e me ajudar não só com o mestrado, mas com tudo desde 2020.

"*Se o porco inteiro fosse perfeito, não haveria cachorro quente*"
(Greg Universe)

Resumo

Problemas de dominação em grafos estão entre os mais clássicos problemas dentro da área de Teoria de Grafos e são amplamente estudados. Uma variação importante do problema de dominação é o Problema de Dominação Total em Grafos, que consiste em saber se um grafo G tem um subconjunto de vértices de tamanho k que faz vizinhança com todos os demais vértices do grafo, incluindo os próprios vértices do subconjunto.

O Problema de Empacotamento Aberto em Grafos é o problema dual da dominação total de um ponto de vista da programação inteira. Esse problema consiste em saber se um grafo G possui um subconjunto de vértices de tamanho k cuja vizinhança aberta, par a par, não tenha interseção. Embora importante, o Problema de Empacotamento Aberto em Grafos segue em aberto em diversas classes de grafos, isto é, segue sem classificação de dificuldade.

Esse trabalho, então, busca estudar a complexidade computacional desse problema para algumas classes de grafos, assim como busca limitantes superiores para outras. Em particular, encontramos algoritmos que calculam o tamanho do maior empacotamento aberto de grafos de intervalo e de grafos de largura arbórea limitada em tempo polinomial. Além disso, encontramos limites superiores para o tamanho de empacotamentos abertos em grafos com diâmetro igual a dois e para grafos que não possuam K_5 como menor.

Palavras-chave: grafo; classes de grafos; complexidade computacional; dominação; empacotamento.

Abstract

Domination problems in graphs are among the most classic problems in the field of Graph Theory and are widely studied. An important variation of the domination problem is the Total Domination Problem in Graphs, which consists of knowing if a graph G has a subset of vertices of size k that is neighbor to all the other vertices of the graph, including the vertices in the subset itself.

The Open Packing Problem in Graphs is the dual object of total domination from an integer programming point of view. This problem consists of knowing whether a graph G has a subset of vertices of size k whose each pair of open neighborhoods have no intersection. Although important, the Open Packing Problem in Graphs remains open for several graph classes, which means it does not have its difficulty classified.

This work seeks to study the computational complexity of this problem for some classes of graphs, as well as search for upper bounds for others. In particular, we found algorithms that compute the size of the largest open packing of interval and partial k-tree graphs in polynomial time. Furthermore, we found upper bounds for the size of open packings in graphs with a diameter equal to two and for graphs that do not have K_5 as a minor.

Keywords: graph; graph classes; computational complexity; domination; packing.

Lista de Figuras

Lista de Algoritmos

Sumário

Capítulo 1

Introdução

Grafos são estruturas matemáticas discretas muito estudadas tanto do ponto de vista teórico quanto prático. Em ciência da computação, os grafos estão entre as abstrações mais utilizadas na modelagem de problemas reais, pois capturam características fundamentais de problemas envolvendo entidades e suas relações. Uma característica interessante de problemas em grafos, é que mesmo problemas cuja definição é simples e sucinta, podem ser computacionalmente desafiadores. Esta dissertação aborda alguns destes problemas.

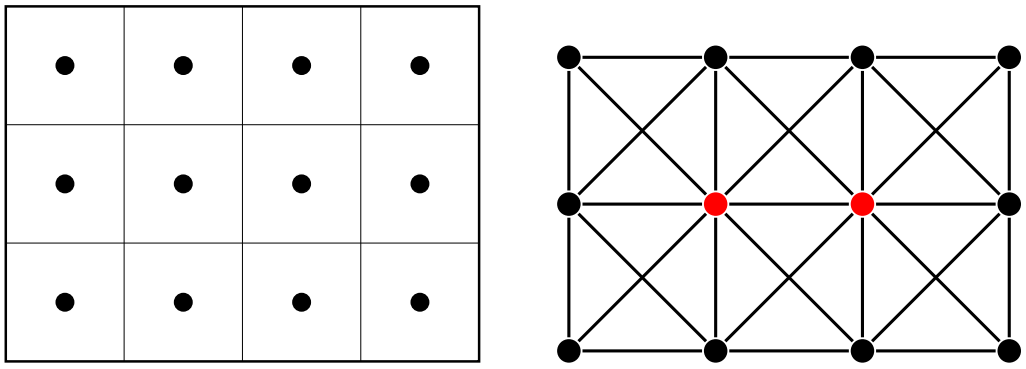
Um conjunto de dominação total em um grafo $G = (V, E)$ é um conjunto de vértices contidos em V de forma que todos os vértices do grafo são adjacentes a pelo menos um vértice do conjunto. Dessa maneira, podemos definir o problema de dominação total em grafos como: dado G, um grafo, e k, um número inteiro, podemos encontrar um conjunto dominante total de tamanho k em G?

Este problema, embora pareça puramente teórico à primeira vista, possui aplicações práticas em diversas áreas, como na otimização de redes de comunicação, vigilância, e distribuição de recursos. A resolução eficiente de problemas de dominação total pode levar a melhorias significativas em sistemas complexos, tornando-se assim um campo de estudo de grande interesse.

Como exemplo, suponha que temos uma galeria de arte dividida em regiões e queremos cobrir essa galeria com câmeras, dado que estão presentes em todas as suas regiões obras caríssimas. Todas as câmeras ficam presas ao teto e podem rotacionar $360^{\underline{o}}$, mas elas só conseguem gravar o que se passa nas suas regiões vizinhas (vizinhança oito) e não na sua própria região, dado que eles não gravam para baixo. O problema que queremos resolver nesse caso é "se podemos colocar câmeras em quaisquer regiões, qual o menor número de câmeras necessárias para cobrir toda a galeria"?

No problema apresentado acima (pode ser visto um exemplo na Figura 1.1), cada câmera representa um vértice do grafo e dois vértices têm uma aresta entre si se suas regiões correspondentes estiverem na vizinhança 8 uma da outra. Assim, procuramos o menor conjunto de dominação total no grafo mostrado anteriormente.

Embora esse seja apenas um exemplo de um tipo específico de dominação, pode-se perceber que dominação, de forma geral, é uma área de Teoria de Grafos de grande importância, sendo, assim, amplamente estudada. Podemos encontrar diversos outros



(a) Exemplo de galeria com cada um dos pontos em que as câmeras podem ser colocadas.

(b) Grafo que mapeia o problema de dominação total da Figura 1.1a.

Figura 1.1: Exemplo de galeria particionada em regiões e grafo resultante, onde cada ponto da Figura 1.1a é um vértice na Figura 1.1b e dois vértices são vizinhos se estão na vizinhança oito um do outro. Os vértices vermelhos são um exemplo de conjunto minimal de dominância total.

exemplos de usos de dominação em grafos em [8]. Entretanto, sabemos que o problema de dominação total é NP-Difícil [15].

A complexidade do problema de dominação total surge não apenas da necessidade de garantir que todos os vértices sejam adjacentes a pelo menos um vértice do conjunto, mas também da busca por soluções ótimas que minimizem o número de vértices no conjunto dominante. Esta otimização é crucial em aplicações práticas, onde recursos como câmeras de vigilância ou pontos de distribuição de serviços são limitados.

Estudos conseguiram encontrar algoritmos polinomiais que resolvem o problema de dominação total para algumas classes de grafos, enquanto para outras conseguiram provar que o problema é NP-Completo (NP-C), como pode ser visto na Tabela 1.1, que apresenta um panorama geral do problema, quando restrito a algumas classes de grafos.

Como pontuado em [12], os objetos duais dos conjuntos dominantes totais são os conjuntos de empacotamento aberto (de um ponto de vista da programação inteira). Um **empacotamento aberto** (EA) é um conjunto de vértices cuja vizinhança aberta de quaisquer dois deles é disjunta. Por serem duais, o tamanho máximo de um conjunto de empacotamento aberto em um grafo G é um limitante inferior para o tamanho mínimo de um conjunto de dominação total nesse mesmo grafo, uma vez que cada um dos elementos do conjunto de empacotamento aberto precisará de um vértice distinto do conjunto de dominação total para cobri-lo, visto que eles não possuem vizinhos em comum.

A dualidade entre dominação total e empacotamento aberto ressalta a profundidade do problema e abre novas perspectivas para abordagens algorítmicas. Enquanto a dominação total visa a cobertura completa dos vértices, o empacotamento aberto foca na separação máxima dos mesmos, oferecendo uma abordagem complementar que pode ser explorada em diversas aplicações, como em estratégias de segurança e alocação de

Classes de Grafos	Complexidade
Grafo Geral	NP-C [15]
Grafo Bipartido	NP-C [24]
Grafo de Comparabilidade	NP-C [24]
Grafo Split	NP-C [23]
Grafo Cordal	NP-C [25]
Grafo Linha	NP-C [21]
Grafo Linha de um Grafo Bipartido	NP-C [21]
Grafo Livre de Garra	NP-C [21]
Grafo Círculo	NP-C [16]
Grafo de Intervalo	P [2]
Grafo de Permutação	P [18]
Grafo Fortemente Cordal	P [7]
Grafo Dualmente Cordal	P [18]
Grafo de Co-comparabilidade	P [19]
Grafo Asteroidal Livre de Trios	P [17]
Grafo de Distância Hereditária	P [18]
Grafo k-polígono (Fixado $k \geq 3$)	P [18]
k-árvore parcial (Fixado $k \geq 3$)	P [27]

Tabela 1.1: Tabela de NP-completude para o problema de Dominação Total para algumas classes de grafo.

recursos. Dado o grafo da Figura 1.1b, um conjunto de empacotamento aberto para esse grafo pode ser visto na Figura 1.2.

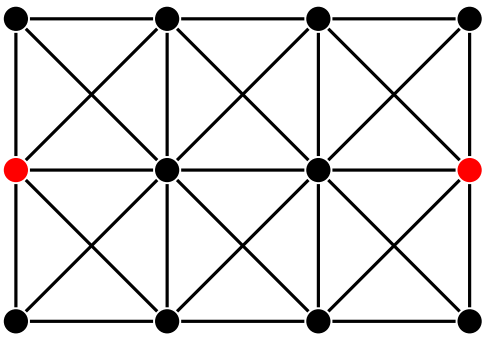


Figura 1.2: Exemplo de conjunto de empacotamento aberto para o grafo da Figura 1.1b. Os vértices em vermelho representam o conjunto.

No exemplo anterior da galeria de arte, poderíamos ilustrar o empacotamento aberto como seguranças que guardam sempre a sua vizinhança, mas por algum motivo não conseguem proteger a própria região em que estão (o que acaba tornando a situação um pouco irreal, mas suponha que nenhum segurança seja bom de combate corpo a corpo, eles só se garantem com algum tipo de arma de longo alcance). O problema que queremos resolver agora é qual o maior número de seguranças que podemos colocar nessa galeria de forma que uma região seja protegida por, no máximo, um segurança.

Nesse trabalho, lidaremos, então, com o número de empacotamento aberto de um grafo G, denotado por $\rho_o(G)$, que consiste na maior cardinalidade dentre todos os conjuntos de empacotamento aberto do grafo. O problema de decisão "Existe empacotamento aberto de tamanho k para o grafo G?", onde k é um número inteiro positivo e G é um grafo geral é NP-Completo [13]. Porém, sabemos que para algumas classes de grafos (como ciclos e caminhos) esse problema é polinomial, ainda que para outras (como grafos bipartidos e cordais) esse problema se mantém NP-Completo. Todos esses resultados podem ser vistos em [13]. Além disso, sabemos que o problema também é NP-Completo para grafos split [26].

1.1 Objetivo

Esse trabalho teve como objetivo estudar a complexidade computacional do problema de empacotamento aberto. Apesar da sua importância, derivada da dualidade com o problema de dominação total, ele segue pouco explorado dando brechas para novas pesquisas.

Veremos em mais detalhes no Capítulo 2, que as únicas classes de grafos já estudadas para esse problema e cujas complexidades foram determinadas são caminhos, ciclos, árvores, bipartidos, cordais e splits.

Investigamos, então, a complexidade computacional desse problema para grafos de intervalo e grafos com largura arbórea limitada, levando em consideração a Tabela 1.1 e buscando algoritmos de tempo polinomial que encontrassem o empacotamento aberto de um grafo dessas classes. Além disso, investigamos também limites para o número de empacotamento aberto de grafos com características específicas.

1.2 Organização do texto

O restante do texto desta dissertação está organizado da seguinte forma. No Capítulo 2 são apresentados as definições, os conceitos e as notações essenciais para o compreendimento desse texto. Conceitos básicos de teoria de grafos importantes para esse trabalho estão presentes na Seção 2.1. Já as classes de grafos importantes para esse trabalho estão definidas na Seção 2.2. Por ser uma classe de grafos importante para esse

trabalho, a Seção 2.3 trata unicamente da classe dos grafos com largura arbórea limitada por k, dado que algumas propriedades específicas dessa classe são complexas, mas devem ser tratadas mais a fundo para que um dos algoritmos apresentados posteriormente possa ser entendido. Na Seção 2.4 apresentamos trabalhos relacionados, que, assim como este, estudaram o empacotamento aberto.

Já no Capítulo 3 começamos apresentando os resultados encontrados que limitam o número de empacotamento aberto para grafos que seguem certas restrições na Seção 3.1. Em seguida, apresentamos algoritmos polinomiais que calculam o número de empacotamento aberto para grafos de intervalo na Seção 3.2.1 e para grafos de largura arbórea limitada na Seção 3.2.2.

Fechamos o trabalho com o Capítulo 4, que sugere quais os passos seguintes para uma continuação desse trabalho.

Capítulo 2

Preliminares

Neste capítulo apresentamos as definições, notações e os trabalhos relacionados que serviram como base para o desenvolvimento deste trabalho. Essas definições e notações são baseadas em [3].

2.1 Teoria dos Grafos

Dado um grafo $G = (V, E)$, onde V é o conjunto de vértices de tamanho n e E é o conjunto de arestas de tamanho m, dizemos que G é um **grafo simples** se todas as suas arestas ligam apenas pares de vértices distintos e G não possui duas ou mais arestas com o mesmo par de extremidades (arestas paralelas). Nesse trabalho, lidamos apenas com grafos simples. Dois vértices u e v pertencentes a $V(G)$ são **adjacentes** se $uv \in E(G)$. A **ordem** de G é justamente o número de vértices do grafo, denotado por $|V(G)| = n$.

Podemos definir a **vizinhança aberta** de um vértice $v \in V$ como $N(v) = \{u \in V \mid uv \in E\}$. Definimos a **vizinhança fechada** de v como $N[v] = \{v\} \cup N(v)$. Para um conjunto $S \in V$, a vizinhança aberta de S é definida como $N(S) = \bigcup_{v \in S} N(v)$ e a vizinhança fechada é definida com $N[S] = N(S) \cup S$.

O grau do vértice $v \in V(G)$, denotado por $d(v)$, é o número de arestas incidentes a v. Em grafos simples, $d(v) = |N(v)|$. Se $d(v) = 0$, v é chamado de **vértice isolado**. Denotamos por $\delta(G)$ o grau mínimo dentre todos os vértices de G, também chamado de **grau mínimo do grafo**, e por $\Delta(G)$ o grau máximo dentre todos os vértices de G, também chamado de **grau máximo do grafo**. Uma **sequência de graus** $d_1, d_2, \ldots, d_n$ de G é uma sequência em que cada elemento é o grau de exatamente um dos vértices de G.

Um **caminho** em um grafo simples é uma sequência de vértices $v_1, v_2, \ldots, v_j$, tal que $j \leq n$, de forma que os vértices consecutivos na sequência são adjacentes. Em um caminho, cada vértice aparece uma única vez na sequência. O tamanho de um caminho é o número de arestas que ele possui. A **distância** entre dois vértices x e y de G é o

tamanho do menor caminho entre eles e é denotada por $dist(x, y)$. O **diâmetro** do grafo G é a maior distância entre dois de seus vértices. Denotamos diâmetro de G por $diam(G)$.

Um grafo é chamado de **conexo** se entre quaisquer dois de seus vértices existe um caminho. De forma geral, todo grafo G pode ser escrito como a união de grafos conexos disjuntos. Cada um desses grafos conexos disjuntos que formam G são chamados, então, de **componentes conexas** de G.

Um **ciclo** em um grafo simples é uma sequência de vértices $v_1, v_2, \ldots, v_j, v_1$, tal que $3 \leq j \leq n$, de forma que vértices consecutivos são adjacentes e, com exceção do primeiro vértice, que também é o último, todos os demais vértices aparecem uma única vez na sequência. O tamanho de um ciclo é o número de vértices que ele possui.

Dado um grafo G qualquer, existem duas maneiras naturais de se derivar grafos menores a partir de G. A primeira maneira é a partir da remoção de arestas desse grafo. A segunda, é apagando vértices (e por consequência, as arestas incidentes a eles). Grafos obtidos a partir de G pela remoção de seus vértices e/ou arestas são chamados de **subgrafos** de G. Um subgrafo é chamado de **subgrafo induzido** de G se ele pode ser obtido removendo-se apenas vértices de G e as arestas incidentes a eles, mas nenhuma outra aresta é apagada. Dizemos que um grafo é **livre de** uma determinada característica/classe se aquele grafo não a possui como subgrafo induzido. Por exemplo, um grafo livre de K_4 é aquele que não possui uma clique de tamanho quatro como subgrafo induzido (a definição de clique será vista a seguir nesse capítulo e K_4 será definido em 2.2).

Dizemos que existe um **isomorfismo** entre os grafos G e H (ou H é isomorfo a G) se existe uma bijeção $f : V(G) \to V(H)$ de forma que dois vértices u e v de G são adjacentes se e somente se $f(u)$ e $f(v)$ de H também o são. Um cópia de um grafo F no grafo G é um subgrafo do grafo G que é isomorfo ao grafo F. Essa cópia também pode ser chamada de F-subgrafo de G.

Identificar vértices não adjacentes x e y de G é substituí-los por um único vértice incidente a todas as arestas que antes eram incidentes a x ou a y em G. Essa operação é denotada por $G/\{x, y\}$. **Contrair** uma aresta $e \in E(G)$ é excluir essa aresta e depois identificar suas extremidades. Essa operação é denotada por G/e. A Figura 2.1 mostra um exemplo para cada uma dessas operações. Um **menor de um grafo** G é um grafo obtido a partir de G por uma sequência de remoções de vértices e arestas e de contrações de arestas.

O **complemento** de um grafo G, denotado por $\overline{G}$, é o grafo cujo conjunto de vértices é $V(\overline{G}) = V(G)$ e o conjunto de arestas $E(\overline{G})$ é tal que se $uv \notin E(G)$, então $uv \in E(\overline{G})$ e se $uv \in E(G)$, então $uv \notin E(\overline{G})$.

Um **conjunto independente** (CI) é um conjunto de vértices $S \subseteq V(G)$ tal que se $u, v \in S$, então $uv \notin E(G)$. Uma **Clique** é um conjunto de vértices $S \subseteq V(G)$ tal que se $u, v \in S$, então $uv \in E(G)$. Os vértices do complemento do grafo induzido por CI formam uma clique, da mesma forma que os vértices do complemento do grafo induzido

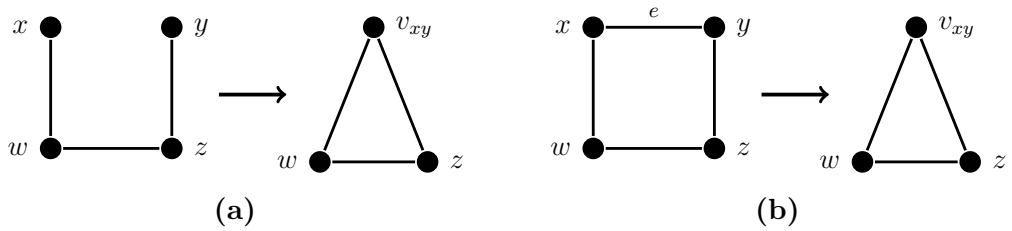


Figura 2.1: Exemplos das operações de identificar vértices e contrair aresta respectivamente. O vértice v_{xy} é o vértice resultante da identificação de x e y.

por uma clique formam um CI. O **número clique** de G é a ordem da maior clique do grafo e é denotado por $\omega(G)$.

Um **conjunto de dominação (ou conjunto dominante)** (CD) em um grafo G é um conjunto $S \subseteq V(G)$ tal que, $\forall v \in V(G)$, ou $v \in S$, ou v é adjacente a um vértice em S. O **número de dominação** de G, denotado por $\gamma(G)$, é o tamanho do menor CD em G. Um **conjunto de dominação total** (CDT) em G é um conjunto $S \subseteq V(G)$ tal que, $\forall v \in V(G)$, v é adjacente a um vértice em S. O **número de dominação total** de um grafo, denotado por $\gamma_t(G)$, é o tamanho do menor CDT em G. Todo grafo sem vértices isolados possui um CDT, uma vez que $V(G) = S$ é um conjunto possível.

Como pontuado em [11], o **empacotamento aberto** (EA) é o objeto dual, do ponto de vista de programação inteira, do CDT. Um **empacotamento** em um grafo G é um conjunto de vértices tais que suas vizinhanças fechadas são disjuntas par a par. O **número de empacotamento superior** de G, que trataremos aqui apenas como número de empacotamento, denotado por $\rho(G)$, é a cardinalidade do maior empacotamento dentre todos aqueles contidos em G. Um **empacotamento aberto** em G é um conjunto de vértices tais que suas vizinhanças abertas são disjuntas par a par. O **número de empacotamento aberto superior** de G, tratado neste trabalho apenas como número de empacotamento aberto, denotado por $\rho_o(G)$, é o tamanho do maior empacotamento aberto em G.

Embora dominação total em grafos seja um assunto amplamente estudado dentro da área de Teoria de Grafos, como pode ser visto em [12], e empacotamento aberto esteja diretamente ligado a ele, EA ainda não possui tantos estudos quanto o seu dual.

Vamos então definir o seguinte problema de decisão:

PROBLEMA DE EMPACOTAMENTO ABERTO
Instância: Um grafo $G = (V, E)$ e um inteiro positivo $k \leq n$.
Pergunta: G possui um empacotamento aberto de tamanho $\geq k$?

Sabemos por [13] que o problema de decisão acima é NP-Completo.

2.2 Classes de Grafos

Nessa seção são apresentadas as classes de grafos que foram importantes para o trabalho.

Um **grafo caminho** é um grafo constituído apenas por um caminho. Dois vértices não consecutivos na sequência não podem ter arestas entre si. Grafos caminho são denotados por P_k, onde k é o número de vértices que o grafo possui. A Figura 2.2 ilustra o grafo P_3.

Figura 2.2: Exemplo de grafo caminho.

Um **grafo ciclo** é um grafo de pelo menos três vértices constituído apenas por um ciclo, assim como no caso de grafos caminho. Dois vértices não consecutivos na sequência não podem ter arestas entre si. Grafos ciclo são denotados por C_k, onde k é o tamanho do ciclo. A Figura 2.3 ilustra o grafo C_5.

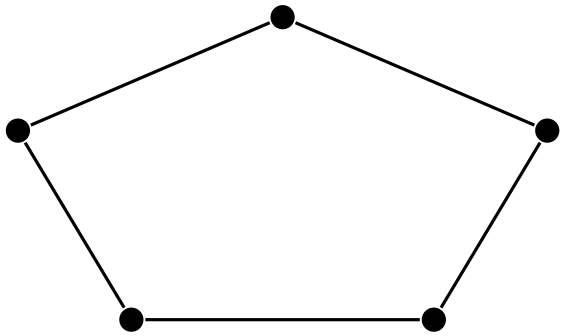


Figura 2.3: Exemplo de grafo ciclo.

Um grafo é **acíclico** se ele não possui nenhum ciclo. Um grafo acíclico conexo é chamado de **árvore**. Um grafo acíclico qualquer, então, é chamado de **floresta**. A Figura 2.4 possui um exemplo de uma árvore.

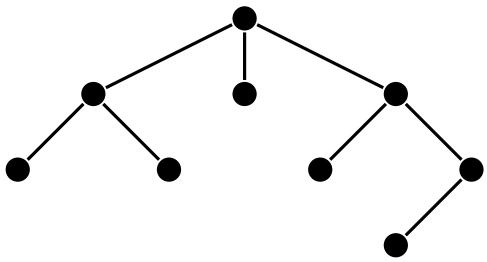


Figura 2.4: Exemplo de árvore.

Em uma árvore é comum pensarmos em um vértice diferenciado r que chamaremos de **raiz**. Esse vértice seria a origem da árvore. Todos os vértices de grau 1 em uma árvore (com exceção, talvez, da raiz) são chamados de **folhas** da árvore.

Um **grafo completo** é um grafo em que todo vértice é adjacente a todos os demais. Ele é denotado por K_n, onde n é sua ordem. A Figura 2.5 ilustra o grafo K_4.

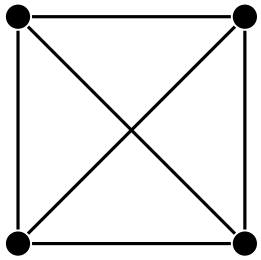
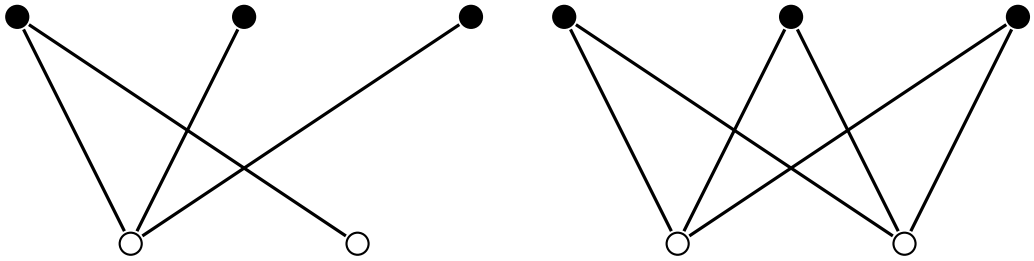


Figura 2.5: Exemplo de grafo completo.

Um **grafo bipartido** é aquele em que seus vértices podem ser particionados em duas classes, X e Y, e todas as arestas do grafo tem uma extremidade em X e a outra em Y. Um grafo é chamado de bipartido completo se ele possui todas as arestas possíveis, isto é, se todo vértice $u \in X$ é ligado a todo vértice $v \in Y$. Os grafos bipartidos completos são denotados por $K_{x,y}$, onde $x = |X|$ e $y = |Y|$. A Figura 2.6b representa um $K_{3,2}$.



(a) **(b)**

Figura 2.6: Exemplos de grafos bipartido incompleto e bipartido completo, respectivamente.

Em um ciclo C de um grafo, uma **corda** é uma aresta pertencente a $E(G) \backslash E(C)$ em que ambas as suas extremidades pertencem ao ciclo. Um **grafo cordal** é um grafo em que em todo ciclo de tamanho maior ou igual a quatro existe uma corda. Outra maneira de definir grafos cordais é dizer que um grafo G é cordal se, e somente se, ele não possui nenhum ciclo de tamanho quatro ou maior como subgrafo induzido. Assim, todo subgrafo induzido de um grafo cordal também é cordal. A Figura 2.7 mostra um exemplo de grafo cordal.

Um grafo é chamado de **grafo split** se seu conjunto de vértices admite uma partição $V(G) = C \cup I$, onde C é uma clique e I é um conjunto independente. O grafo da Figura 2.7 também é um grafo split: os nós a, b, d e e formam uma clique de tamanho quatro e os vértices c e f são um conjunto independente.

Um grafo **planar** é aquele que pode ser desenhado em um plano de forma que suas arestas só se interceptem nas extremidades. Outra maneira de defini-los é dizer que um grafo é planar se, e somente se, ele não possui K_5 ou $K_{3,3}$ como menores [20]. Todos

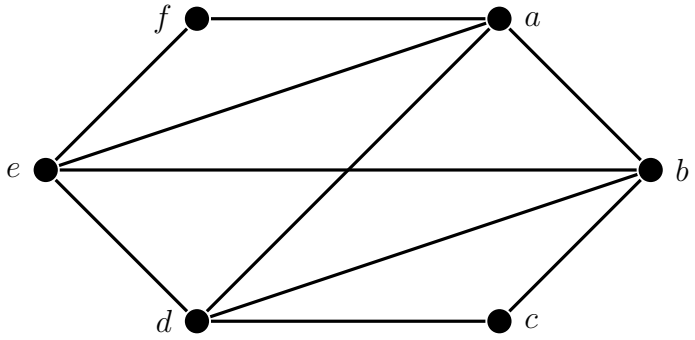


Figura 2.7: Exemplo de grafo cordal.

os grafos desenhados nessa seção até o momento são planares. Mesmo que não estejam desenhados de forma que suas arestas não se interceptem, existe uma maneira de serem desenhados dessa forma. Basta perceber que nenhum deles tem K_5 ou $K_{3,3}$ como menores.

Já um grafo de **intervalo** é um grafo composto por um sistema de conjuntos $H = (\mathcal{V}, \mathcal{F})$, onde $\mathcal{V}$ é o conjunto dos números reais e $\mathcal{F}$ é um conjunto de intervalos fechados contidos nos números reais. Cada intervalo em $\mathcal{F}$ representa um vértice em G e vértices cujos intervalos possuem interseção possuem arestas entre si. A Figura 2.8 apresenta um conjunto H com seus intervalos e o respectivo grafo G.

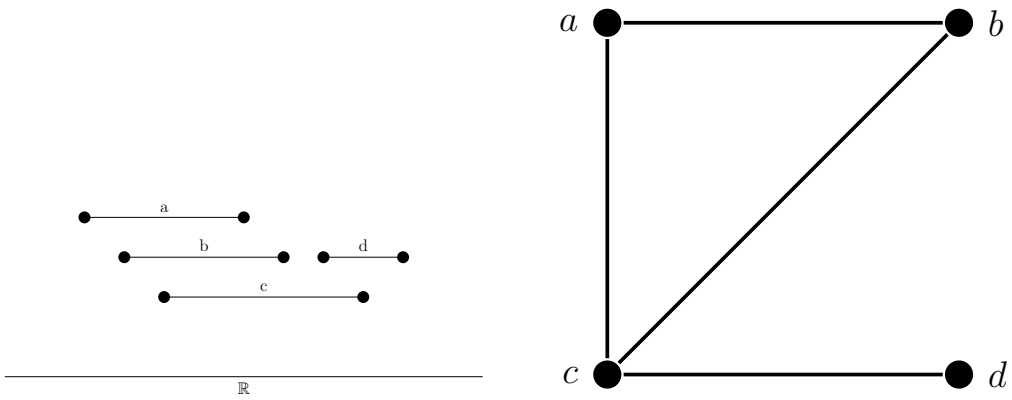


Figura 2.8: Exemplo de conjunto de intervalos e grafo de intervalos resultante.

Uma k**-árvore** é uma classe de grafo que pode ser definida indutivamente da seguinte forma: uma clique com $k+1$ vértices é uma k-árvore. Uma nova k-árvore G pode ser obtida de uma outra k-árvore G' adicionando-se um novo vértice v e fazendo com que ele seja adjacente a k vértices de G' que formam uma clique nesse último grafo. Uma árvore é também uma 1-árvore.

A Figura 2.9 é um exemplo de uma 2-árvore. Podemos obtê-la começando pela clique que contêm os vértices a, b, c e adicionar o vértice d, fazendo com que ele seja vizinho da clique com os vértices b e c. Adicionamos então o vértice e para ser vizinho da clique com os vértices a e b e, logo em seguido, o vértice f, vizinho da clique com

os vértices a e e. Por último, adicionamos os vértice g, vizinho da clique que contém os vértices a e c.

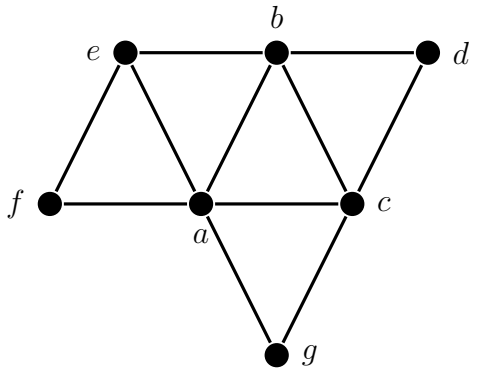


Figura 2.9: Exemplo de uma 2-árvore.

A partir dessa última definição, podemos definir uma **k-árvore parcial (ou grafo de largura arbórea limitada k)** como um subgrafo de uma k-árvore, ou seja, grafos obtidos a partir de k-árvores ao se deletar zero ou mais de suas arestas [1]. A partir do grafo da Figura 2.9 podemos obter o grafo da Figura 2.10.

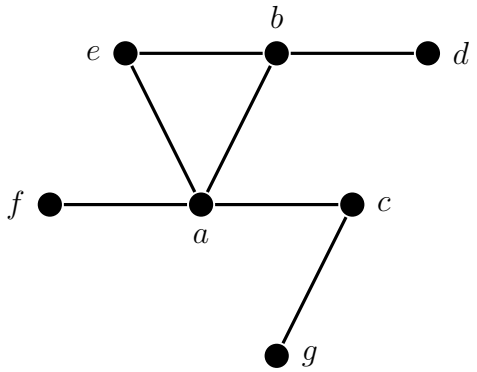


Figura 2.10: Exemplo de uma 2-árvore parcial.

A Seção 2.3 a seguir apresenta outra definição para as k-árvores parciais, que a partir de então serão chamadas apenas de grafos de largura arbórea limitada por k. Essa segunda definição será importante para um dos algoritmos apresentados posteriormente, como resultado dessa dissertação.

2.3 Grafos de largura arbórea limitada por k

Nessa seção, apresentamos uma outra definição para os grafos de largura arbórea limitada por k que foi definida em [9]. Apresentamos também conceitos e propriedades

importantes para essa classe.

Começamos definindo o que é uma **decomposição arbórea**. A decomposição arbórea de um grafo G é um par $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, onde T é uma árvore em que a cada vértice t é atribuído um subconjunto $X_t \subseteq V(G)$, que chamaremos de **bolsa**, seguindo as três condições a seguir:

1. $\bigcup_{t \in V(T)} X_t = V(G)$. Ou seja, todo vértice de G está contido em pelo menos uma bolsa.

2. Para toda aresta $uv \in E(G)$ existe um nó t de T tal que a bolsa X_t contenha tanto u, quanto v.

3. Para todo $u \in V(G)$, o conjunto $T_u = \{t \in V(T) : u \in X_t\}$, i.e., o conjunto de nós de t de T, cujas bolsas X_t correspondentes contêm u, induzem uma subárvore em T.

A **largura** de uma decomposição arbórea é igual a $\max_{t \in V(T)}(|X_t| - 1)$, ou seja, ao tamanho da maior bolsa de T menos 1. Já a **largura arbórea** de um grafo G, denotada por $tw(G)$ (em inglês, largura arbórea é chamada de *treewidth*), é a menor largura possível de uma decomposição arbórea de G. Uma maneira de ver a decomposição arbórea de um grafo é pensar que estamos tentando transformá-lo em uma árvore e a largura arbórea dele mede o quão distante ele está de realmente ser uma. Os grafos conexos com largura arbórea 1 são justamente as árvores, então, intuitivamente, quanto menor a largura arbórea de um grafo conexo, mais próximo de uma árvore ele está.

Podemos dizer que (A, B) é uma **separação** do grafo G se $A \cup B = V(G)$ e não existem arestas entre os subconjuntos $A \backslash B$ e $B \backslash A$. A partir disso, dizemos que $A \cap B$ é um **separador** dessa separação e $|A \cap B|$ é a **ordem da separação**. É importante perceber que qualquer caminho que começa em A e termina em B possui vértices contidos em $A \cap B$. Temos também a definição de **borda de um subconjunto de vértices**, onde seja $A \subseteq V(G)$, a borda de A, denotada por $\partial(A)$, é o conjunto de vértices contidos em A que fazem vizinhança com $V(G) \backslash A$. Note que $(A, (V(G) \backslash A) \cup \partial(A))$ é uma separação de G com separador $\partial(A)$.

Tome por exemplo o grafo da Figura 2.10 e observe a Figura 2.11. Se tomarmos $A = \{a, b, e, f\}$ (conjunto vermelho) e $B = \{a, b, c, d, g\}$ (conjunto azul), (A, B) é uma separação válida de G, dado que a união dos dois conjuntos forma o grafo inteiro e não existem arestas que ligam $A \backslash B = \{e, f\}$ e $B \backslash A = \{c, d, g\}$. O separador nesse caso é $A \cap B = \{a, b\}$ de ordem de separação igual a 2. É fácil ver que tomando A como o mesmo conjunto citado anteriormente, temos que $\partial(A) = \{a, b\}$ e $(V(G) \backslash A) \cup \partial(A)$ é exatamente o B definido anteriormente.

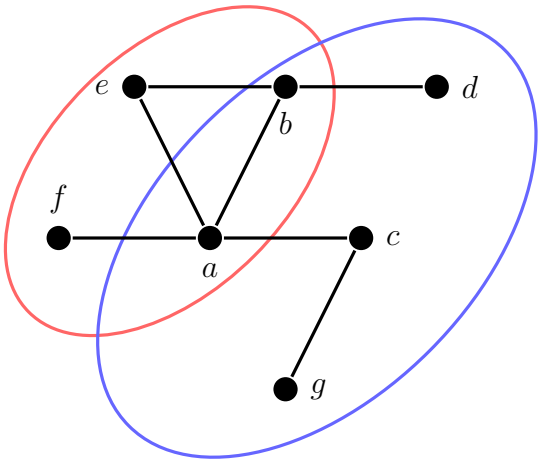


Figura 2.11: Exemplo separação, separador e borda de um subconjunto para o grafo da Figura 2.10.

Em [9] nos é apresentado o seguinte lema:

Lema 1. *Seja $(T, \{X_t\}_{t \in V(T)})$ uma decomposição arbórea de um grafo G e seja ab uma aresta de T. A floresta $T - ab$ obtida a partir de T ao remover-se a aresta ab consiste em duas componentes conexas T_a (que contém a) e T_b (que contém b). Seja $A = \bigcup_{t \in V(T_a)} X_t$ e $B = \bigcup_{t \in V(T_b)} X_t$, então $\partial(A), \partial(B) \subseteq X_a \cap X_b$. Equivalentemente, (A, B) é uma separação de G com separador $X_a \cap X_b$.*

É fácil perceber que se duas bolsas têm seus nós correspondentes adjacentes e contêm os mesmos vértices, podemos contrair a aresta que liga esses nós e obter uma nova decomposição com menos vértices e de mesma largura. Assim, quando conveniente, podemos assumir que bolsas com nós correspondentes adjacentes não contêm os exatos mesmos vértices de G. Então, se $(T, \{X_t\}_{t \in V(T)})$ tem largura k, a partir de Lema 1 temos que cada uma das separações que podemos realizar em T tem ordem de separação de tamanho no máximo k. Isso porque a interseção de dois conjuntos de tamanho máximo $k+1$ que não podem ser iguais, pode ter no máximo k itens em comum.

Outra definição importante para esse trabalho é a definição das boas decomposições arbóreas. Neste trabalho, pensaremos em boas decomposições arbóreas como **árvores enraizadas**. Isso significa que para uma decomposição $(T, \{X_t\}_{t \in V(T)})$, existe um nó distinto r de T que chamaremos de **raiz** de T, introduzindo uma relação natural de parentalidade e ancestralidade em T. Sendo t um nó de T, chamaremos de V_t a união de todas as bolsas presentes na subárvore de T cuja raiz é t, incluindo a bolsa X_t, e G_t é o subgrafo de G induzido pelos vértices presentes nas bolsas de V_t. Perceba que, pelas explicações dadas anteriormente, o subgrafo induzido por V_t só pode se comunicar com o restante do grafo via vértices presentes na bolsa do vértice raiz desta subárvore, que será de tamanho reduzido dado que estamos falando apenas dos grafos com largura arbórea limitada por k.

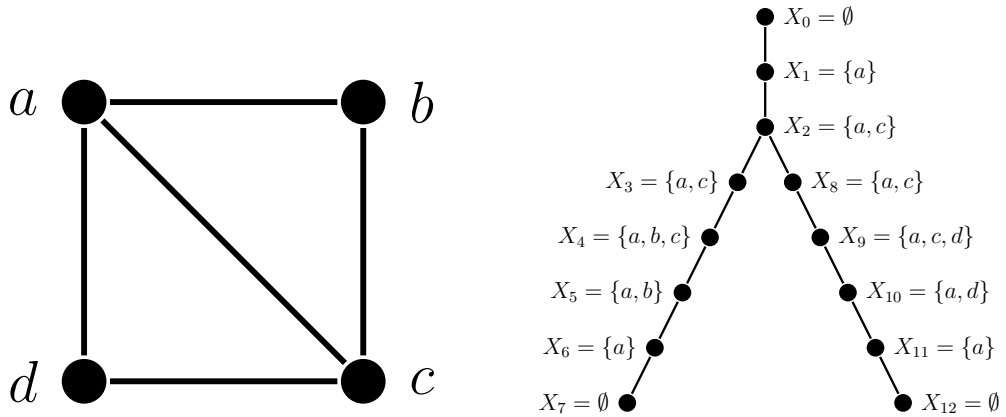
Uma decomposição arbórea é uma **boa decomposição arbórea** se ela satisfaz as seguintes condições:

1. $X_r = \emptyset$ e $X_l = \emptyset$ para toda folha l de T. Assim, tanto a raiz da decomposição arbórea, quanto suas folhas são bolsas vazias, ou seja, não contêm nenhum vértice de G.

2. Todo nó de T que não é uma folha deve ser de um dos seguintes tipos:

   - Nó de **introdução**: um nó t com exatamente um nó filho t' tal que $X_t = X_{t'} \cup \{v\}$ para algum vértice $v \in V(G)$ e $v \notin V_{t'}$. Temos então que v foi introduzido em t.
   - Nó de **esquecimento**: um nó t com exatamente um nó filho t'' tal que $X_t = X_{t''} \backslash \{w\}$ para algum vértice $w \in V(G)$ e $w \in V_{t''}$. Temos então que w foi esquecido em t.
   - Nó de **junção**: um nó t com dois nós filhos t_1 e t_2, tal que $X_t = X_{t_1} = X_{t_2}$.

A Figura 2.12 mostra um grafo e uma boa decomposição arbórea. O nó 0 é a raiz de T. Os nós 7 e 12 são suas folhas. Os nós 4, 5, 6, 9, 10 e 11 são de introdução. Os nós 0, 1, 3 e 8 são de esquecimento. O nó 2 é um nó de junção.



(a) **(b)**

Figura 2.12: Exemplo de um grafo G e de uma boa decomposição arbórea $\mathcal{T}$. Na Figura 2.12b, rotulamos os nós $t \in \{0, \ldots, 12\}$ e os vértices contidos em cada bolsa, entre as chaves.

Note que com os nós de junção, agora o tamanho do maior separador possível passa de k para $k + 1$, mas essa mudança não tem efeito significante no tempo de execução assintótico de algoritmos que usam decomposição arbórea. Por outro lado, o uso da boa decomposição arbórea ajuda bastante a provar a corretude de algoritmos nesses casos. Além disso, em [9] nos é apresentado o seguinte lema:

Lema 2. *Se um grafo G admite uma decomposição de largura no máximo k, então ele também admite uma boa decomposição arbórea de largura no máximo k. Ademais, dada uma decomposição arbórea $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ de G de largura no máximo k, é possível calcular em tempo $O(k \max(|V(T)|, |V(G)|))$ uma boa decomposição arbórea de G de largura no máximo k que tenha no máximo $O(k|V(G)|)$ nós.*

Assim, definimos nesse trabalho a classe dos grafos de largura arbórea limitada k como os grafos que possuem uma boa decomposição arbórea com largura arbórea k.

2.4 Trabalhos Relacionados

O número de empacotamento aberto de um grafo já foi estudado algumas vezes na literatura. Nessa seção, serão apresentados resultados já existentes.

Como dito anteriormente, o problema de Empacotamento Aberto pertence a classe dos problemas NP-Completos [13]. Graças a isso, muitos estudos na área de empacotamento aberto buscam encontrar limites superiores para $\rho_o(G)$ dadas certas características de G. Esse limite algumas vezes depende de $\delta(G)$ ou $\Delta(G)$, mas pode também depender dos graus dos vértices de forma mais geral, como na proposição, abaixo encontrada em [13].

Proposição 1. *Seja $G = (V, E)$ um grafo de ordem $n \geq 2$ com sequência de graus $d_1, d_2, \ldots, d_n$ onde $d_1 \leq d_2 \leq \ldots \leq d_n$. Então $\rho_o(G) \leq max\{k \mid d_1 + d_2 + \ldots + d_k \leq n\}$.*

Essa proposição nos diz que $\rho_o(G)$ é menor ou igual que k, tal que a soma dos k menores graus dos vértices é menor ou igual que n. A partir dessa proposição os autores conseguiram como resultado imediato que se G é um grafo de ordem $n \geq 2$ e grau mínimo δ, então $\rho_o(G) \leq n/\delta$ e que esse limite é o melhor possível, dado que existem grafos onde $\rho_o(G) = n/\delta$, como ilustra a Figura 2.13. Nela, o conjunto de empacotamento aberto do grafo são os vértices pretos.

Em [22], é apresentado o limite a seguir, que estende um resultado previamente encontrado no artigo anterior.

Teorema 1. *Seja $G = (V, E)$ um grafo sem vértices isolados de ordem $n \geq 3$. Então, $\rho_o(G) \leq 2n/3$.*

Podemos encontrar limites para o número de empacotamento aberto que dependem do $\gamma_t(G)$, como no lema abaixo, encontrado em [10].

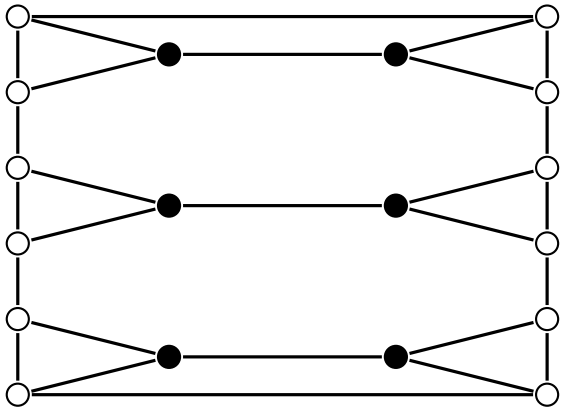


Figura 2.13: Exemplo de grafo G onde $\rho_o(G) = n/\delta$.

Teorema 2. *Seja $G = (V, E)$ um grafo de ordem n com $\delta(G) \geq 2$. Então $\rho_o(G) \leq n - \gamma_t(G)$.*

Outro limite encontrado pelos mesmos autores pode ser visto a seguir.

Lema 3. *Seja $G = (V, E)$ um grafo de ordem $n \geq 3$. Então $\rho_o(G) = 1$ se, e somente se, $diam(G) \leq 2$ e todo vértice do grafo pertence a um triângulo em G.*

Eles também encontraram um limite para $\rho_o(G)$ relacionando-o com o $\omega(G)$.

Teorema 3. *Para qualquer grafo conexo G de ordem $n \geq 3$, $\rho_o(G) \leq n - \omega(G) + 1$ com igualdade se, e somente se, G é K_n ou um grafo obtido a partir de K_{n-1} com a adição de um vértice v ligado a exatamente um vértice de K_{n-1}.*

Alguns estudos da área trabalham procurando valores exatos para $\rho_o(G)$ para classes de grafos específicas. Como dito anteriormente, sabemos que encontrar $\rho_o(G)$ para caminhos e ciclos é polinomial. Assim, [13] nos apresenta as seguintes proposições.

Proposição 2. *Para $n \geq 2$:*

$$\rho_o(P_n) = \begin{cases} \frac{n}{2} & \text{se } n \equiv 0 \ (mod\ 4) \\ \lfloor \frac{n+2}{2} \rfloor & \text{caso contrário} \end{cases}$$

Proposição 3. *Para $n \geq 3$:*

$$\rho_o(C_n) = \begin{cases} \frac{n}{2} - 1 & \text{se } n \equiv 2 \ (mod\ 4) \\ \lfloor \frac{n}{2} \rfloor & \text{caso contrário} \end{cases}$$

Em [5], temos que, se T é uma árvore, $\rho_o(T) = \gamma_t(T)$. Como sabemos por [12] que existe algoritmo polinomial que calcula $\gamma_t(T)$, temos então que podemos calcular $\rho_o(T)$ em tempo polinomial.

Já [6] conseguiu encontrar o número de empacotamento aberto para um grafo G conexo sem P_4 ou C_4 como subgrafo induzido e $n \geq 2$. Nesse caso, $\rho_o(G)$ só pode ser 1 ou 2, sendo $\rho_o(G) = 2$ se, e somente se, $\Delta(G) = n - 1$ e $\delta(G) = 1$. Outro resultado obtido nesse artigo pode ser visto a seguir.

Teorema 4. *Seja G um grafo conexo bipartido que não é completo com bipartição (X, Y), onde $2 \leq |X| \leq |Y|$. Então $\rho_o(\overline{G})$ só pode ser 1 ou 2. Além disso, $\rho_o(\overline{G}) = 2$ se, e somente se, ou $|X| = 2$ ou existem dois vértices $x \in X$ e $y \in Y$ tais que $N(x) \supseteq Y \backslash \{y\}$ e $N(y) \supseteq X \backslash \{x\}$.*

Enquanto isso, outro caminho adotado é estudar a complexidade do problema de Empacotamento Aberto em casos particulares mostrando, por exemplo, que o problema é NP-Completo mesmo para classes de grafos específicas. Nesse caso, [13] nos diz que o problema se mantém NP-Completo mesmo para grafos bipartidos ou grafos cordais e [26] nos diz que o problema de Empacotamento Aberto se mantém NP-Completo mesmo para grafos split com $\delta(G) \geq 2$.

Outro estudo interessante feito por [13] é a investigação de como a remoção de uma aresta afeta o número de empacotamento aberto. Eles obtiveram o seguinte resultado.

Teorema 5. *Para toda aresta e' de um grafo G, $\rho_o(G) \leq \rho_o(G - e') \leq \rho_o(G) + 2$, e esse limite é apertado.*

A Tabela 2.1 resume as complexidades já encontradas para o problema do empacotamento aberto e que foram citadas nessa seção. As linhas em azul representam os resultados encontrados nesse trabalho e que serão apresentados no Capítulo 3.

Classes de Grafos	Complexidade
Grafo Geral	NP-C [13]
Grafo Bipartido	NP-C [13]
Grafo Split	NP-C [26]
Grafo Cordal	NP-C [13]
Caminhos	P [13]
Ciclos	P [13]
Árvores	P [5] e [13]
Grafo conexo livre de P_4 e C_4	P [6]
Grafo de Intervalo [3.2.1]	P
k-árvore parcial (Fixado $k \geq 3$) [3.2.2]	P

Tabela 2.1: Tabela de NP-completude para o problema de Empacotamento Aberto para determinas classes de grafo.

Capítulo 3

Resultados Obtidos

Neste capítulo apresentamos os resultados obtidos para o limite do número de empacotamento aberto e algoritmos para o cálculo de $\rho_o(G)$, onde G é um grafo de intervalo ou um grafo de largura arbórea limitada.

3.1 Limites para o número de empacotamento aberto

Como dito anteriormente na Seção 2.4, muito se estuda sobre limites para $\rho_o(G)$. Obtivemos então os seguintes resultados relacionados a limites.

Teorema 6. *Seja $G = (V, E)$ um grafo conexo com $diam(G) = 2$ e seja S um empacotamento aberto em G. Então, $|S| \leq 2$.*

Demonstração. Seja $S = \{v_1, v_2, \ldots, v_{|S|}\}$ um empacotamento aberto de G. Como $diam(G) = 2$, então $\forall u, v \in S$, u e v são vizinhos ou $N(u) \cap N(v) \neq \emptyset$. Mas, pela definição de empacotamento aberto, não pode existir nenhum par $u, v \in S$ tal que $N(u) \cap N(v) \neq \emptyset$. Então, todos os vértices de S são vizinhos entre si sem nenhum vértice em comum nas suas vizinhanças. Entretanto, isso significa que o conjunto S é uma clique e se $|S| \geq 3$ temos que os vértices da clique tem vizinhança aberta em comum, o que não pode acontecer em um empacotamento aberto. Assim, conclui-se que $|S| \leq 2$. □

Corolário 1. *Seja $G = (V, E)$ um grafo conexo com $diam(G) = 2$. Então, $\rho_o(G) \leq 2$.*

Corolário 2. *Seja $G = (V, E)$ um grafo conexo livre de P_4. Então, $\rho_o(G) \leq 2$.*

O Corolário 1 generaliza o resultado do Corolário 2, uma vez que grafos conexos livres de P_4 possuem diâmetro menor ou igual a 2.

Ainda considerando restrições de diâmetro, apresentamos o seguinte resultado para grafos livres de K_5 como menor.

Teorema 7. *Seja $G = (V, E)$ um grafo conexo livre de K_5 como menor e com $diam(G) = 3$ e seja S um empacotamento aberto de G. Então, $|S| \leq 4$.*

Demonstração. Seja $S = \{v_1, v_2, \ldots, v_{|S|}\}$ um empacotamento aberto de G e suponha que $|S| \geq 5$. Encontraremos um K_5 como menor, para obter uma contradição. Se tomarmos dois vértices quaisquer de S, ou eles têm aresta entre si, ou não. Se esses vértices têm aresta entre si, deixamos eles como estão. Caso contrário, sejam u e v dois vértices sem aresta entre si. Como o $diam(G) = 3$, existe pelo menos um vértice de $N(v)$, que é vizinho de algum vértice de $N(u)$. Sejam x e y tais vértices, respectivamente. Contraindo as arestas que ligam v a x e u a y, temos por consequência que os vértices resultantes da contração têm aresta entre si. Note que não pode existir nenhum vértice em $N(v)$ que também esteja em $N(u)$ uma vez que, caso isso acontecesse, u e v não poderiam estar em S. Além disso, x não pode ser outro vértice de S, uma vez que ele e u teriam o vértice de y como vizinho em comum, contrariando novamente a definição de empacotamento aberto. Dessa forma, podemos contrair as arestas que ligam os vértices de S a seus respectivos vizinhos iguais ao x, para todas as duplas de vértices de S que não têm aresta entre si. Enquanto executamos esse procedimento, sempre existirá um representante do vértice de S que será criado ao contrairmos a aresta entre ele e seu vizinho, mas como dois vértices de S nunca são vizinhos ou tem vizinho em comum, o conjunto formado por esses novos vértices e os remanescentes de S mantém o tamanho $|S|$. Porém, isso significa que esse grafo tem um $K_{|S|}$ como menor e, como $|S| \geq 5$, o grafo G tem K_5 como menor, o que é uma contradição. Logo, $|S| \leq 4$. □

A partir do teorema acima, podemos enunciar também o corolário a seguir, dado que grafos planares são grafos livres de K_5.

Corolário 3. *Seja $G = (V, E)$ um grafo planar conexo com $diam(G) = 3$. Então, $\rho_o(G) \leq 4$.*

Sabemos por [13] que se $G = (V, E)$ é um grafo conexo e de ordem $n \geq 3$, então $\rho_o(G) \leq 2n/3$ e esse limite é apertado. Entretanto, na Figura 3.1 o limite dado pelo Teorema 1 é que $\rho_o(G) \leq 16/3$. Enquanto isso, como o grafo da figura também é planar, temos que $\rho_o(G) = 4$, mostrando que o limite do Corolário 3 é apertado.

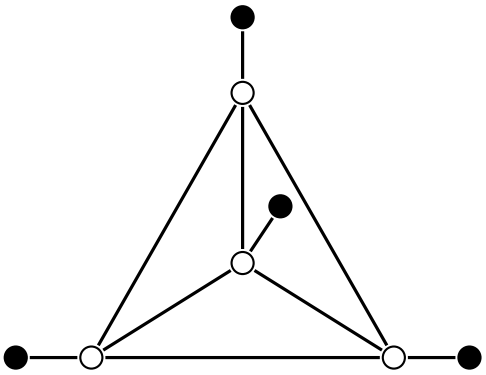


Figura 3.1: Exemplo de grafo planar com $diam(G) = 3$ e $\rho_o(G) = 4$. O empacotamento aberto são os vértices pretos.

3.2 Programação Dinâmica para o número de empacotamento aberto

Enquanto para algumas classes de grafos temos apenas limites para $\rho_o(G)$, em outras conseguimos encontrar o valor exato para ele. Esse é o caso das classes de grafos presentes nessa seção. Por motivos que serão apresentados a seguir, usaremos programação dinâmica para encontrar o valor de $\rho_o(G)$ para grafos dessas classes.

3.2.1 Grafos de Intervalo

Ao estudar grafos de intervalo, percebemos que infelizmente algoritmos gulosos que utilizam ordenação de vértices por tempo de início ou de final não funcionam para encontrar $\rho_o(G)$ nessa classe. Um exemplo pode ser visto na Figura 2.8. Podemos tomar que $a = [1, 4]$, $b = [2, 5]$, $c = [3, 7]$ e $d = [6, 8]$, respeitando a Figura 2.8a.

Se, para tentar encontrar $\rho_o(G)$ para o grafo na Figura 2.8b, utilizarmos um algoritmo guloso com ordenação pelos tempos de início (ou de final) dos intervalos, podemos perceber que o vértice a sempre será escolhido. Entretanto, ao escolhermos o nó a, automaticamente excluímos todos os demais nós, uma vez que todos eles têm vizinhos em comum com a. Por outro lado, podemos perceber que a solução ótima para o maior empacotamento aberto nesse grafo são os vértices c e d.

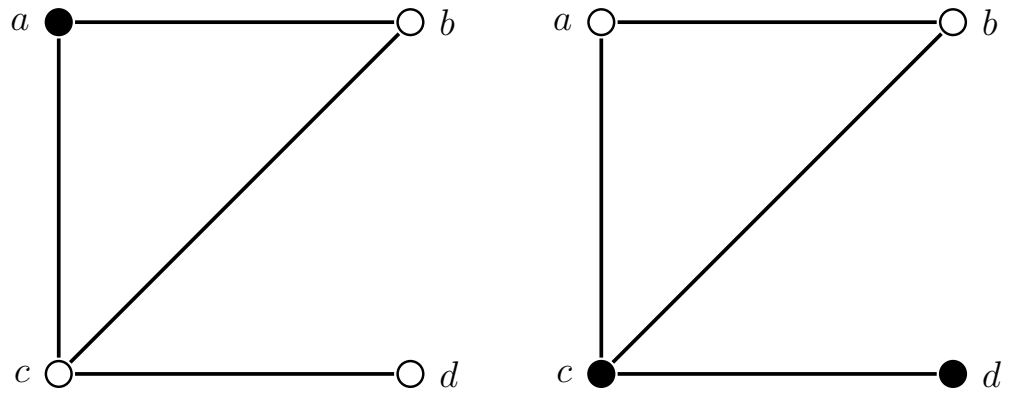


Figura 3.2: Exemplos de empacotamentos abertos (vértices pretos) para o grafo da Figura 2.8b.

Assim, uma segunda abordagem possível seria utilizar um algoritmo de programação dinâmica para encontrar $\rho_o(G)$ em grafos de intervalo. Podemos assumir que estamos trabalhando apenas com grafos conexos pois, caso contrário, é possível utilizar o algoritmo para calcular o ρ_o de cada uma das componentes conexas de G e o valor de $\rho_o(G)$ é a soma desses valores.

Iniciamos definindo alguns conceitos básicos para a implementação do algoritmo. Dado um conjunto de intervalos $\mathcal{F} = \{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ ordenados de acordo com os tempos de final, o grafo de intervalo desse conjunto é G. Os intervalos estão ordenados de tal forma que o i-ésimo intervalo é representado pelo vértice i. Vértices com tempo final igual devem ser ordenados de acordo com o tempo de início, logo se x e y tem tempo de final igual, mas tempo de início de x é menor do que de y, então $x < y$. Vamos criar, então, um intervalo artificial (a_0, b_0), cujo tempo de final é inferior ao menor entre todos os tempos de início ($b_0 < a_i, \forall i \in \mathcal{F}$).

Agora, vamos chamar de $q(i, j)$ o tamanho do maior empacotamento aberto que é um subconjunto de $\{1, \ldots, j\}$ no grafo G, contendo os vértices i e j, tais que $i < j$, e sem conter nenhum vértice k com $i < k < j$, mas que pode conter vértices menores que i. Note que se $i = 0$, então apenas o vértice j pode pertencer ao conjunto. Podemos perceber que, para quaisquer i, j ($0 \leq i < j \leq n$), podemos calcular $q(i, j)$ da seguinte forma:

$$q(i,j) = \begin{cases} 1, & \text{se } i = 0, \\ -\infty, & \text{se } i \text{ e } j \text{ têm vizinhos em comum}, \\ \max\limits_{\substack{0 \leq h < i \\ N(h) \cap N(j) = \emptyset}} q(h,i) + 1, & \text{caso contrário.} \end{cases} \tag{3.1}$$

É importante dizer que a vizinhança de i e j (ou h e j) não deve ser tomada apenas entre os vértices pertencentes a $\{1, \ldots, j\}$, mas em todo o grafo G.

O Algoritmo 1 encontra o tamanho do maior empacotamento aberto de um grafo de intervalo G.

Algoritmo 1 Algoritmo de programação dinâmica para $\rho_o(G)$ em grafos de intervalo

Entrada: $|V(G)| > 0$ e o conjunto de intervalos do grafo G

```
Calcule para cada vértice i do grafo o vetor n_i contendo os vértices que não possuem
vizinhos em comum com ele em todo G e que são menores que ele na ordenação
Crie matriz q de tamanho n × n
para 0 < i ≤ |V(G)| faça
    q(0, i) ← 1
fim para
para 0 < j ≤ |V(G)| faça
    val ← 1
    max ← 1
    para 0 < i < j faça
        se i ∈ n_j então
            para 0 ≤ h < i faça
                se q(h, i) ≥ val e h ∈ n_j então
                    val ← q(h, i) + 1
                fim se
            fim para
        senão
            val ← −∞
        fim se
        q(i, j) ← val
        se q(i, j) > max então
            max ← q(i, j)
        fim se
    fim para
fim para
retorna max
```

Teorema 8. *O Algoritmo 1 computa $\rho_o(G)$, onde G é um grafo de intervalo.*

Demonstração. Podemos perceber que a primeira parte do Algoritmo 1 (até antes do último *para*) apenas executa o que está escrito na Equação 3.1. Então, para provarmos a corretude do algoritmo, precisamos provar que a relação de recorrência está correta.

Comecemos então pelo caso base: como dito anteriormente, se o maior vértice do empacotamento é j e o segundo maior é o "vértice" 0, dado que o intervalo (a_0, b_0) é artificial, temos que o conjunto de empacotamento aberto possui apenas um vértice (o vértice j) e, por consequência, $q(0, j) = 1$.

Já para o segundo caso, pela definição de um empacotamento aberto, se dois vértices possuem vizinhos em comum, então eles não podem estar contidos no mesmo EA. Colocamos então um valor simbólico de menos infinito para saber que o empacotamento aberto contendo i e j não existe.

Já para o último caso da equação, um conjunto de empacotamento aberto pode ser estendido se temos um novo vértice que não possui vizinho em comum com nenhum dos demais vértices do conjunto de EA. Se temos um conjunto de EA onde i é maior vértice e h é segundo maior, queremos provar que j pode entrar nesse conjunto se, e somente se, j e h não tem vizinhos em comum. Isso porque a condição para entrarmos nesse caso é que i e j não tenham vizinhos em comum, então não temos que nos preocupar com vizinhanças entre esses dois vértices. Se j pode ser inserido no EA, então j e h não têm vizinhos em comum, uma vez que, se eles tivessem vizinhos em comum, isso contradiria a definição de um conjunto empacotamento aberto e, por consequência, j não poderia ser inserido no conjunto. Precisamos provar, então, que se j e h não tem vizinhos em comum, então j pode ser inserido nesse conjunto. Se j interceptasse h, os vértices i e h teriam j como vizinho em comum, o que contradiz nossa hipótese, dado que h e i estão em um EA. Então, podemos concluir que $b_h < a_j$. Suponha, então, que existe vértice $h' < h$ que pertence ao conjunto de EA e tem vizinho w em comum com j. Nesse caso, teríamos que $a_w < b_{h'} < b_h$, mas ao mesmo tempo $b_w > a_j > b_h$, uma contradição dado que o intervalo correspondente a w também interceptaria h e, por isso, h' e h teriam vizinho w em comum e não poderiam estar no mesmo EA. Assim, fica claro que os conjuntos que podem ser estendidos por j são exatamente aqueles em que i é o maior vértice, h é o segundo maior e h e j não tem vizinhos em comum. Se tomarmos a cardinalidade de todos esses conjuntos e somarmos um, o maior valor encontrado será justamente a cardinalidade do maior empacotamento aberto contendo j como maior vértice e i como segundo maior.

Dado que temos o valor de $q(i, j)$ para todo $0 < i < j < |V(G)|$, temos todos os empacotamentos abertos maximais do grafo (e mais alguns extras). Sendo assim, o maior entre esses valores é o $\rho_o(G)$.

□

No Algoritmo 1 levamos em consideração que o conjunto de intervalos $\mathcal{F}$ nos é dado, mas sabemos que podemos identificar se um grafo é de intervalo em tempo $O(n+m)$ por [4] e [14], sendo que o segundo nos dá um algoritmo simplificado que encontra o conjunto $\mathcal{F}$ de um grafo, caso ele seja de intervalo, com a mesma complexidade. Além disso, podemos assumir que todas as extremidades dos intervalos de $\mathcal{F}$ são valores inteiros e que o maior deles tem valor linear em n, fazendo com que a ordenação dos vértices no primeiro passo seja linear em n também.

Podemos assumir também que o grafo está representado como uma matriz de adjacências e, para calcular o n_i para cada vértice do grafo, vamos varrer todos os vértices e, para cada um deles, vamos varrer todas as arestas. Para o vértice i, primeiro inicializamos um vetor booleano com 1, onde a j-ésima posição indica se i e j têm vizinhos em comum. Depois, para cada aresta $e = jk$, temos que:

- Se $j < i$ e k é vizinho de i, mudamos a posição j do vetor para zero.
- Se $k < i$ e j é vizinho de i, mudamos a posição k do vetor para zero.

Em todos os demais casos, nenhum valor é alterado. Inicializamos cada vetor booleano em $O(n)$ e, para cada vértice, usamos tempo $O(m)$ para calcular o vetor n_i, totalizando $O(nm)$ para calcular todos os vetores n_i do grafo.

Assim, o vetor n_i é um vetor binário, onde a posição j do vetor é 0 se i e j tem vizinhos em comum e é 1 caso não tenham. O primeiro *"para"* do algoritmo 1 é executado n vezes, uma vez que precisamos varrer todos os vértices de G para encontrar o maior EA. Já o segundo *"para"* executará j vezes, que é limitado por $O(n)$. O terceiro *"para"* será executado $j - 1$ vezes, que tem o mesmo limite do *"para"* anterior. Sendo assim, o Algoritmo 1 é $O(n^3)$. É importante dizer que essa complexidade só é válida uma vez que usamos a estrutura anterior para os vetores n_i. Isso porque checar em cada um dos *"se"*'s se $v_i \in n_j$ é só checar se o vetor n_i na posição j é igual a 1.

Outra observação importante é que o algoritmo pode ser adaptado para encontrar um conjunto $\rho_o(G)$, se desejado. Para isso, basta, para cada $q(i, j)$ calculado, guardar não só seu o valor, mas também os vértices que fazem parte desse empacotamento aberto.

A seguir apresentamos um exemplo de como o Algoritmo 1 funciona.

Exemplo 1. Suponha que tenhamos o conjunto $\mathcal{F}$ da Figura 3.3a e seu respectivo grafo G em 3.3b.

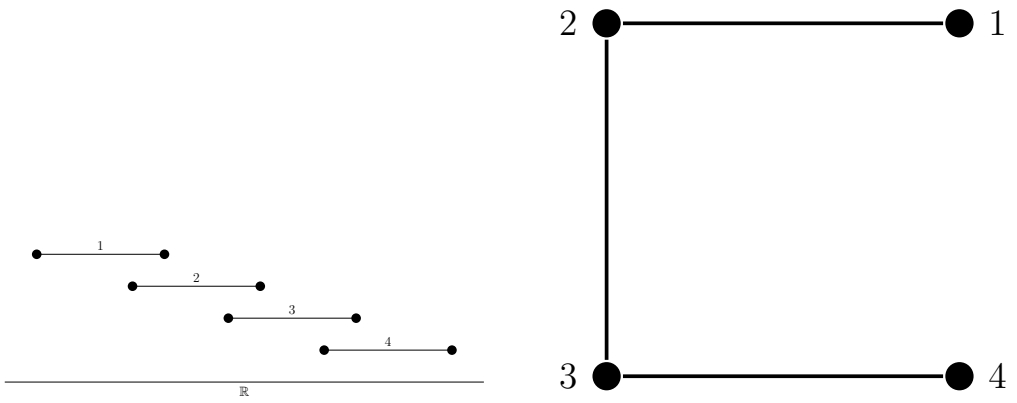


Figura 3.3: Exemplo de um conjunto de intervalos e grafo de intervalos resultante.

Os vetores n_i desse grafo vão ficar como na Figura 3.4.

A partir deles podemos, então, calcular o valor de $\rho_o(G)$. Iniciamos com o valor de $q(0, j) = 1$ para todo $0 < j \leq 4$.

0	1	2	3	4
1	0	1	0	1

(a) n_1

0	1	2	3	4
1	1	0	1	0

(b) n_2

0	1	2	3	4
1	0	1	0	1

(c) n_3

0	1	2	3	4
1	1	0	1	0

(d) n_4

Figura 3.4: Vetores n_i para $i \in \{1, \ldots, n\}$ do grafo em 3.3b

$$
\begin{array}{cc} & j \\ & \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ i & \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & & & & \\ 2 & & & & \\ 3 & & & & \\ 4 & & & & \end{bmatrix} \end{array}
$$

Logo em seguida, vamos tomar $j = 1$. Perceba que começamos preenchendo uma linha inteira, mas a partir de agora vamos começar a preencher coluna por coluna da matriz. Como o único inteiro menor do que 1 na ordenação é 0 e o valor $q(0, 1)$ já foi preenchido, vamos para $j = 2$. Note que exatamente por já ter sido preenchido, vamos pular 0 para o valor de i em todos os demais j's. Para $j = 2$, temos $i = 1$ que é menor que 2 na ordenação e o valor de $n_2(1) = 1$, logo podemos buscar $q(h, 1)$. Como dito anteriormente, o único valor que h pode ter é 0 e como $n_2(0) = 1$, temos então que $q(1, 2) = q(0, 1) + 1 = 1 + 1 = 2$.

$$
\begin{array}{cc} & j \\ & \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ i & \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & & 2 & & \\ 2 & & & & \\ 3 & & & & \\ 4 & & & & \end{bmatrix} \end{array}
$$

Para $j = 3$ temos 1 e 2 como possíveis valores de i, mas, ao analisar o vetor n_3, é possível perceber que $n_3(1) = 0$, logo $q(1, 3) = -\infty$, pois esses dois vértices têm vizinhos em comum e não podem estar na mesma solução. Já $n_3(2) = 1$, permitindo que olhemos $q(h, 2)$. Nesse, caso h pode ser 0 ou 1, mas, pelo mesmo motivo que $q(1, 3)$ não é válido, h só pode ter o valor 0, então $q(2, 3) = q(0, 2) + 1 = 2$.

$$
\begin{array}{cc} & \begin{array}{ccccc} & & & j & & \\ & 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} \\ 0 \\ 1 \\ i \quad 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} & 1 & 1 & 1 & 1 \\ & & 2 & -\infty & \\ & & & 2 & \\ & & & & \\ & & & & \end{bmatrix} \end{array}
$$

Por último, para $j = 4$, $i = \{1, 2, 3\}$. Se $i = 1$ não temos problemas, uma vez que $n_4(1) = 1$ e novamente temos que o único valor válido para h é 0, tendo então que $q(1, 4) = 2$. Porém, para $i = 2$, como $n_4(2) = 0$, temos que $q(2, 4) = -\infty$. Por último, temos $i = 3$ e podemos olhar os valores de $h = 0, 1$, dado que 2 tem vizinho em comum com 4. Como $q(0, 3) = 1$ e $q(1, 3) = -\inf$, o valor final de $q(3, 4) = q(0, 3) + 1 = 2$.

$$
\begin{array}{cc} & \begin{array}{ccccc} & & & & j & \\ & 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} \\ 0 \\ 1 \\ i \quad 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} & 1 & 1 & 1 & 1 \\ & & 2 & -\infty & 2 & \\ & & & 2 & -\infty & \\ & & & & 2 & \\ & & & & & \end{bmatrix} \end{array}
$$

Assim, temos que $\rho_o(G) = 2$, como pode ser provado pelo resultado apresentado na Proposição 3.

3.2.2 Grafos de Largura Arbórea Limitada

Após o estudo de grafos de intervalo, um próximo passo claro parecia o estudo de grafos com largura arbórea limitada, uma vez que muitos problemas nessa classe são resolvidos com algoritmos que usam programação dinâmica. Neste trabalho, assumimos para a programação dinâmica apresentada a seguir que uma boa decomposição arbórea é dada juntamente com o grafo. Em [9], são apresentados algoritmos que computam boas decomposições arbóreas de largura (aproximadamente) ótima.

Para começar as nossas definições, é importante lembrar que X_t é a bolsa associada a $t \in T$ que contém vértices de $V(G)$, V_t é a união de todas as bolsas presentes na subárvore de T cuja raiz é t e G_t é o subgrafo de G induzido por V_t. Além disso, o grafo G é conexo.

Defina como $C_t(A, B)$ o tamanho de um empacotamento aberto máximo de G_t contido em V_t, onde A e B são subconjuntos de X_t, sendo A o conjunto de vértices obrigatoriamente contidos na solução e B o conjunto de vértices cujos vizinhos estão obrigatoriamente na solução. Isso significa que $A \cup B$ pode ser diferente de V_t, uma vez que podem existir vértices em um grafo que não estão no EA e não têm vizinhos nele. Além disso, nem todos os vértices considerados da solução vão estar em X_t, dado que se X_t não é uma folha em T, é possível que vértices contidos em V_t, mas já esquecidos, estejam na solução. Outro ponto é que para cada $C_t(A, B)$ existe um conjunto solução $\mathcal{S}_{AB}$ contido em V_t cuja exata interseção de $\mathcal{S}_{AB}$ com X_t é A e se tomarmos todos os vizinhos dos vértices de $\mathcal{S}_{AB}$ em X_t temos B. Assim, quando o conjunto $\mathcal{S}_{AB}$ é realmente um EA máximo em V_t, temos que $C_t(A, B) = |\mathcal{S}_{AB}|$. É fácil perceber que pode existir mais de um conjunto de EA em V_t que satisfaz as condições de A e B, mas para obter o $\rho_o(G)$ precisamos tomar os conjuntos que contenham não só os vértices de A, mas também a maior quantidade possível de vértices pertencentes a $V_t \backslash X_t$.

Também é importante perceber que nem todos os conjuntos solução vão pertencer ao conjunto de empacotamento aberto máximo de G, dado que A e B vão assumir os valores de todos os subconjuntos de vértices possíveis dentro de cada um dos X_t, inclusive aqueles que não poderiam formar empacotamentos abertos (por isso, em alguns casos, teremos que $C_t(A, B) = -\infty$).

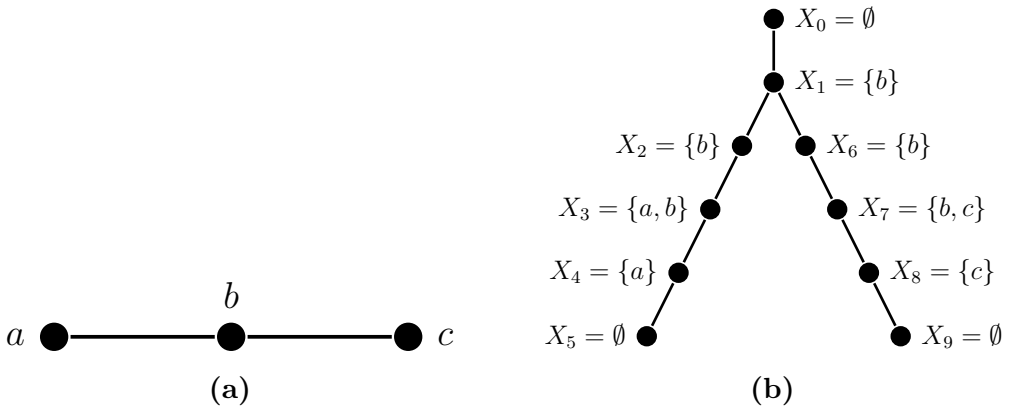


Figura 3.5: Exemplo de um grafo G e sua boa decomposição arbórea $\mathcal{T}$. Na Figura 3.5b, $t = \{0, \ldots, 9\}$ e os vértices contidos em cada bolsa estão entre as chaves.

Tome como exemplo a Figura 3.5. Para V_5 vamos calcular apenas $C_5(\emptyset, \emptyset)$. Já para V_4, vamos calcular $C_4(\emptyset, \emptyset)$, $C_4(\emptyset, \{a\})$, $C_4(\{a\}, \emptyset)$ e $C_4(\{a\}, \{a\})$. Assim, para uma bolsa com k vértices, vamos calcular 2^{2k} valores a partir dos valores presentes na sua subárvore. Para que fique mais claro o que cada um desses valores significa, tome $C_4(\{a\}, \emptyset)$ como exemplo: esse valor irá indicar o tamanho do maior empacotamento aberto presente na subárvore enraizada pela bolsa 4 que obrigatoriamente contenha o vértice $a \in V(G)$ e não tenha nenhum vértice em X_4 que é vizinho de vértices na solução. Porém, analisar apenas essa subárvore é analisar o grafo G_4 que contém apenas o vértice a e o tamanho

do maior empacotamento aberto desse grafo é, obviamente, 1, que é exatamente o EA que contém o a e não existe nenhum vértice que tenha vizinhos na solução (dado que não existem outros vértices no grafo). Assim, temos que $C_4(\{a\}, \emptyset) = 1$.

Vamos, então, começar nosso algoritmo de programação dinâmica pensando nas folhas de T: como $\mathcal{T}$ é uma boa decomposição arbórea, toda folha l é vazia, então temos como única opção $C_l(\emptyset, \emptyset) = 0$, uma vez que não existem vértices na bolsa X_l e ela compõe toda a árvore. No exemplo da Figura 3.5b as bolsas 5 e 9 são as únicas folhas e temos, então, que $C_5(\emptyset, \emptyset) = 0$ e $C_9(\emptyset, \emptyset) = 0$.

Como próximo passo, suponha então que estamos no caso de um **nó de introdução**. Um exemplo de nó de introdução é o 4. A bolsa do nó 4 introduz o vértice a na árvore, dado que no nó 5 temos uma bolsa vazia. Outros exemplos de nós de introdução são os nós 3, 7 e 8. Lembre que, nesse caso, a bolsa t raiz da subárvore tem uma única bolsa filha t' e $X_t = X_{t'} \cup \{v\}$, onde v é um vértice tal que $v \notin X_{t'}$.

Lema 4. *Se t é um nó de introdução, então:*

$$C_t(A,B) = \begin{cases} -\infty & \text{se } N(v) \bigcap A \geq 2, \\ -\infty & \text{se } N(v) \bigcap A = 1 \text{ e } v \notin B, \\ -\infty & \text{se } N(v) \bigcap A = 0 \text{ e } v \in B, \\ -\infty & \text{se } v \in A \text{ e } (N(v) \cap X_t) \not\subseteq B, \\ C_{t'}(A - \{v\}, B - N[v]) + 1 & \text{se } v \in A \text{ e } (N(v) \cap X_t) \subseteq B, \\ C_{t'}(A, B - \{v\}) & \text{se } v \notin A. \end{cases} \tag{3.2}$$

Demonstração. Provemos, então, a corretude da relação de recorrência acima. O primeiro caso nos diz que não temos um valor de $C_t(A, B)$ caso $N(v) \cap A \geq 2$. Isso é verdade, uma vez que dois vértices não podem estar contidos em um mesmo empacotamento aberto de G_t se eles têm um vizinho em comum, nesse caso v. O segundo caso nos diz que também não temos um valor de $C_t(A, B)$ caso $N(v) \cap A = 1$ e $v \notin B$. Isso porque se v tem algum vizinho em A, como $v \in X_t$, v precisa estar em B para que tenhamos conjuntos A e B que formam solução válida. O terceiro caso é bem parecido com os anteriores, pois $C_t(A, B)$ não pode ter valor válido se v não tem nenhum vizinho em A, mas $v \in B$, dado que B é o conjunto de vértices em X_t com vizinhos na solução. O quarto caso é mais um em que $C_t(A, B)$ não pode possuir valor válido por motivo óbvio: se v pertence ao empacotamento aberto, a vizinhança de v contida em X_t deve obrigatoriamente estar em B para que os conjuntos A e B formem solução válida.

Agora apresentaremos os casos em que A e B possivelmente formam solução válida. Se vamos adicionar um vértice novo em um empacotamento aberto já existente, precisamos não só que ele esteja no conjunto A, mas também que $(N(v) \cap X_t) \subseteq B$. Podemos, então, encontrar o valor de $C_t(A, B)$ somando 1 ao valor $C_{t'}(A', B')$, onde $A' = A - \{v\}$ e $B' = B - N[v]$, dado que $\mathcal{S}_{A'B'}$ é o maior empacotamento aberto de $G_{t'}$ em que v pode ser

inserido. Isso acontece pois nenhum vértice na vizinhança de v em G_t se encontra em B', uma vez que, caso contrário, haveria algum vértice que já está em $\mathcal{S}_{A'B'}$ e teria vizinho em comum com v. Ao mesmo tempo, não precisamos nos preocupar com algum vizinho de v em G_t ter sido esquecido ao longo da subárvore, porque como estamos lidando com uma boa decomposição arbórea e t é um nó de introdução, v só pode ter arestas nesse subgrafo com os vértices da bolsa X_t, dado que cada vértice só aparece nas bolsas formando uma subárvore conexa nesse tipo de decomposição e toda aresta do grafo está contida em uma bolsa. Se retirarmos algum outro vértice de A para obter A', teríamos um conjunto solução menor do que o obtido no caso acima e se retirarmos algum outro vértice de B para obter B' ou os conjuntos A' e B' não formariam solução válida, dado que algum vértice estaria em A e seu conjunto de vizinhos não estaria contido em B, ou retiraríamos algum vértice já esquecido do nosso conjunto solução e ele também seria menor que o conjunto encontrado acima.

Por último, basta observar que se $v \notin A$, v pode estar no máximo em B. Se $v \notin B$, então $C_t(A, B)$ é exatamente $C_{t'}(A, B)$. Mas se $v \in B$, repare que se dois vértices em A tem v como vizinho, os conjuntos A e B não seriam válidos e cairíamos no primeiro caso. Por consequência, apenas quando $v \notin A$ e $N(v) \cap A < 2$ consideraremos esse caso e nessa situação temos $C_t(A, B) = C_{t'}(A, B - \{v\})$ pois v não está presente em $G_{t'}$.

Assim, cobrimos todas as possibilidades para um nó de introdução e provamos que a igualdade em 3.2 é válida.

□

Vamos então para o caso em que estamos em um **nó de esquecimento**. A bolsa raiz t tem novamente uma única filha, que chamaremos agora de t'', mas, nesse caso, $X_t = X_{t''} \setminus \{w\}$, onde $w \in X_{t''}$. Um exemplo de nó de esquecimento na Figura 3.5b é o nó 2, que contém em sua bolsa apenas o vértice b. Porém o nó filho dele (o nó 3) contém os vértices a e b. Sendo assim, o vértice a foi esquecido em 2. Outros exemplos de nós de esquecimento são 0 e 6.

Lema 5. *Se t é um nó de esquecimento, temos que:*

$$C_t(A, B) = \max\{C_{t''}(A, B), C_{t''}(A \cup \{w\}, B), C_{t''}(A, B \cup \{w\}), C_{t''}(A \cup \{w\}, B \cup \{w\})\} \tag{3.3}$$

Demonstração. Uma vez que t é um nó de esquecimento, temos que a diferença da bolsa dele para a do seu nó filho é apenas o vértice w que será retirado. Como A e B assumem todos as combinações possíveis, esses conjuntos vão assumir exatamente as mesmas combinações do nó anterior, com exceção dos casos em que w poderia estar em um desses conjuntos. Temos então que A e B (cada um) poderiam ser em $X_{t''}$ de duas formas distintas: exatamente iguais aos possíveis em X_t ou essas mesmas combinações acrescidas de w. As combinações em $X_{t''}$ que contém w seguem sendo EAs em V_t graças as propriedades

da boa decomposição arbórea que nos dizem que w não voltará a aparecer mais acima na árvore e que w só tem arestas para os vértices que compartilham alguma bolsa com ele. Isso porque mesmo se w estiver na solução e tentarmos adicionar posteriormente a ela algum vértice que tenha vizinho em comum com w, embora w não faça mais parte do conjunto A por não fazer parte de X_t, seus vizinhos continuam no conjunto B e esse vértice não poderá ser adicionado no mesmo empacotamento aberto que w, dado que em 3.2 para que v possa ser acrescido a S_{AB}, tomamos em $G_{t'}$ um conjunto solução que não contenha nenhum vizinho de v em B. Se a vizinhança de w também não está mais na bolsa raiz da subárvore, então não existem mais vértices que possuem arestas com essa vizinhança pelas propriedades de boas decomposições arbóreas citadas anteriormente, então não precisamos nos preocupar com um vértice conflitar com w no conjunto solução. Para o caso em que w possuía algum vizinho na solução e foi apagado, basta perceber que w não terá arestas com nenhum vértice mais acima de T e, por consequência, não haverá um novo vértice que terá w como vizinho em comum com algum outro já no EA. Para os casos em que w não está em algum dos dois conjuntos podemos perceber que continuamos com um EA de mesmo tamanho, uma vez que, mesmo retirando w, ainda estamos procurando um EA para todo o V_t e não apenas para os vértices de X_t, o que significa que o tamanho do EA aberto não pode crescer. Ele se mantém igual, considerando, inclusive, os mesmos vértices. Assim, podemos pegar apenas o maior valor encontrado até agora em t'' que contenha os conjuntos A e B acrescidos de, no máximo, w.

□

Por último, temos o **nó de junção**. Nesse último caso, a bolsa raiz t tem duas filhas t_1 e t_2, tal que $X_t = X_{t_1} = X_{t_2}$. Na Figura 3.5b, temos apenas um único nó de junção, o nó 1. Isso fica claro pois ele é o único nó que possui dois filhos.

Lema 6. *Se t é um nó de junção, temos que:*

$$C_t(A, B^*) = \max_{\forall B_{t_1}, B_{t_2} \text{válidos}} \{C_{t_1}(A, B_{t_1}) + C_{t_2}(A, B_{t_2}) - |A|\}, \tag{3.4}$$

onde são considerados válidos todos os pares de subconjuntos $B_{t_1}, B_{t_2} \subseteq X_t$ satisfazendo $(B_{t_1} - N(A)) \cap (B_{t_2} - N(A)) = \emptyset$ e $B_{t_1} \cup B_{t_2} = B^*$.

Demonstração. Para começar essa prova, perceba que, se queremos uma solução em G_t que contenha os vértices de A, precisamos ter esses mesmos vértices nas soluções em G_{t_1} e em G_{t_2}. Isso porque, caso algum vértice $v \in X_t$ estivesse contido na solução de G_{t_1}, mas não estivesse na solução de G_{t_2}, não teríamos como saber se nessa segunda solução esse vértice poderia causar algum tipo de conflito. Quando esse vértice está contido em ambas as soluções, sabemos que ele não tem vizinhança em comum com nenhum vértice dos dois conjuntos solução. Como isso deve ser válido para todos os vértices de A, temos que os vértices na solução contidos em X_{t_1} e em X_{t_2} devem ser iguais, sendo o exato conjunto A em cada um deles.

Quando tratamos dos vértices que possuem vizinhos na solução, dado que A pertence a ambas as soluções, todos os vértices em X_t que são vizinhos de A devem estar em B^*. Porém, se temos um vértice $u \in X_t$ tal que ele não é vizinho de nenhum vértice de A e está contido em B_{t_1} e em B_{t_2}, isso significa que para cada uma das subárvores esse vértice tem um vizinho na solução que já foi esquecido. Isso é um problema, dado que pelas propriedades de uma boa decomposição arbórea esses dois vértices não podem ser o mesmo, dado que um vértice de G só pode aparecer em $\mathcal{T}$ formando uma componente conexa e, nesse caso, esse vértice formaria duas componentes conexas. Então, teríamos dois vértices distintos que estão na solução e compartilham um vizinho, indo contra a definição de empacotamento aberto. Por consequência, temos que $(B_{t_1} - N(A)) \cap (B_{t_2} - N(A))$ deve ser vazio. Além disso, todos os vértices em X_t que possuem vizinhos na solução devem aparecer em B^* e, assim, temos que $B_{t_1} \cup B_{t_2} = B^*$.

Como existem diversos conjuntos B_{t_1} e B_{t_2} que respeitam as regras expostas acima e queremos o maior de empacotamento aberto que respeite os conjuntos A e B^*, devemos pegar o $\max\{C_{t_1}(A, B_{t_1}) + C_{t_2}(A, B_{t_2}) - |A|\}$ levando em conta todos os conjuntos que podem representar B_{t_1} e B_{t_2} seguindo as regras apresentadas acima. Somamos o tamanho das duas soluções pois as duas precisam ser consideradas para evitar conjuntos inválidos e subtraímos $|A|$ do valor pois ao somarmos esses dois valores contamos os vértices de A duas vezes na solução, uma para a solução de cada subárvore. □

A partir dos Lemas 4, 5 e 6 já provados, conseguimos criar o Algoritmo 2, que calcula o número de empacotamento aberto para um grafo de largura arbórea limitada. Chamamos de $f_1(i)$ o valor que ocupada a primeira posição na lista $f(i)$, e $f_2(i)$ o valor na segunda posição.

Teorema 9. *A complexidade do Algoritmo 2 é polinomial.*

Demonstração. Para calcular a ordenação dos vértices, podemos rodar uma DFS em $\mathcal{T}$ e esse algoritmo tem complexidade $O(|V(T)|)$ e, consequentemente pelo Lema 2, tem complexidade $O(n)$. Podemos usar a DFS também para criar o vetor de filhos f: sempre que olharmos os vizinhos do nó i que ainda não foram vistos, adicionamos esses nós em $f(i)$. Note que a lista em $f(i)$ terá no máximo duas posições, sendo fácil checar se um nó j pertence a $f(i)$. Para criar um rótulo podemos usar o vetor f e ver quantos elementos têm o conjunto retornado: caso tenha 2 elementos, temos um nó de junção, caso não tenha nenhum, é uma folha e caso tenha 1 filho, podemos comparar $|X_i|$ e $|X_{f(i)}|$. Se $|X_i| > |X_{f(i)}|$ temos um nó de introdução e caso contrário, temos um nó de esquecimento. Calcular os rótulos para todos os vértices teria complexidade $O(n)$. Sendo assim, a parte mais complexa do algoritmo é calcular o $C_i(A, B)$ para cada i e cada conjunto $A, B \subseteq X_i$.

É importante perceber que, se temos uma boa decomposição arbórea de um grafo qualquer, as relações mostradas anteriormente são válidas. Porém, um algoritmo que faça uso dessas relações facilmente explodiria em um caso geral, uma vez que precisamos

Algoritmo 2 Algoritmo de programação dinâmica para $\rho_o(G)$ em grafos de largura arbórea limitada

Entrada: $|V(G)| > 0$ e $\mathcal{T}$ de G

Ordene os nós de T de forma que um nó só apareça na ordenação se todos os seus filhos já apareceram antes
Crie um vetor f em que a posição i é ocupada por uma lista com os filhos do nó i
Para cada nó, crie um rótulo dizendo se ele é um nó de introdução, esquecimento, junção ou uma folha
para $0 < i \leq |V(T)|$ **faça**
    Crie matriz C_i de tamanho $2^{|X_i|} \times 2^{|X_i|}$
    se i é folha **então** $C_i(\emptyset, \emptyset) \leftarrow 0$
    senão se i é nó de introdução **então**
        Detectar o vértice v que foi introduzido
        para cada conjunto $A, B \subseteq X_i$ **faça**
            se $N(v) \cap A \geq 2$ **então** $C_i(A, B) \leftarrow -\infty$
            senão se $N(v) \cap A = 1$ e $v \notin B$ **então** $C_i(A, B) \leftarrow -\infty$
            senão se $N(v) \cap A = 0$ e $v \in B$ **então** $C_i(A, B) \leftarrow -\infty$
            senão se $v \in A$ e $N(v) \notin B$ **então** $C_i(A, B) \leftarrow -\infty$
            senão se $v \in A$ e $N(v) \in B$ **então** $C_i(A, B) \leftarrow C_{f(i)}(A - \{v\}, B - N[v]) + 1$
            senão se $v \notin A$ **então** $C_i(A, B) \leftarrow C_{f(i)}(A, B - \{v\})$
            fim se
        fim para
    senão se i é nó de esquecimento **então**
        Detectar o vértice w que foi introduzido
        para cada conjunto $A, B \subseteq X_i$ **faça**
            $C_i(A, B) \leftarrow \max(C_{f(i)}(A, B), C_{f(i)}(A \cup \{w\}, B),$
                $C_{f(i)}(A, B \cup \{w\}), C_{f(i)}(A \cup \{w\}, B \cup \{w\}))$
        fim para
    senão
        para cada conjunto $A, B^* \subseteq X_i$ e $B_{f_1(i)} \subseteq X_{f_1(i)}, B_{f_2(i)} \subseteq X_{f_2(i)}$ t.q.
            $(B_{f_1(i)} - N(A)) \cap (B_{f_2(i)} - N(A)) = \emptyset$, $B_{f_1(i)} \cup B_{f_2(i)} = B^*$ **faça**
                $C_i(A, B^*) \leftarrow \max\{C_{f_1(i)}(A, B_{f_1(i)}) + C_{f_2(i)}(A, B_{f_2(i)}) - |A|\}$
        fim para
    fim se
fim para
retorna $C_{|V(G)|}(\emptyset, \emptyset)$

gerar o valor de $C_i(A, B)$ para todos os conjuntos A e B possíveis para todos os nós da decomposição arbórea, já que o tamanho do maior empacotamento aberto de G será o valor de $C_r(\emptyset, \emptyset)$ para o nó raiz r (que na ordenação ocupará a última posição) da árvore T, dado que X_r é uma bolsa vazia. Porém, quando falamos de grafos com *largura arbórea limitada*, limitamos o número de conjuntos gerados em cada passo e conseguimos encontrar o número de empacotamento aberto de um grafo em tempo polinomial, dado que, se a maior bolsa tem tamanho k, geraremos no máximo 2^{2k} valores para cada nó, mas como o tamanho da árvore é polinomial no tamanho do grafo e k é constante, o algoritmo

é polinomial. □

Exemplo 2. Vamos rodar o algoritmo acima para encontrar o número de empacotamento aberto do grafo G da Figura 3.5a, usando sua boa decomposição arbórea que está na Figura 3.5b. Ao rodarmos o algoritmo de ordenação em T, teríamos a nova ordem presente na Figura 3.6.

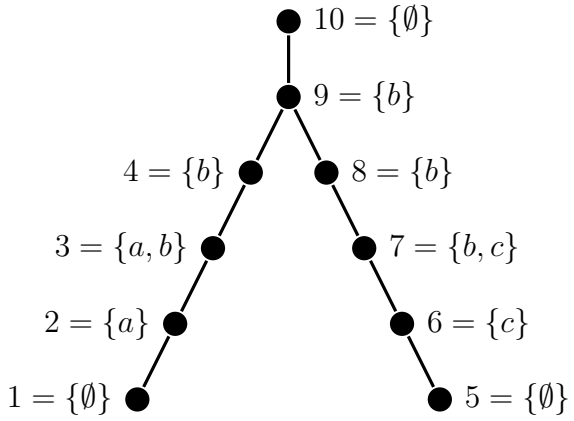


Figura 3.6: Nova ordenação da árvore da Figura 3.5b.

Seguindo com o código para as linhas 2 e 3, teríamos os dois vetores apresentados na Figura 3.7.

1	2	3	4	5	6	7	8	9	10
∅	1	2	3	∅	5	6	7	4,8	9

(a) Vetor de filhos

1	2	3	4	5	6	7	8	9	10
fol	int	int	esq	fol	int	int	esq	jun	esq

(b) Vetor de rótulos

Figura 3.7: Vetores necessários para a execução do algoritmo

Começamos, então, a calcular $C_i(A, B)$ para todos os nós de T. Começando pelo nó 1, sabemos pelo vetor de rótulos que ele é uma folha e temos então que $C_1(\emptyset, \emptyset) = 0$.

Já o nó 2 é um nó de introdução e precisamos calcular o valor de $C_2(A, B)$ para todos os conjuntos A e B possíveis. Como esse nó só possui um vértice em seu conjunto, A e B só podem assumir os valores $\emptyset$ e $\{a\}$. Para $C_2(\emptyset, \emptyset)$ usamos o sexto caso de 3.2 e temos que $C_2(\emptyset, \emptyset) = C_1(\emptyset, \emptyset) = 0$. Já para $C_2(\emptyset, \{a\})$ usamos o terceiro caso, dado que $\{a\} \in B$, mas $N(a) \cap A = 0$ e temos $C_2(\emptyset, \{a\}) = -\infty$. Para $C_2(\{a\}, \emptyset)$ usamos o quinto caso e temos que $C_2(\{a\}, \emptyset) = C_1(\emptyset, \emptyset) + 1 = 1$ e, por último, $C_2(\{a\}, \{a\}) = -\infty$ pelo mesmo motivo de $C_2(\emptyset, \{a\})$. Assim, temos os seguintes valores de $C_2(A, B)$:

$$\begin{array}{cc} & B \\ & \begin{array}{cc} \emptyset & \{a\} \end{array} \\ A \begin{array}{c} \emptyset \\ \\ \{a\} \end{array} & \begin{bmatrix} 0 & -\infty \\ \\ 1 & -\infty \end{bmatrix} \end{array}$$

O nó 3 é também um nó de introdução e precisamos calcular o valor de $C_3(A, B)$ para todos os conjuntos $\emptyset$, $\{a\}$, $\{b\}$ e $\{a, b\}$. Seguindo a mesma lógica do exemplo acima, temos os seguintes valores de $C_3(A, B)$:

$$\begin{array}{cc} & B \\ & \begin{array}{cccc} \emptyset & \{a\} & \{b\} & \{a,b\} \end{array} \\ A \begin{array}{c} \emptyset \\ \{a\} \\ \\ \{b\} \\ \{a,b\} \end{array} & \begin{bmatrix} 0 & 0 & -\infty & -\infty \\ -\infty & -\infty & 1 & -\infty \\ \\ -\infty & 1 & -\infty & -\infty \\ -\infty & -\infty & -\infty & 2 \end{bmatrix} \end{array}$$

O nó 4 é um nó de esquecimento. Como ele só possui um elemento, precisamos calcular $C_4(A, B)$ apenas quando A e B são os conjuntos $\emptyset$ e $\{b\}$. Mas dessa vez usaremos a Equação 3.3. Temos então:

$$\begin{aligned} C_4(\emptyset, \emptyset) &= \max\{C_3(\emptyset, \emptyset), C_3(\{a\}, \emptyset), C_3(\emptyset, \{a\}, C_3(\{a\}, \{a\})\} \\ &= \max\{0, -\infty, 0, -\infty\} = 0 \end{aligned}$$

$$\begin{aligned} C_4(\emptyset, \{b\}) &= \max\{C_3(\emptyset, \{b\}), C_3(\{a\}, \{b\}), C_3(\emptyset, \{a, b\}, C_3(\{a, b\}, \{b\})\} \\ &= \max\{-\infty, 1, -\infty, -\infty\} = 1 \end{aligned}$$

$$\begin{aligned} C_4(\{b\}, \emptyset) &= \max\{C_3(\{b\}, \emptyset), C_3(\{a, b\}, \emptyset), C_3(\{b\}, \{a\}, C_3(\{a, b\}, \{a\})\} \\ &= \max\{-\infty, -\infty, 1, -\infty\} = 1 \end{aligned}$$

$$\begin{aligned} C_4(\{b\}, \{b\}) &= \max\{C_3(\{b\}, \{b\}), C_3(\{a, b\}, \{b\}), C_3(\{b\}, \{a, b\}, C_3(\{a, b\}, \{a, b\})\} \\ &= \max\{-\infty, -\infty, -\infty, 2\} = 2 \end{aligned}$$

É fácil perceber que o lado direito da árvore (nós 5, 6, 7 e 8) é idêntico ao lado esquerdo da árvore, apenas com o vértice a trocado pelo c e, dessa forma, os valores são idênticos respeitando essa troca. Assim, não vamos calcular todos os valores novamente.

Como o nó 9 é um vértice de junção, vamos usar a Equação 3.4, e novamente A e B^* só podem ser os conjuntos $\emptyset$ e $\{b\}$. Sabendo que $f_1(9) = 4$ e $f_2(9) = 8$, $B_4 \subseteq X_4$ e $B_8 \subseteq X_8$ só podem assumir valores tais que $(B_4 - N(A)) \cap (B_8 - N(A)) = \emptyset$ e $B_4 \cup B_8 = B^*$. Assim, temos que:

$$C_9(\emptyset, \emptyset) = \max\{C_4(\emptyset, \emptyset) + C_8(\emptyset, \emptyset) - 0\} = \max\{0 + 0 - 0\} = 0$$

$$\begin{aligned} C_9(\emptyset, \{b\}) &= \max\{(C_4(\emptyset, \emptyset) + C_8(\emptyset, \{b\}) - 0), (C_4(\emptyset, \{b\}) + C_8(\emptyset, \emptyset) - 0)\} \\ &= \max\{(0 + 1 - 0), (1 + 0 - 0)\} = 1 \end{aligned}$$

$$C_9(\{b\}, \emptyset) = \max\{C_4(\{b\}, \emptyset) + C_8(\{b\}, \emptyset) - 1\} = \max\{1 + 1 - 1\} = 1$$

$$\begin{aligned} C_9(\{b\}, \{b\}) &= \max\{(C_4(\{b\}, \emptyset) + C_8(\{b\}, \{b\}) - 1), (C_4(\{b\}, \{b\}) + C_8(\{b\}, \emptyset) - 1)\} \\ &= \max\{(1 + 2 - 1), (2 + 1 - 1)\} = 2 \end{aligned}$$

Como o nó 10 é um nó de esquecimento sem nenhum vértice contido em sua bolsa, pela Equação 3.3 temos que o valor de $C_{10}(\emptyset, \emptyset)$ é exatamente o maior dos valores calculados para $C_9(A, B)$ e temos que $C_{10}(\emptyset, \emptyset) = 2$. Logo, temos que $\rho_o(G) = 2$. É fácil notar que ambos os conjuntos de empacotamento aberto $\{a, b\}$ e $\{b, c\}$ são válidos. Isso fica claro quando calculamos $C_9(\{b\}, \{b\})$, pois os dois valores que ele poderia assumir são iguais, então os dois são valores máximos.

Capítulo 4

Conclusão

Esse trabalho teve como foco o problema de Empacotamento Aberto, que foi anteriormente demonstrado ser NP-Completo para grafos gerais. A relação entre esse problema e o problema de dominação total, muito estudado e relacionado a diversas aplicações, faz com que estudá-lo se torne importante. Além disso, ao contrário do caso de dominação total, para o qual a complexidade do problema foi extensivamente estudada, para diversas classes de grafos, pouco se sabia sobre a dificuldade do problema quando restrito a classes específicas, com raras exceções.

Inicialmente, obtivemos limites superiores para o número de empacotamento aberto para grafos G com $diam(G) = 2$ e para grafos G' livre de K_5 com diâmetro de tamanho 3. Para tais grafos, temos que $\rho_o(G) \leq 2$ e $\rho_o(G') \leq 4$. Além disso, encontramos algoritmos polinomiais que calculam ρ_o para grafos de intervalo e para grafos com largura arbórea limitada por um k fixo.

4.1 Trabalhos Futuros

Apesar dos novos resultados obtidos neste trabalho, muitas direções de pesquisa permanecem viáveis para este problema.

Primeiramente, seria interessante tentar encontrar algoritmos que calculam o número de empacotamento aberto para as demais classes presentes na Tabela 1.1 com complexidade polinomial para o problema de Dominação Total.

Dado que grafos split são um subconjunto de grafos cordais e sabemos que o problema de Empacotamento Aberto é NP-Completo para grafos split, temos que ele também é para grafos cordais. Porém, grafos de intervalo são subconjunto de grafos fortemente cordais que também são subconjunto de grafos cordais. Dado que provamos que o problema é polinomial para grafos de intervalo, a classe de grafos fortemente cordais pode ser, então, um bom próximo passo para um trabalho futuro. Outro caminho possível é tentar provar que para grafos linha ou livres de garra o problema é NP-Completo, assim

como o problema de dominação total.

Outra pergunta natural para esse problema é sobre sua complexidade parametrizada. Embora tenhamos nos focado em classes de grafos, o algoritmo obtido para grafos de largura arbórea limitada pode ser enunciado como um algoritmo tratável por parâmetro fixo parametrizado pela largura arbórea. Uma pergunta ainda mais fundamental seria pensar no parâmetro natural para esse problema: qual a complexidade parametrizada do problema, quando parametrizado por k, o tamanho do empacotamento sendo buscado? Outras direções naturais, ainda neste sentido, seriam outras parametrizações estruturais que não sejam relacionados à largura arbórea, ou que sejam limitados superiormente por alguma função desta, como a largura de clique (*cliquewidth*).

Referências

[1] Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989.

[2] A. A. Bertossi. Total domination in interval graphs. *Inform. Process. Lett.*, 23:131–134, 1986.

[3] A. Bondy and U. S. R. Murty. *Graph Theory*. Springer-Verlag London, 2008.

[4] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[5] B. Brešar, K. Kuenzel, and D. Rall. Graphs with a unique maximum open packing, 2019. arXiv, 1901.09859, math.CO.

[6] K. R. Chandrasekar and S. Saravanakumar. Open packing number for some classes of perfect graphs. *Ural Mathematical Journal*, 6:2:38–43, 2020.

[7] G. J. Chang. Labeling algorithms for domination problems in sun-free chordal graphs. *Discrete Appl. Math*, 22:21–34, 1988.

[8] T. R. Coake. Using domination to analyse RNA structures. Master's thesis, Department of Mathematics, East Tennessee State University, 2005.

[9] M. Cygan, F. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. 2016.

[10] I. S. Hamid and S. Saravanakumar. Packing parameters in graphs. *Discussiones Mathematicae*, 35:5–16, 2015.

[11] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker, 1998.

[12] M. A. Henning. A survey of selected recent results on total domination in graphs. *Discrete Mathematics*, 309:32–63, 2009.

[13] M. A. Henning and P. J. Slater. Open packing in graphs. *JCMCC*, 29:3–16, 1999.

[14] W. L. Hsu and T. H. Ma. Substitution decomposition on chordal graphs and applications. In Wen-Lian Hsu and R. C. T. Lee, editors, *ISA'91 Algorithms*, pages 52–60, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[15] R. M. Karp. Reducibility among combinatorial problems. *Complexity of computer computations*, pages 85–103, 1972.

[16] J. M. Keil. The complexity of domination problems in circle graphs. *Discrete Appl. Math*, 42:51–63, 1993.

[17] D. Kratsch. Domination and total domination on asteroidal triple-free graphs. *Discrete Appl. Math*, 99:111–123, 2000.

[18] D. Kratsch and L. Stewar. Total domination and transformation. *Inform. Process. Lett.*, 63:167–170, 1997.

[19] D. Kratsch and L. Stewart. Domination on cocomparability graphs. *SIAM J. Discrete Math*, 6:400–417, 1993.

[20] C. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta mathematicae*, 15:271–283, 1930.

[21] A. A. McRae. Generalizing np-completeness proofs for bipartite and chordal graphs. *Ph.D Thesis*, 1994.

[22] M. Mohammadi, M. Maghasedi, and Karaj. A note on the open packing number in graphs. *Mathematica Bohemica*, 144:2:221–224, 2019.

[23] J. Pfaff and R. C. Laskar. Domination and irredundance in split graphs. *Technical Report*, 430, 1983.

[24] J. Pfaff, R. C. Laskar, and S. T. Hedetniemi. NP-completeness of total and connected domination and irredundance for bipartite graphs. *Technical Report*, 428, 1983.

[25] J. Pfaff, R. C. Laskar, S. T. Hedetniemi, and S. M. Hedetniemi. On the algorithmic complexity of total domination. *Algebraic Discrete Methods*, 5:420–425, 1984.

[26] I. F. Ramos, V. F. dos Santos, and J. L. Szwarcfiter. Complexity aspects of the computation of the rank of a graph. *DMTCS*, 16:2:73–86, 2014.

[27] J. A. Telle. Complexity of domination-type problems in graphs. *Nordic J. Comput.*, 1:157–171, 1994.