

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA APLICADA

**Decomposição arbórea e localização de
autovalores em grafos**

por

Leonardo Consorte Veit

Dissertação submetida como requisito parcial
para a obtenção do grau de
Mestre em Matemática Aplicada

Prof. Dr. Carlos Hoppen
Orientador

Porto Alegre, maio de 2023

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Veit, Leonardo Consorte

Decomposição arbórea e localização de autovalores em grafos / Leonardo Consorte Veit.—Porto Alegre: PPGMAp da UFRGS, 2023.

91 p.: il.

Dissertação (mestrado)— Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Matemática Aplicada, Porto Alegre, 2023.

Orientador: Hoppen, Carlos

Dissertação: Matemática Aplicada: Matemática Discreta, Teoria Espectral de Grafos, Decomposição Arbórea, Algoritmo de Localização, Grafo Cacto

Decomposição arbórea e localização de autovalores em grafos

por

Leonardo Consorte Veit

Dissertação submetida ao Programa de Pós-Graduação em Matemática Aplicada do Instituto de Matemática e Estatística da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de

Mestre em Matemática Aplicada

Linha de Pesquisa: Matemática Discreta

Orientador: Prof. Dr. Carlos Hoppen

Banca examinadora:

Profa. Dra. Carla Silva Oliveira
ENCE/IBGE

Prof. Dra. Claudia Marcela Justel
IME

Prof. Dr. Rodrigo Orsini Braga
PPGMAp - UFRGS

Prof. Dr. Vilmar Trevisan
PPGMAp - UFRGS

Dissertação apresentada e aprovada em
Maio de 2023.

Prof. Dr. Lucas da Silva Oliveira
Coordenador

SUMÁRIO

LISTA DE FIGURAS	iv
LISTA DE SIGLAS E SÍMBOLOS	viii
RESUMO	ix
ABSTRACT	x
1 INTRODUÇÃO	1
2 CONCEITOS BÁSICOS	5
2.1 Definições	5
2.2 Teoria Espectral de Grafos	11
2.3 Decomposição em Blocos	14
3 DECOMPOSIÇÃO ARBÓREA	17
4 DIAGONALIZAÇÃO DE MATRIZES SIMÉTRICAS DE GRAFOS UTILIZANDO DECOMPOSIÇÃO ARBÓREA	29
5 LOCALIZANDO AUTOVALORES EM ÁRVORES	50
6 GRAFOS CACTOS	61
6.1 Aplicando o Algoritmo Diagonal Arbórea a grafos Cactos	62
6.2 Grafos Cactos Reflexivos	78
7 CONSIDERAÇÕES FINAIS	85
REFERÊNCIAS BIBLIOGRÁFICAS	91

LISTA DE FIGURAS

2.1	Grafo G com 7 vértices.	6
2.2	Subgrafos do grafo G da Figura 2.1: um subgrafo H , o subgrafo induzido $G[\{v_1, v_2, v_3, v_6, v_7\}]$ e o subgrafo $G - X$ com $X = \{v_1\}$	6
2.3	Operação de contração de arestas.	7
2.4	Grafo H é um menor de G	8
2.5	Processo para obtenção de H	8
2.6	Grafo K_6	9
2.7	Ciclo C_6 e um caminho P_4	10
2.8	Uma árvore T	10
2.9	Grafo com representação da matriz M	12
2.10	Grafo G com os seus vértices de articulação destacados em pontilhado circulado.	15
2.11	Grafo G com seus blocos destacados por contornos.	15
2.12	Grafo G com seus blocos e a árvore de blocos.	16
3.1	Grafo G e duas de suas decomposições arbóreas. Os elementos das bolsas estão representados no interior do nó correspondente.	18
3.2	Aresta $e = \{3, 4\} \in B_4$ e o subgrafo conexo de \mathcal{T} induzido pelo nós que contém o vértice $v = 6$	18
3.3	Grafo octaedro, de Wagner e prisma pentagonal.	24

3.4	Um grafo G com 5 vértices e uma decomposição agradável do grafo. Os nós de esquecimento são representados por \mathbf{E} , os nós de junção por \mathbf{J} , os nós de introdução por \mathbf{I} , e os nós folhas por \mathbf{F}	25
3.5	A primeira passagem remove nós cujas bolsas estão contidas nas bolsas dos seus pais.	26
3.6	A segunda passagem equilibra a quantidade de vértices nas bolsas. . .	26
3.7	A terceira passagem substitui vértices de grau grande em uma árvore binária com nós do tipo junção.	27
3.8	Na quarta passagem, adicionam-se nós alternantes para obter uma decomposição agradável.	27
3.9	Passo 5: Esvaziando o último nó para obter uma bolsa vazia.	27
3.10	Decomposição agradável do exemplo 3.2.	28
4.1	Modelo de caixa produzida pelo algoritmo.	30
4.2	Nó folha com sua respectiva caixa.	32
4.3	Nó do tipo introdução e sua respectiva caixa.	33
4.4	Caixas produzidas pelos filhos do nó junção.	34
4.5	Caixa intermediária do nó junção.	34
4.6	Passos intermediários para transformação de N_i^* para N_i	35
4.7	Passos intermediários para transformação de N_i^* para N_i	36
4.8	Permutação de linhas e colunas para v ocupar a primeira posição. . .	36
4.9	Caso 1a: Vetores nulos ou vazios.	37

4.10	Caso 1b: $d_v \neq 0$ e $\mathbf{y}_v \neq [0 \dots 0]$	38
4.11	Caso 1c: $d_v = 0$ e $\mathbf{y}_v \neq [0 \dots 0]$	38
4.12	Passo intermediário para a execução do caso 2.	39
4.13	Tornando elementos aptos para diagonalização.	39
4.14	Decomposição agradável do exemplo 3.2.	41
5.1	Ordenamento dos vértices de T que estende \preceq_T	52
5.2	Processamentos dos vértice 1 a 4 de T	52
5.3	Processamentos dos vértice 5 a 7 de T	53
5.4	A floresta obtida após a realização dos dois primeiros passos na construção de \mathcal{T}	55
5.5	Transição de T para \mathcal{T}	55
5.6	Representação de uma transição de T para \mathcal{T}	56
6.1	Um grafo cacto à esquerda e um grafo que não é cacto à direita.	61
6.2	Ciclo C_b e uma decomposição arbórea \mathcal{T}	62
6.3	Grafo C_4 e uma decomposição agradável \mathcal{T}	65
6.4	Um ciclo C_4 com os valores processados e um caminho P_4 sendo processado.	66
6.5	Ciclo C_5 e uma decomposição agradável \mathcal{T}	69
6.6	Operações em ciclos.	71

6.7	Subgrafos anexos ao vértice de corte \mathbf{i} , bem como a sua representação em blocos, seguida de uma decomposição arbórea.	72
6.8	Junção de estrutura anexa a um vértice \mathbf{i} do ciclo.	74
6.9	Diagonalização com algum filho nulo para o vértice \mathbf{i} do ciclo C	76
6.10	Processamento com filho e vizinho de \mathbf{i} nulos.	77
6.11	Processamento com filho e vizinho de \mathbf{i} nulos com $\mathbf{i} \neq \mathbf{b}$	78
6.12	Os grafos Smith.	80
6.13	Grafo cactos que forma um pacote.	80
6.14	Representação por blocos de um grafo cacto que forma um pacote.	81

LISTA DE SIGLAS E SÍMBOLOS

Lista de Símbolos

$\mathcal{T}(G)$	Decomposição arbórea de G .
$A(G)$	Matriz de adjacências de G .
G	Grafo G .
$M(G)$	Matriz simétrica associada ao grafo G .
T	Árvore T .
$\text{tw}(G)$	Largura arbórea de G .
$w(G)$	Largura de G .

RESUMO

A busca por autovalores de matrizes associadas a grafos é um passo fundamental da Teoria Espectral de Grafos. Neste trabalho, abordamos um Algoritmo de Localização de Autovalores em grafos que utiliza como parte da entrada uma decomposição arbórea. Exemplificamos e detalhamos seu funcionamento para grafos em geral e demonstramos seu desempenho para duas classes específicas. A primeira classe é a das árvores, onde realizamos uma comparação com um algoritmo desenvolvido especificamente para esse propósito. Por fim, utilizamos o algoritmo para a classe de grafos cactos como uma de suas aplicações.

ABSTRACT

The search for eigenvalues of matrices associated with graphs has become one of the important lines of research in the Spectral Graph Theory. In this work, we address an Eigenvalue Localization Algorithm in graphs that uses tree decomposition as part of its input. We exemplify and detail its operation for a general graph and demonstrate its performance for two specific classes. The first class is that of trees, where we make a comparison with an algorithm developed for this purpose. Finally, we use the algorithm for the class of Cactus graphs as one of its applications.

1 INTRODUÇÃO

A teoria dos grafos é um ramo da matemática que estuda as propriedades de estruturas interconectadas de um determinado conjunto. Os grafos são utilizados para modelar diversas situações reais, como redes de transporte, redes de comunicação, redes sociais, moléculas e muitos outros sistemas complexos. A partir da análise dos grafos, é possível entender melhor as relações entre os elementos de um sistema e descobrir propriedades importantes. A teoria dos grafos envolve conceitos como grafos em superfícies, teoria dos grafos aleatórios, algoritmos em grafos e teoria espectral de grafos, por exemplo. Além disso, a teoria dos grafos está intimamente relacionada a outras áreas da matemática, como a teoria dos conjuntos, a álgebra linear e a teoria dos números.

O foco desta dissertação será o estudo dos autovalores de matrizes associadas a grafos, inserindo-se no campo de pesquisa denominado teoria espectral dos grafos. O estudo dos autovalores é de grande importância em diversas áreas da matemática e suas aplicações incluem, por exemplo, o ranking de páginas na web e análise de redes. As matrizes mais comuns utilizadas para a representação de grafos são a matriz de adjacência, a matriz laplaciana e a matriz de incidência, sendo que cada uma delas reflete diferentes aspectos da estrutura do grafo. Para representações através de matrizes quadradas, algumas das propriedades dos grafos que podem ser inferidas a partir de seus espectros incluem o número de vértices e arestas, a conectividade, e a presença de ciclos.

Com o desenvolvimento de novas técnicas e algoritmos, a teoria dos grafos e seus espectros tem se mostrado uma ferramenta poderosa e tem sido útil em diversas aplicações de grafos, como imersão de grafos no plano, para problemas de particionamento e agrupamento, para a descrição topológica de moléculas [34], para o planejamento de redes eficientes [2] e para a descrição geométrica de conjuntos de dados, por exemplo.

A obtenção do espectro de uma matriz de grafo pode ser uma tarefa desafiadora e computacionalmente custosa, especialmente para grafos grandes. O cálculo do espectro envolve a determinação dos autovalores da matriz, que são as raízes do polinômio característico da matriz. A complexidade computacional envolvida na obtenção do espectro de uma matriz de grafo aumenta de maneira cúbica em relação ao número de vértices do grafo, o que pode tornar o processo impraticável para grafos muito grandes. Nos últimos anos, foram desenvolvidas diversas técnicas para lidar com as limitações dos métodos existentes para obtenção do espectro.

Na teoria espectral dos grafos, o desenvolvimento de algoritmos de localização de autovalores se mostrou uma ferramenta importante. Chamamos de localização de autovalores o seguinte problema de computação simbólica: dado um grafo G , um intervalo real I e uma matriz real e simétrica $M(G)$, determine o número de autovalores de $M(G)$ (contando multiplicidades) que pertencem ao intervalo I .

Em 2011, Jacobs e Trevisan [19] desenvolveram o primeiro algoritmo desse tipo, capaz de localizar autovalores da matriz de adjacência de qualquer árvore T . O algoritmo recebe como entrada uma árvore T e um escalar real α , e é executado com base em um ordenamento dos vértices de T . Em seguida, retorna uma matriz diagonal congruente à matriz $A(T) + \alpha I$, onde $A(T)$ denota a matriz de adjacências da árvore T . Essa técnica tem se mostrado útil para a obtenção de autovalores em outras classes de grafos, além de árvores, e tem contribuído significativamente para a solução de problemas na área de teoria espectral dos grafos. Desde então, algoritmos de localização de autovalores foram desenvolvidos para outras matrizes e outras classes de grafos, como grafos unicíclicos [4], grafos threshold [20] e cografos [21], por exemplo.

O princípio dos algoritmos de localização de autovalores é baseado na Lei de Inércia de Sylvester, que estabelece que matrizes simétricas congruentes têm o mesmo número de autovalores positivos, negativos e nulos. O algoritmo de Jacobs e Trevisan é importante por sua complexidade, que é $O(n)$, onde n é o número de

vértices da árvore de entrada. Essa é a melhor complexidade possível, exceto pela constante implícita na expressão assintótica "O", e é substancialmente menor que a complexidade da aplicação de métodos numéricos a matrizes gerais.

Recentemente, novos algoritmos foram desenvolvidos para localização de autovalores em matrizes de grafos de forma mais geral. Esses algoritmos são aplicáveis a diversas classes de matrizes associadas a grafos, mas sua complexidade depende da existência de uma decomposição conveniente do grafo em questão.

Um desses algoritmos utiliza a decomposição em clique (*clique decomposition* [7]) do grafo de entrada [13], outro utiliza a decomposição arbórea (*tree decomposition*) [32]) do grafo de entrada [12]. A complexidade desses algoritmos é $O(k^2n)$, onde k é um parâmetro associado à decomposição. Assim, esse algoritmo é linear para matrizes associadas a grafos que admitem decomposições com k limitado. Além disso, o parâmetro k está relacionado à qualidade da decomposição, ou seja, quanto menor o valor de k , melhor a decomposição e maior a eficiência do algoritmo.

Embora esses resultados generalizem algoritmos obtidos anteriormente, ainda é necessário explorá-los como ferramenta na solução de problemas em aberto relacionados ao espectro de grafos. De fato, os algoritmos desenvolvidos para classes restritas de grafos foram ferramentas muito úteis para resolver conjecturas importantes na área. Muitas dessas conjecturas envolvem a busca do valor máximo ou mínimo de um determinado parâmetro espectral em uma classe de grafos específica, a caracterização dos objetos que atingem tal valor máximo ou mínimo, ou mesmo a ordenação total dos elementos dessa classe com base no valor de um parâmetro espectral.

Um trabalho explorou especificamente a conjectura de que, para um valor fixo n , a árvore com n vértices de menor energia laplaciana é um caminho, enquanto que a árvore de maior energia laplaciana é uma estrela, que é uma árvore com um vértice central e $n - 1$ folhas ligadas a ele. Utilizando os resultados do artigo

[19], foi possível demonstrar em [11] que a estrela é de fato a árvore com a energia laplaciana máxima dentre todas as árvores com n vértices. No entanto, a conjectura sobre a energia laplaciana mínima permanece sem solução.

Neste trabalho, nosso foco foi compreender o funcionamento do algoritmo desenvolvido para classes mais gerais utilizando a decomposição arbórea [12] e fornecer uma compreensão detalhada de todos os processos executados por esse algoritmo. Em seguida, comparamos com o Algoritmo de Árvore desenvolvido por Jacobs e Trevisan, a fim de obter possíveis semelhanças e entender onde os algoritmos se equivalem. Depois, aplicamos o *Algoritmo Diagonal Arbórea* para uma classe de grafos chamados de cactos.

No Capítulo 2, apresentamos as principais definições e conceitos que serão utilizados ao longo deste trabalho. Já no Capítulo 3, abordamos os principais resultados sobre decomposição arbórea na literatura, incluindo teoremas clássicos e definições relevantes para o tema. No Capítulo 4, apresentamos o *Algoritmo Diagonal Arbórea*, com diversas representações e detalhes para que o leitor possa se familiarizar com os processos envolvidos. Nos Capítulos 5 e 6, aplicamos o *Algoritmo Diagonal Arbórea* para verificar possíveis semelhanças com o *Algoritmo de Árvore* e para a aplicação na classe de grafos cactos, respectivamente. Por fim apresentamos os tópicos de fechamento da dissertação.

2 CONCEITOS BÁSICOS

Diversas situações do mundo real podem ser descritas através de diagramas consistindo de um conjunto de pontos e um determinado conjunto de linhas ligando pares destes pontos. Um exemplo concreto que ilustra a aplicação dos grafos é o gerenciamento de projetos em empresas. Nesse contexto, cada ponto do grafo pode representar uma tarefa que precisa ser realizada, e as linhas entre os pontos indicam as dependências entre as tarefas. Por exemplo, se a tarefa x deve ser concluída antes que a tarefa y possa ser iniciada, uma linha é traçada entre os pontos x e y . Essa abordagem permite visualizar claramente as interdependências entre as tarefas e auxilia na tomada de decisões durante a gestão do projeto. Estruturas matemáticas como essa, que envolvem interconexões entre elementos, são a base do conceito de grafo.

2.1 Definições

Um *grafo* é uma estrutura constituída por um par ordenado $G = (V, E)$ de conjuntos tais que V é um conjunto finito e não vazio e E é formado por subconjuntos de dois elementos de V . Os elementos de V são denominados *vértices* e os elementos de E de *arestas*. Indicamos por $|V|$ e $|E|$, respectivamente, o número de vértices e arestas de G . O número $|V|$ é chamado de ordem do grafo. Nesta dissertação, consideramos unicamente *grafos simples*, isto é, grafos sem arestas múltiplas, sem laços e sem orientação.

Dado um grafo $G = (V, E)$ e $u, v \in V$, dizemos que a aresta $e = \{u, v\} \in E$ é incidente aos vértice u e v . Vértices pertencentes à mesma aresta são ditos adjacentes ou vizinhos.

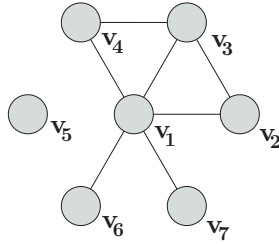


Figura 2.1: Grafo G com 7 vértices.

O grafo representado na Figura 2.1 tem como conjunto de vértices e arestas:

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$$

$$E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_1, v_6\}, \{v_1, v_7\}, \{v_2, v_3\}, \{v_3, v_4\}\}.$$

Definição 2.1. Um grafo H é *subgrafo* de um grafo G se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$, e denotamos por $H \subseteq G$. Dizemos também que G contém H , ou que H está contido em G , ou que G é supergrafo de H . Se $H \subseteq G$, mas $G \neq H$ então dizemos que H é subgrafo próprio de G , e denotamos por $H \subset G$. Se G é um grafo e $\emptyset \neq X \subseteq V(G)$, o subgrafo induzido por X é o subgrafo H de G tal que $V(H) = X$ e $E(H)$ é precisamente o conjunto das arestas de G que têm ambos os extremos em X . Neste caso, H é denotado por $G[X]$.

Denotamos por $G - X$ o subgrafo induzido de G por $V(G) \setminus X$, o qual é o subgrafo obtido de G removendo-se todos os vértices em X e todas as arestas que incidem neles.

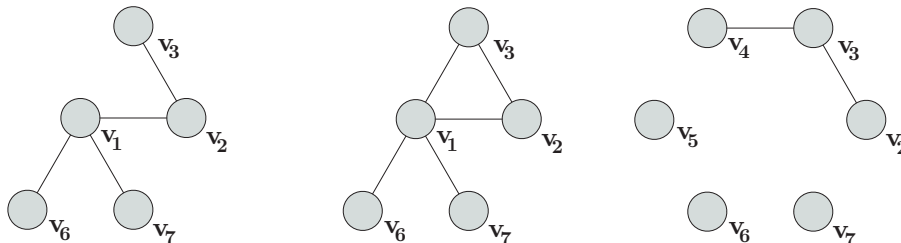


Figura 2.2: Subgrafos do grafo G da Figura 2.1: um subgrafo H , o subgrafo induzido $G[\{v_1, v_2, v_3, v_6, v_7\}]$ e o subgrafo $G - X$ com $X = \{v_1\}$.

Na figura acima, temos um subgrafo de G induzido pelos vértices v_1, v_2, v_3, v_6, v_7 , além do subgrafo obtido a partir de G após a remoção do vértice v_1 , isto é, $G - \{v_1\}$.

Definição 2.2. Uma propriedade é dita *hereditária* quando é satisfeita por um grafo G e por todos os subgrafos induzidos de G .

Em diversas situações, é conveniente modificar a estrutura de um grafo G para formar outro que possa ser mais útil para o estudo de algum tópico ou propriedade. Operações entre grafos constroem novos grafos utilizando informações derivadas do grafo original. Algumas operações apenas alteram a estrutura original do grafo, como a retirada de alguns vértices, arestas ou ambos como nos obtidos no exemplo acima, enquanto outras geram conjunto de vértices e arestas diferentes do original, como a contração de arestas.

Definição 2.3. A contração de uma aresta $e = \{u, v\}$ de um grafo G é uma operação que resulta em um novo grafo G' substituindo-se os vértices u e v por um novo vértice w tal que para todo x, y em $V(G) - \{u, v\}$, $\{x, y\}$ é aresta de G' se e só se $\{x, y\}$ é aresta de G e $\{x, w\}$ é aresta de G' se e só se $\{x, u\}$ ou $\{x, v\}$ é aresta de G . Denotamos o grafo G' obtido de G pela contração da aresta uv por G/uv .

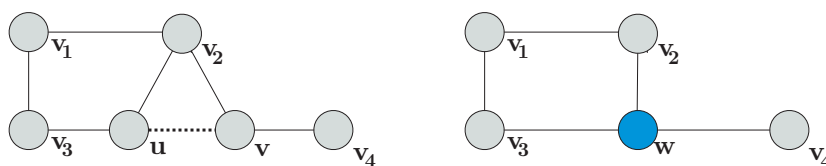


Figura 2.3: Operação de contração de arestas.

Definição 2.4. Um grafo H é um *menor* de G se H pode ser obtido de G por uma série de operações de contrações de arestas, remoção de vértices e arestas.

Uma das primeiras aparições importante de menores de grafos ocorre no Teorema de Wagner [36], que trata de planaridade. Um grafo G é considerado planar se for possível representá-lo no plano sem que as arestas se cruzem. O Teorema de

Wagner estabelece que um grafo G é planar se, e somente se, tanto o grafo completo K_5 quanto o grafo bipartido completo $K_{3,3}$ não forem menores de G .

Exemplo 2.1. O grafo H representado à esquerda na 2.4 é um menor do grafo G mostrado à direita, pois H pode ser obtido a partir de G pela remoção de alguns vértices e/ou arestas ou contração de arestas como explicado abaixo.

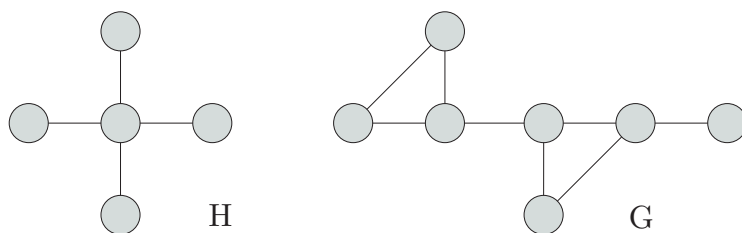


Figura 2.4: Grafo H é um menor de G .

No primeiro passo, as três arestas pontilhadas e o vértice que ficará isolado são removidos, conforme ilustrado na 2.5. Em seguida, a aresta sinuosa é contraída.

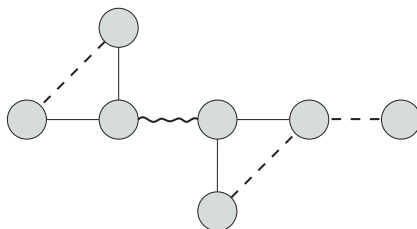


Figura 2.5: Processo para obtenção de H .

Definição 2.5. Uma propriedade de grafos é dita *fechada por menores* se ela é preservada pela operação de obtenção de menores, isto é, se ao se remover vértices, arestas ou contrair arestas de um grafo preservando a propriedade, o grafo resultante também possui essa propriedade.

Definição 2.6. Seja $G = (V, E)$ um grafo. O grau de um vértice v denotado por $d(v)$ é o número de arestas que incidem em v . O grau mínimo de um grafo G é o número $\delta(G) = \min\{d(v) : v \in V\}$ e o grau máximo é o número $\Delta = \max\{d(v) : v \in V\}$. Um vértice v com $d(v) = 0$ é chamado vértice isolado de G . Um vértice *pendente*

é um vértice de grau 1. Um vértice *quase pendente* é um vértice adjacente a um vértice pendente.

O grafo da Figura 2.1 possui $\delta(G) = 0$, $\Delta(G) = 5$, os vértices v_6, v_7 são vértices pendentos, v_1 é um vértice quase pendente e v_5 é um vértice isolado.

Definição 2.7. Um grafo no qual quaisquer dois vértices distintos são adjacentes é dito grafo *completo*. Denotamos por K_n o grafo completo de n vértices.

Definição 2.8. Um grafo é k -regular, ou regular de grau k , se todos os seus vértices possuem grau k .

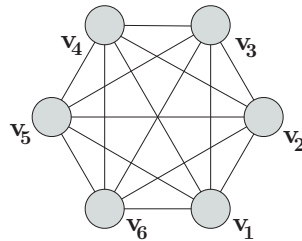


Figura 2.6: Grafo K_6 .

O grafo acima representa o grafo completo de 6 vértices. Note que todos os vértices de um grafo completo possuem o mesmo grau, que é igual a $n - 1$, onde n é o número de vértices do grafo. Portanto, o grafo completo K_n é $(n - 1)$ -regular.

Definição 2.9. Um *passeio* de comprimento n em um grafo $G = (V, E)$ é uma sequência de vértices v_1, \dots, v_n tal que $\{v_i, v_{i+1}\} \in E$ para $1 \leq i \leq n - 1$. Um *caminho* é um passeio sem repetição de vértices. Um ciclo de comprimento n é obtido de um caminho de comprimento n pela adição da aresta $\{v_1, v_n\}$.

O comprimento de um caminho ou ciclo é justamente o número de arestas que este possui. O caminho com n vértices é denotado por P_n , enquanto que o ciclo com n vértices é denotado por C_n . Na Figura 2.7, é possível observar um exemplo de um caminho P_4 e um ciclo C_6 .

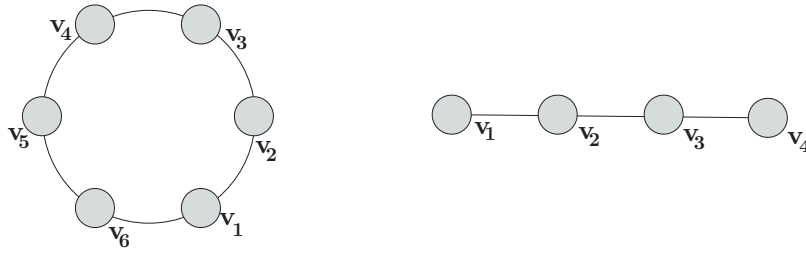


Figura 2.7: Ciclo C_6 e um caminho P_4 .

Definição 2.10. Um grafo é dito *conexo*, se para cada dois vértices u e v de G existe um caminho ligando u e v . Caso contrário, o grafo é dito desconexo.

Definição 2.11. Dado um grafo $G=(V, E)$, a distância entre dois vértices u e $v \in V$ é denotada por $d(u, v)$ e é o comprimento do menor caminho que liga u e v . O máximo das distâncias entre dois vértices de G é chamada de *diâmetro* de G .

Definição 2.12. A união disjunta de dois grafos, $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ é um novo grafo $G = (V, E)$ onde V é a união disjunta dos conjuntos de vértices V_1 e V_2 , ou seja, $V = V_1 \cup V_2$, e E é a união disjunta dos conjuntos de arestas $E = E_1 \cup E_2$.

Definição 2.13. Uma *árvore* é um grafo conexo sem ciclos. A união disjunta de árvores é denominada *floresta*. Os vértices de grau um de uma árvore são chamados de *folhas*.

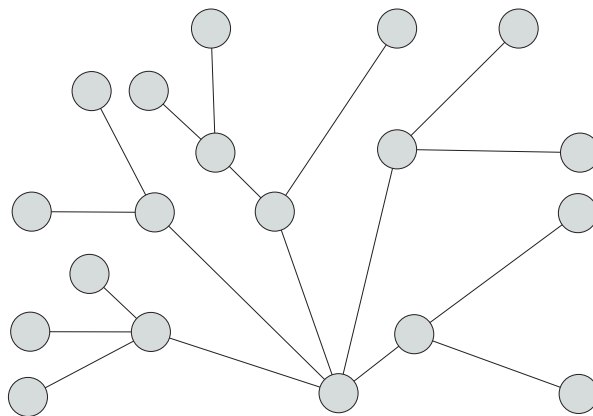


Figura 2.8: Uma árvore T .

Teorema 2.1. [10] *Seja $G = (V, E)$ um grafo com n vértices. As seguintes afirmações são equivalentes:*

- (i) *G é uma árvore.*
- (ii) *G é conexo e possui $n - 1$ arestas.*
- (iii) *G possui $n - 1$ arestas e não possui ciclos.*
- (iv) *Existe exatamente um caminho entre cada par de vértices.*
- (v) *G não contém ciclo e para todo $v, w \in V$ a adição da aresta $e = \{v, w\}$ produz no grafo exatamente um ciclo.*

Uma árvore *enraizada* é uma árvore no qual um vértice em especial é destacado e dito raiz da árvore. O *pai* de um vértice v é o vértice conectado a v no caminho para a raiz. Cada vértice tem um pai único, exceto a raiz que não possui pai. Um filho de um vértice v é um vértice do qual v é o pai. Um descendente de um vértice v é qualquer vértice que seja filho de v , ou um descendente de qualquer um de seus filhos. Uma folha é um vértice sem filhos e um vértice interno é um vértice que não é uma folha. Note que todas as folhas da árvore que não a raiz são vértices sem filhos. A profundidade de um vértice é o comprimento do caminho até a raiz da árvore.

As árvores enraizadas são uma estrutura de dados fundamental na ciência da computação, frequentemente utilizada em algoritmos *bottom-up*. Nessa abordagem, um vértice só é processado após seus descendentes já tiverem sido processados.

2.2 Teoria Espectral de Grafos

Além da representação de um grafo através de uma lista ou diagrama, é possível representá-lo por uma estrutura de adjacência, como uma matriz. Essas estruturas fornecem maneiras mais eficientes de representar um grafo grande ou

denso do que uma representação gráfica. Como computadores são mais eficientes no reconhecimento de números do que de imagens, é comum representar grafos por meio de uma matriz associada. A seguir, apresentaremos a definição de matrizes associadas a grafos.

Definição 2.14. Seja $M = (M_{ij})$ uma matriz real e simétrica $n \times n$, podemos associá-la a um grafo $G = (V, E)$ onde $V = \{1, \dots, n\}$ e $\{i, j\} \in E$ se, e somente se, $M_{ij} \neq 0$ e $i \neq j$.

A matriz M representada a seguir

$$\begin{array}{c}
 v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6 \quad v_7 \\
 \begin{array}{c}
 v_1 \\
 v_2 \\
 v_3 \\
 v_4 \\
 v_5 \\
 v_6 \\
 v_7
 \end{array}
 \begin{bmatrix}
 2 & 7 & 3 & 9 & 0 & 3 & 2 \\
 7 & -3 & 1 & 0 & 0 & 0 & 0 \\
 3 & 1 & 4 & 2 & 0 & 0 & 0 \\
 9 & 0 & 2 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
 3 & 0 & 0 & 0 & 0 & 3 & 0 \\
 2 & 0 & 0 & 0 & 0 & 0 & 7
 \end{bmatrix}
 \end{array}$$

pode ser vista como a representação do grafo ponderado:

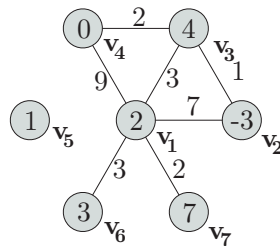


Figura 2.9: Grafo com representação da matriz M .

O grafo da Figura 2.9 é um grafo ponderado, onde cada aresta $\{i, j\}$ possui um peso $m_{ij} \neq 0$, e cada vértice $i \in V$ tem um peso m_{ii} . Reciprocamente, há diversas representações matriciais que associam grafos a matrizes. As matrizes mais conhecidas são a de adjacência, laplaciana, laplaciana sem Sinal, laplaciana Normalizada, entre outras. Ao longo deste trabalho usaremos a matriz de adjacência.

Definição 2.15. (Matriz de Adjacência) Dado um grafo $G = (V, E)$ com n vértices, com conjunto de vértices $V = \{v_1, \dots, v_n\}$ denotamos como matriz de adjacência de G , representada por $A(G)$, a matriz quadrada de ordem n cujos elementos são:

$$a_{ij} = \begin{cases} 1, & \text{se } \{v_i, v_j\} \in E \\ 0, & \text{caso contrário.} \end{cases}$$

Definição 2.16. (Polinômio característico). O polinômio característico de uma matriz quadrada M de ordem n é o polinômio definido como:

$$P_M(\lambda) = \det(\lambda I - M).$$

Em uma matriz quadrada que representa um grafo, o seu conjunto de autovalores podem ser interpretados como autovalores do grafo.

Teorema 2.2. [16] *Se M é uma matriz real simétrica, então seus autovalores são reais.*

Definição 2.17. Considere os autovalores distintos da matriz simétrica M na ordem $\lambda_1 > \dots > \lambda_k$ e as respectivas multiplicidades algébricas $m(\lambda_1), \dots, m(\lambda_k)$. O espectro do grafo G , denotado como $spect(G)$ é definido por uma matriz $2 \times k$, onde a primeira linha é composta pelos autovalores de M e a segunda linha pelas suas respectivas multiplicidades algébricas.

$$spect(G) = \begin{bmatrix} \lambda_1 & \dots & \lambda_k \\ m(\lambda_1) & \dots & m(\lambda_k) \end{bmatrix}$$

Teorema 2.3. [8] *Se G é um grafo conexo e H é um subgrafo próprio de G , então $\lambda_1(H) < \lambda_1(G)$.*

Definição 2.18. Duas matrizes quadradas A e B são ditas semelhantes ou similares, se existe uma matriz C inversível tal que $A = C^{-1}BC$. Se A e B são similares, denotamos por $A \sim B$.

Teorema 2.4. [16] *Se duas matrizes são similares, elas possuem o mesmo polinômio característico.*

Definição 2.19. Duas matrizes são ditas congruentes se existe uma matriz inversível C tal que $A = C^T B C$.

A inércia de uma matriz real simétrica M é uma propriedade que descreve a distribuição dos autovalores de M em relação ao sinal. Em outras palavras, a inércia de M é caracterizada pelo número de autovalores positivos (p), negativos (q) e nulos (r) de M . Essa propriedade é denotada por $i(M) = (p, q, r)$.

Apresentamos a seguir a Lei de Inércia de Sylvester, uma propriedade importante envolvendo congruência de matrizes simétricas. A prova da lei pode ser encontrada em [35], que utiliza propriedades de álgebra linear, ou em uma abordagem mais recente, descrita no exemplo 7.6.4 em [23].

Teorema 2.5. (Lei de Inercia de Sylvester) *Sejam A e B matrizes reais e simétricas, e de mesma ordem. As matrizes A e B são congruentes se, e somente se, elas possuem a mesma inércia.*

Portanto duas matrizes reais simétricas são congruentes, se e somente se, elas têm o mesmo número de autovalores positivos, negativos e nulos. Neste trabalho, é fundamental destacar a importância da Lei de Inércia de Sylvester para os algoritmos de localização de autovalores em grafos. Em capítulos posteriores, será demonstrado como os algoritmos são baseados nesse teorema e como ele contribui para a eficácia desses algoritmos na análise de grafos.

2.3 Decomposição em Blocos

A decomposição em blocos de um grafo é um processo fundamental na Teoria dos Grafos que tem como objetivo dividir o grafo em subgrafos conexos cha-

mados de blocos. Essa técnica permite a identificação de propriedades estruturais importantes do grafo, tais como vértices de articulação e propriedades relacionadas à conectividade. Além disso, os blocos são estruturas que simplificam a análise do grafo, uma vez que proporcionam uma visão geral da sua estrutura. Algumas referências que abordam a definição de blocos e pontos de articulação são [37] e [14].

Definição 2.20. (Vértice de Corte) Um vértice de corte, também conhecido como vértice de articulação, é um vértice em um grafo conexo cuja remoção faz com que o grafo se torne desconexo e aumenta o número de componentes do grafo.

Na Figura 2.10, podemos observar que os vértices **d** e **e** são exemplos de vértices de corte no grafo.

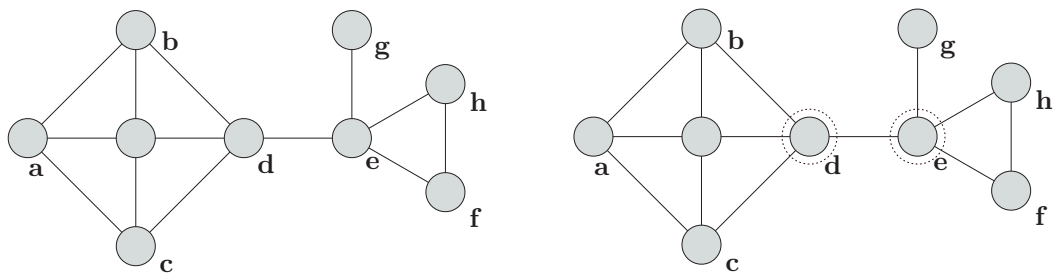


Figura 2.10: Grafo G com os seus vértices de articulação destacados em pontilhado circulado.

Definição 2.21. (Bloco) Um bloco de um grafo é um subgrafo maximal conexo que não contém vértices de corte.

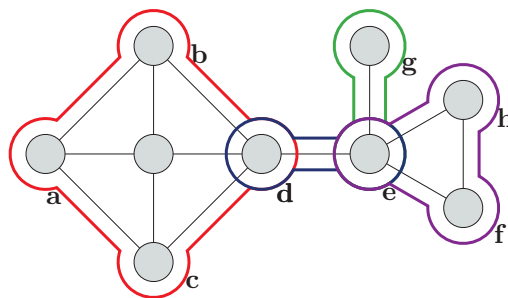


Figura 2.11: Grafo G com seus blocos destacados por contornos.

A decomposição em blocos de um grafo resulta em uma estrutura hierárquica chamada árvore de blocos, onde cada nó da árvore representa um bloco ou um vértice de corte no grafo original. A conexão entre dois blocos é feita por meio de um vértice de corte comum, que é compartilhado pelos dois blocos. A árvore de blocos pode ser usada para analisar as propriedades globais do grafo e para encontrar algoritmos eficientes que exploram a estrutura do grafo. A Figura 2.12 apresenta a árvore de blocos e a estrutura de cada bloco correspondente do grafo da Figura 2.11.

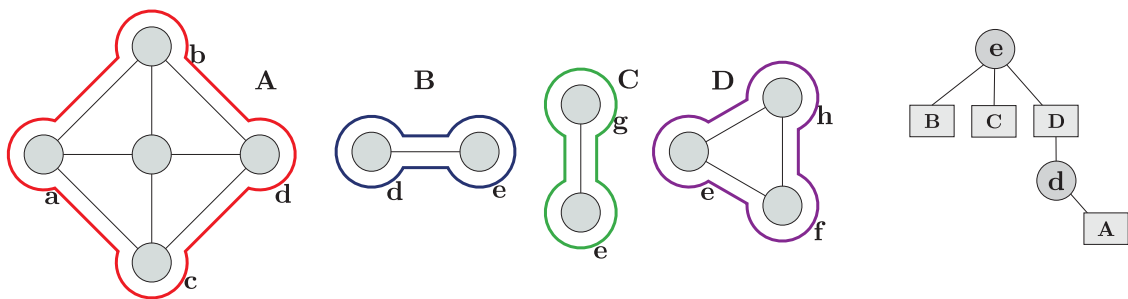


Figura 2.12: Grafo G com seus blocos e a árvore de blocos.

A decomposição em blocos é um passo essencial para compreender a classe de grafos conhecida como cactos e o seu uso no *Algoritmo Diagonal Arbórea*, que será apresentada em capítulos posteriores com mais detalhes. No próximo capítulo, faremos uso de várias definições apresentadas neste capítulo para apresentar os principais resultados acerca da decomposição arbórea, incluindo a demonstração dos principais teoremas clássicos.

3 DECOMPOSIÇÃO ARBÓREA

Inúmeros problemas de otimização combinatória podem ser tratados e modelados por grafos. Dentre as várias vantagens desta abordagem, uma das mais relevantes encontra-se no fato de que, para determinadas classes de grafos, podem ser projetados algoritmos específicos que exploram propriedades estruturais da classe para solucionar o problema com uma complexidade pequena. Dentre as classes de grafos que levam a algoritmos eficientes, destacam-se as árvores, principalmente para elaboração de modelos recursivos. Contudo, a maior parte dos problemas não podem ser modelados em árvores, o que inviabiliza o uso de tal estratégia.

Uma estratégia possível seria estender a modelagem realizada para árvores para outras classes de grafos, esperando que, à medida que o grafo se assemelhe mais a uma árvore, a complexidade seja menor e, se o grafo for menos semelhante, a complexidade seja maior. Uma das abordagens viáveis para essa finalidade é a utilização de decomposições em grafos. Uma delas, que iremos utilizar durante nosso trabalho, é chamada de decomposição arbórea, que foi redescoberta por Robertson e Seymour em 1986 [32].

Definição 3.1. Seja $G = (V, E)$ um grafo com $V = \{1, \dots, n\}$. Uma **decomposição arbórea** ou **decomposição em árvore** de G é uma árvore \mathcal{T} com um conjunto de nós $\mathcal{V} = \{1, \dots, m\}$, na qual cada nó $i \in \mathcal{V}$ está associado a um conjunto $B_i \subset V$, chamado de **bolsa**. Uma decomposição arbórea \mathcal{T} satisfaz:

- 1) $\bigcup_{i=1}^m B_i = V$.
- 2) Para toda aresta $\{u, v\} \in E$ existe $i \in \mathcal{V}$ tal que $u, v \in B_i$.
- 3) Para todo $v \in V$, o subgrafo de \mathcal{T} induzido pelos nós que contêm v é conexo.

Se \mathcal{T} possui um único nó com $B_1 = V$, então \mathcal{T} é chamada decomposição arbórea trivial de G .

Exemplo 3.1. Seja G o grafo cujo conjunto de vértices é dado por $V = \{1, \dots, 7\}$ mostrado na Figura 3.1:

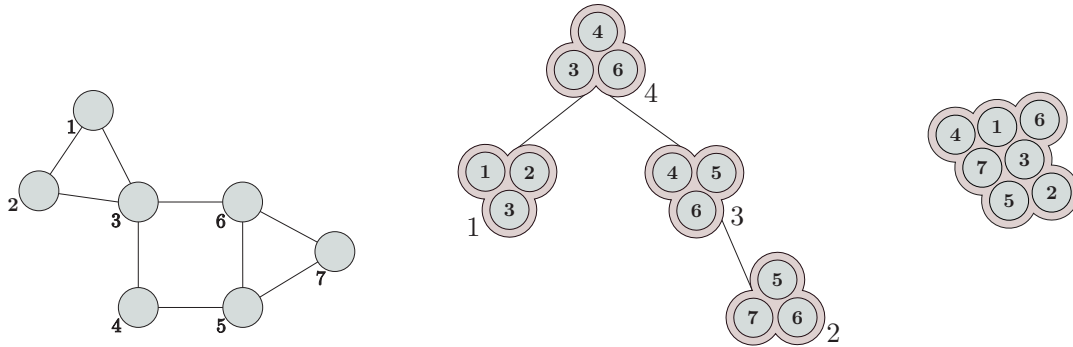


Figura 3.1: Grafo G e duas de suas decomposições arbóreas. Os elementos das bolsas estão representados no interior do nó correspondente.

As decomposições apresentadas acima satisfazem as três condições impostas pela definição. No primeiro caso, podemos verificar que todos os vértices de G e todas arestas estão presente em pelo menos uma bolsa. Por exemplo, a aresta $e = \{3, 4\} \in B_4$. Por fim, note que os nós da decomposição com bolsas contendo $v = 6$ são os nós 2, 3 e 4. Estes nós induzem um caminho na árvore, que é conexo. Já na outra decomposição de G todas condições são satisfeitas trivialmente. Diante disso, as decomposições arbóreas não são únicas e não determinam o grafo.

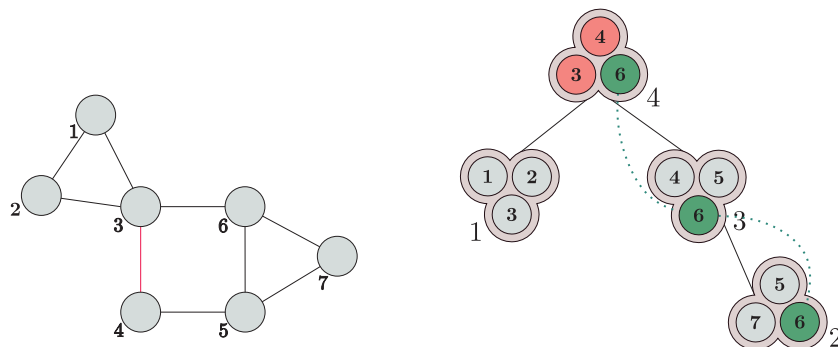


Figura 3.2: Aresta $e = \{3, 4\} \in B_4$ e o subgrafo conexo de \mathcal{T} induzido pelo nós que contém o vértice $v = 6$.

A decomposição arbórea de um grafo G procura estabelecer uma relação estrutural do grafo com a de uma árvore. Um parâmetro associado para avaliar a qualidade da decomposição \mathcal{T} é chamado de **largura**, definido como

$$w(\mathcal{T}) = \max_{i \in \mathcal{V}} \{|B_i| - 1\}.$$

O critério para avaliar a similaridade de G e com uma árvore \mathcal{T} é conhecido como **largura arbórea** ou **largura em árvore** definida como:

$$tw(G) = \min\{w(\mathcal{T}) : \mathcal{T} \text{ é uma decomposição arbórea de } G\}.$$

Em outras palavras, a largura arbórea de um grafo G é a largura mínima dentre todas as decomposições possíveis de G . Via de regra, quanto menor a largura arbórea de um grafo, mais similar ele é a uma árvore. No Exemplo 3.1, podemos observar que as decomposições obtidas satisfazem $w(\mathcal{T}) = 2$ e $w(\mathcal{T}) = 6$, respectivamente. Entretanto, como veremos a seguir, no exemplo em questão, $tw(G) = 2$, ou seja, não existe uma decomposição arbórea em que a maior bolsa tenha no máximo dois vértices.

Teorema 3.1. *Um grafo conexo G com pelo menos dois vértices tem $tw(G) = 1$ se, e somente se, for uma árvore.*

Demonstração. Seja \mathcal{T} uma decomposição arbórea de G com $\mathcal{V} = \{1, \dots, m\}$ seu conjunto de nós. Vamos mostrar que, se $tw(G) = 1$, então G é uma árvore. A prova será por indução no número de nós de \mathcal{T} . Vamos supor que todos os nós de \mathcal{T} possuem bolsas com dois vértices. Caso isso não aconteça podemos adicionar algum vértice de um dos seus vizinhos na bolsa. Além disso, suporemos que nós adjacentes sempre possuirão bolsas diferentes, caso contrário poderíamos contrair nós com bolsas iguais em um único nó mantendo a propriedade de ser uma decomposição arbórea que representa o grafo G . Se $|\mathcal{V}| = 1$, G é uma aresta e então G é uma árvore. Para o passo de indução, suponha que j seja uma folha de \mathcal{T} tal que $B_j = \{u, v\}$ e com pai i . Seja $\mathcal{T}' = \mathcal{T} \setminus j$ e $G(\mathcal{T}')$ o subgrafo induzido pelos vértices contido nas bolsas de

\mathcal{T}' . Ao removermos uma folha da decomposição arbórea de G , estamos removendo uma bolsa que contém apenas dois vértices, um dos quais não aparece em outras bolsas e portanto é um vértice de grau um em G . Portanto, ao removermos uma bolsa que contém uma folha, estamos removendo uma aresta que conecta uma folha em G . Assim, ao removermos uma folha da decomposição de \mathcal{T} , a árvore subjacente $G(\mathcal{T}')$ permanece conexa. Podemos então criar um nó que contém u, v em qualquer nó da decomposição para obtermos a decomposição desejada.

Para condição necessária vamos provar por indução no número de vértices da árvore G . Se u e v são os únicos vértices da árvore então a decomposição em árvore que contém uma única bolsa contendo u e v é uma decomposição em árvore de largura 1. Dado $k \geq 3$, suponha que o resultado seja verdadeiro para árvores com $|V| < k$. Seja T uma árvore com k vértices e seja v uma folha e u o único vizinho de v . Consideramos a árvore $T' = T - v$. Como $|V(T')| < k$, portanto T' possui uma decomposição arbórea \mathcal{T}' de T' com $\text{tw}(T') = 1$. Agora, construa uma bolsa B_i que contém u e v de forma que seja conectado um novo nó i em qualquer nó que as bolsas contém u em \mathcal{T}' para produzir \mathcal{T} . A nova decomposição arbórea \mathcal{T} é uma decomposição de T com largura $\text{tw}(T) = 1$. \square

Lema 3.2. *Para todo grafo G com $\text{tw}(G) = k$ existe v tal que $d_G(v) \leq k$.*

Demonstração. Considere G um grafo e \mathcal{T} uma decomposição arbórea de G com largura k . Seja ℓ uma folha de \mathcal{T} . Se a bolsa de algum nó i estiver contida na bolsa de um de seus vizinhos j , removemos i e ligamos diretamente aos demais vizinhos de i a j mantendo a propriedade de ser uma decomposição arbórea de G . Agora, a bolsa B_ℓ inclui ao menos um vértice v que não está em nenhuma bolsa vizinha de \mathcal{T} , e portanto em nenhuma outra bolsa. Como $|B_\ell| \leq k + 1$, v tem grau no máximo k em G . \square

Teorema 3.3. *Um grafo conexo G com n vértices tem $\text{tw}(G) = n - 1$ se, e somente se, $G = K_n$.*

Demonstração. Vamos provar a condição necessária e suficiente por contraposição. Se G não é grafo completo, existem dois vértices $u, v \in V(G)$ tais que $\{u, v\} \notin E(G)$. Neste caso podemos construir uma decomposição arbórea com dois nós 1 e 2, tal que $B_1 = V \setminus \{u\}$ e $B_2 = B \setminus \{v\}$. Dessa forma, \mathcal{T} possui $\text{tw}(G) \leq n - 2$.

Por outro lado, se G possui uma decomposição arbórea \mathcal{T} com $\text{tw}(G) < n - 1$, pelo Lema 3.2, existe um vértice que tem grau no máximo $n - 2$ e portanto G não pode ser uma clique. \square

Lema 3.4. *Se $K \subseteq V$ induz uma clique em G , isto é, K induzido por G é grafo um completo, e \mathcal{T} é uma decomposição arbórea de G , então existe um nó i tal que $K \subseteq B_i$.*

Demonstração. Vamos provar o lema por indução no número de vértices da clique. Se $|K| \in \{1, 2\}$ o lema é verdadeiro pela definição da decomposição arbórea de G . Vamos supor que o lema vale para toda clique que satisfaz $|K| < k$ e provaremos que é válido para $|K| = k$. Seja K uma clique de G com k vértices e $v \in K$. Seja uma decomposição arbórea \mathcal{T} de G . Por hipótese de indução, existe um nó i em \mathcal{T} tal que $K - v \subseteq B_i$. Tome a subárvore de \mathcal{T}_v de \mathcal{T} formada pelos vértices que contêm v em suas bolsas e seja I o conjunto de índices desses nós. Seja $j \in I$ o índice que minimiza a distância de B_i e B_j . Note que tomando um vértice arbitrário $w \in K - v$, existe um nó $j' \in I$ tal que $\{v, w\} \in B_{j'}$ em \mathcal{T}_v e consequentemente em \mathcal{T} , por definição. Mas como o subgrafo de \mathcal{T} induzido pelos nós cuja bolsa contém w é conexo, a bolsa $B_{j'}$ está em um índice igual a j ou após ele pela condição de minimizar a distância de B_i e B_j , logo $w \in B_j$ e $K \subseteq B_j$, como gostaríamos. \square

Lema 3.5. *Seja $G = (V, E)$ um grafo e H um menor de G . Então $\text{tw}(H) \leq \text{tw}(G)$.*

Demonstração. Por definição, H pode ser obtido de G por uma série de contrações, retiradas de vértice ou de arestas. Se H é obtido através de uma sequência de remoção de arestas, a decomposição arbórea \mathcal{T} que representa o grafo G também representa

H , logo $tw(H) \leq tw(G)$. Caso H seja obtido através de uma sequência de retiradas de vértices, tomamos uma decomposição arbórea \mathcal{T} de G com a menor largura e removemos das bolsas de \mathcal{T} os vértices em $V(G) \setminus V(H)$. Este processo não pode aumentar a largura. Agora, seja uma decomposição arbórea \mathcal{T} de G e $e = \{x, y\}$ uma aresta de G a ser contraída como operação para obter H e seja v_e o novo vértice adicionado. Para cada bolsa de \mathcal{T} que contém x ou y , retiramos os vértices x e y da bolsa e incluímos o vértice v_e para obter uma decomposição arbórea para H . Note que o tamanho das bolsas permanece inalterado, a menos das bolsas que contêm os vértices x e y , que diminuem uma unidade. \square

Lema 3.6. *Se G é um grafo com componentes G_1, \dots, G_ℓ , então*

$$tw(G) = \max\{tw(G_1), \dots, tw(G_\ell)\}.$$

Demonstração. O lema anterior implica que $\max\{tw(G_1), \dots, tw(G_\ell)\} \leq tw(G)$. Reciprocamente, podemos produzir uma decomposição arbórea de G conectando as decomposições arbóreas disjuntas dos grafos G_k a um nó novo com bolsa vazia. \square

O conceito de decomposição em árvore foi um passo crucial para a prova do **Teorema de Robertson-Seymour (Graph Minor Theorem)**. A prova original foi apresentada ao longo de mais de 20 artigos abrangendo mais de 500 páginas de 1983 a 2004 (veja [31] e [33], por exemplo).

Teorema 3.7. (Teorema de Robertson-Seymour) *Para toda família \mathcal{F} de grafos que é fechada sob menores, existe um conjunto finito $\mathcal{P}_{\mathcal{F}}$ tal que $G \in \mathcal{F}$ se, e somente se, nenhum elemento de $\mathcal{P}_{\mathcal{F}}$ é menor de G .*

O Teorema de Wagner estabelece que os grafos planares são aqueles que não contêm o grafo completo K_5 ou o grafo bipartido completo $K_{3,3}$ como menores, ou seja, $\{K_5, K_{3,3}\}$ é uma família de menores proibidos para grafos planares. Esses grafos também são ditos grafos obstrução para propriedade de planaridade.

Seja \mathcal{F} a classe de grafos que são florestas, ou seja, não possuem ciclos. É fácil observar que essa classe é fechada em relação à tomada de menores, uma vez que todo menor de uma floresta é ainda uma floresta. Dessa forma, um grafo $G \in \mathcal{F}$ se, e somente se, G não possui ciclo como subgrafo, ou alternativamente, não possui ciclo como menor.

Para determinar os grafos obstrução para \mathcal{F} , precisamos encontrar os grafos minimais que não pertencem a \mathcal{F} , ou seja, os grafos minimais que possuem ciclo. Um exemplo clássico de grafo com ciclo é o C_n , ciclo de tamanho $n \geq 3$. Portanto, podemos dizer que C_n é um menor proibido para \mathcal{F} .

Encontrar todos os grafos de obstrução para uma classe de grafos pode ser uma tarefa árdua, pois a quantidade de grafos proibidos pode aumentar rapidamente. No entanto, para algumas larguras específicas, é possível determinar com precisão quais são esses grafos obstrução. Esse conhecimento permite uma análise mais precisa e detalhada das propriedades da classe de grafos em questão. Os lemas apresentados a seguir são fundamentais para a compreensão das classes de grafos com largura arbórea limitada. Eles estabelecem uma relação direta entre a largura arbórea desses grafos e a existência de um conjunto finito de grafos proibidos, que não podem ser menores de nenhum grafo pertencente à classe. Em outras palavras, esses lemas mostram que as classes de grafos com largura arbórea limitada são fechadas em relação aos menores proibidos especificados.

Lema 3.8. *A classe dos grafos com largura arbórea no máximo 1 é exatamente a classe dos grafos que tem K_3 como menor proibido.*

Lema 3.9. *A classe dos grafos com largura em árvore no máximo 2 é exatamente a classe de grafos que tem K_4 como menor proibido.*

Lema 3.10. *A classe dos grafos com largura arbórea no máximo 3 é exatamente a classe de grafos que tem K_5 , e os grafos da Figura 3.3, como menores proibidos.*

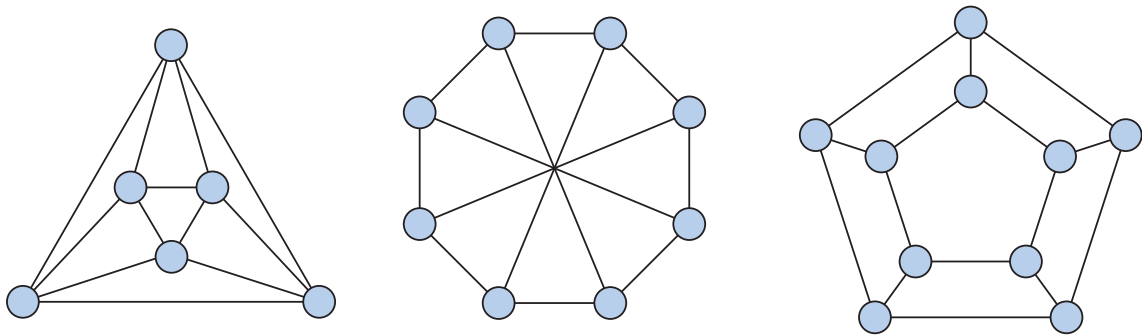


Figura 3.3: Grafo octaedro, de Wagner e prisma pentagonal.

Ao contrário das classes de grafos com largura arbórea no máximo $k \in \{1, 2\}$, que podem ser caracterizadas proibindo-se um único grafo menor, a determinação das classes de grafos com largura arbórea máxima para $k > 3$ é uma tarefa mais complexa. Para o caso $k = 3$, a classe de grafos com largura arbórea no máximo 3 é caracterizada pela proibição do grafo K_5 e de outros grafos menores. Neste contexto, é interessante destacar que a proibição do grafo K_5 é uma condição necessária, mas não suficiente, para que o grafo tenha largura arbórea no máximo 3. A prova deste resultado e outros lemas relacionados podem ser encontrados em detalhes em [1].

Neste trabalho, utilizaremos uma decomposição arbórea especial chamada de **Decomposição Agradável**, do inglês, *nice tree decomposition*, introduzida por Kloks [22]. Essa decomposição é uma árvore enraizada que possui quatro tipos diferentes de nós, a saber: folha, introdução, esquecimento e junção.

Um nó folha representa um conjunto inicial de vértices, enquanto um nó introdução representa a adição de um único vértice ao conjunto de vértices que já estão representados pelo filho do nó introdução. O nó esquecimento, por sua vez, é o inverso do nó introdução, isto é, remove um único vértice do conjunto de vértices representados pelo filho do nó esquecimento. Por fim, o nó junção possui exatamente dois filhos e representa a união dos conjuntos de vértices representados pelos seus

filhos. Definimos esse conceito formalmente e simbolicamente da seguinte maneira:

- a) **(Folha)** O nó i é uma folha de \mathcal{T} .
- b) **(Introdução)** O nó i introduz o vértice v , isto é, ele tem um filho único j , $v \notin B_j$ e $B_i = B_j \cup \{v\}$.
- c) **(Esquecimento)** O nó i esquece o vértice v , isto é, ele tem um filho único j , $v \notin B_j$ e $B_i = B_j \setminus \{v\}$.
- d) **(Junção)** O nó i é junção, isto é, ele possui exatamente dois filhos j e ℓ , e $B_i = B_j = B_\ell$.

Neste trabalho, requeremos que a decomposição agradável do grafo seja enraizada em um vértice com bolsa vazia. Caso a decomposição não possua essa propriedade, podemos transformá-la em uma decomposição com bolsa vazia adicionando um caminho de, no máximo, $k + 1$ nós, onde todos os nós são do tipo esquecimento. Ademais, a propriedade (3) da decomposição arbórea implica que um vértice pode ser esquecido apenas uma vez.

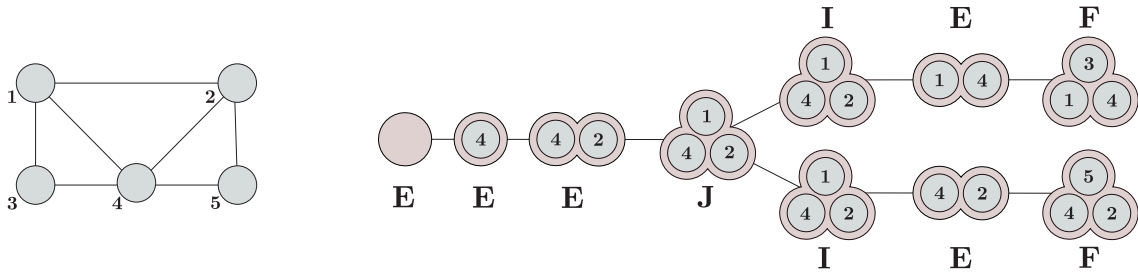


Figura 3.4: Um grafo G com 5 vértices e uma decomposição agradável do grafo. Os nós de esquecimento são representados por **E**, os nós de junção por **J**, os nós de introdução por **I**, e os nós folhas por **F**.

Podemos realizar a passagem de uma decomposição arbórea para uma decomposição agradável de maneira eficiente mantendo a largura do grafo. Apesar das exigências adicionais da decomposição, Kloks [22] demonstra em seu trabalho que podemos transformar eficientemente uma decomposição arbórea \mathcal{T} de largura k e m nós em uma decomposição agradável de largura k e no máximo $4m$ nós. Para

que possamos usar essa decomposição agradável em nossos estudos, é importante compreendermos como isso pode ser feito. O lema a seguir foi extraído de [3].

Lema 3.11. *Seja $G = (V, E)$ com uma decomposição arbórea \mathcal{T} de largura k , então G possui uma decomposição agradável com a mesma largura k .*

Demonstração. O procedimento percorre a árvore \mathcal{T} , das folhas à raiz r , em quatro passagens sucessivas. No primeira passagem, cada nó cuja bolsa está contida na bolsa do pai é removido e seus filhos são ligado ao seu avô.

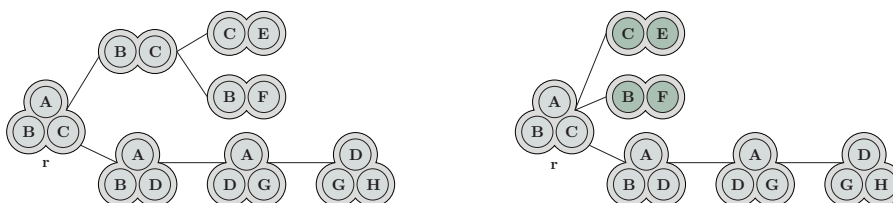


Figura 3.5: A primeira passagem remove nós cujas bolsas estão contidas nas bolsas dos seus pais.

No segundo passo, caso um nó possuir uma bolsa com uma quantidade menor de vértices do que seu pai, adicionamos alguns vértices da bolsa do pai até as bolsas possuírem o mesmo tamanho.

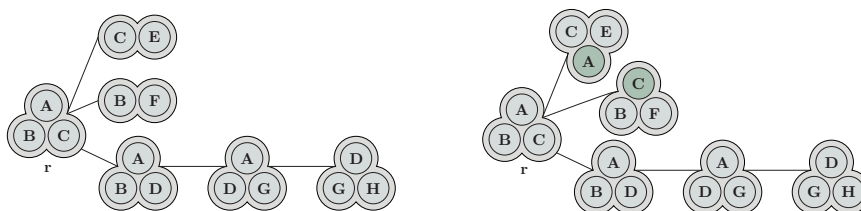


Figura 3.6: A segunda passagem equilibra a quantidade de vértices nas bolsas.

No terceiro passo, cada nó i que possui mais de $c \geq 2$ filhos é substituído por uma árvore binária que tem exatamente c folhas. Todos os nós novos recebem a bolsa B_i . Cada filho do nó original se torna um filho único de uma das folhas

da árvore binária recém-criada. Com isso, garantimos que todos os nós da árvore resultante possuem no máximo dois filhos, sendo que aqueles que possuem dois filhos são do tipo junção.

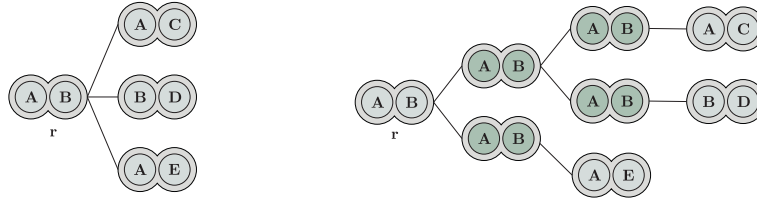


Figura 3.7: A terceira passagem substitui vértices de grau grande em uma árvore binária com nós do tipo junção.

Para qualquer nó i com um filho único j , se necessário, substituíamos a aresta $\{i, j\}$ por um caminho de forma que as bolsas se diferenciem exatamente por um vértice a etapa. Isso é feito de j à i por uma sequência de nós começando com um nó do tipo esquecimento e em seguida um nó do tipo introdução, alternando dessa forma, e possivelmente terminando com uma sequência de nós do tipo esquecimento caso a bolsa do filho seja maior que a de seu pai.

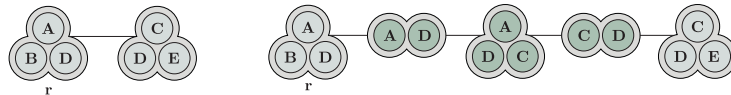


Figura 3.8: Na quarta passagem, adicionam-se nós alternantes para obter uma decomposição agradável.

Por fim, se a raiz r não tiver uma bolsa vazia, nós adicionamos um caminho na raiz onde cada nó é do tipo esquecimento até a bolsa ficar vazia. Note que o caminho pode ser realizado por no máximo $k + 1$ passos em virtude de $|B_i| \leq k + 1$.

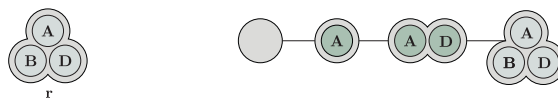


Figura 3.9: Passo 5: Esvaziando o último nó para obter uma bolsa vazia.

De fato \mathcal{T}' é uma decomposição agradável de G com a mesma largura k de \mathcal{T} . Para verificar que temos uma decomposição arbórea, note que as propriedades (1) e (3) são satisfeitas em cada passo. No entanto, o tamanho das bolsas foi modificado durante o passo 2, mas o tamanho de uma nova bolsa está limitada pelo tamanho de uma bolsa já existente na árvore. Para verificar (2), note que se $\{u, v\} \in E(G)$, há um nó i em \mathcal{T} tal que $u, v \in B_i$, e \mathcal{T}' contém a maior bolsa que continha o par u, v originalmente em \mathcal{T} . Portanto, \mathcal{T}' é uma decomposição arbórea e pela construção do procedimento anterior, \mathcal{T} é uma decomposição agradável de G de largura k . \square

De acordo com os passos do lema, a decomposição arbórea de largura 2 do Exemplo 3.1 é ilustrada na Figura 3.10.

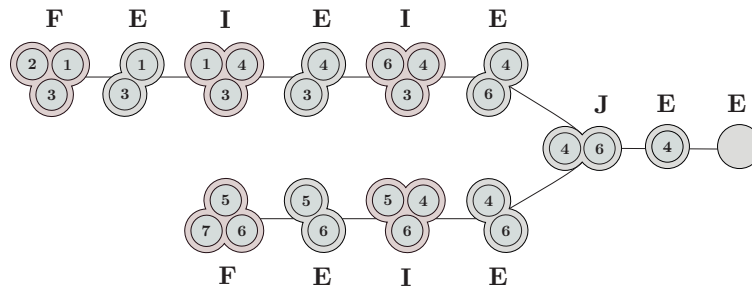


Figura 3.10: Decomposição agradável do exemplo 3.2.

No próximo capítulo, a decomposição agradável de um grafo será a base para a execução de um algoritmo que encontra uma matriz diagonal congruente a uma matriz simétrica do grafo. O algoritmo será denominado *Algoritmo Diagonal Arbórea* e servirá como base para o desenvolvimento dos capítulos posteriores.

4 DIAGONALIZAÇÃO DE MATRIZES SIMÉTRICAS DE GRAFOS UTILIZANDO DECOMPOSIÇÃO ARBÓREA

Neste capítulo apresentaremos o *Algoritmo Diagonal Arbórea*. Dada uma matriz simétrica M associada a um grafo $G = (V, E)$ com conjunto de vértices $V = \{1, \dots, n\}$, queremos encontrar uma matriz diagonal D congruente a M , utilizando uma decomposição arbórea agradável de G . O algoritmo proposto pode ser utilizado para localizar autovalores de qualquer matriz M simétrica de um grafo G . O algoritmo proposto permite obter a inércia de qualquer matriz simétrica M de um grafo G , observando o número de entradas positivas, negativas e nulas da diagonal de D . Vamos supor que uma decomposição agradável do grafo G seja parte da entrada do algoritmo e uma matriz M simétrica associada ao grafo. O algoritmo funciona com um ordenamento das folhas em direção à raiz da árvore, desse modo um nó i só é processado se seu filho já tiver sido. A entrada do algoritmo é composta por uma matriz $M(G)$ que representa um grafo G e uma decomposição arbórea \mathcal{T} correspondente. A seguir é exibido o pseudocódigo do algoritmo que depende de procedimentos específicos para cada tipo de nó, os quais serão descritos nas próximas páginas:

Algoritmo 1: Diagonal Arbórea ($M(G), \mathcal{T}$)

Entrada: Decomposição agradável \mathcal{T} e uma matriz $M(G)$

Saída: Matriz D diagonal congruente a M

- 1 Ordene os nós de \mathcal{T} com $1, \dots, m$ de modo que se i é pai de j então $j < i$
 - 2 **para** $i = 1$ **até** m **faça**
 - 3 **se** i é *folha* **então** Caixa de Folhas(B_i) **(Alg. 2)**
 - 4 **se** i é *introdução* **então** Caixa de Introdução(B_i) **(Alg. 3)**
 - 5 **se** i é *esquecimento* **então** Caixa de Esquecimento(B_i) **(Alg. 4)**
 - 6 **se** i é *junção* **então** Caixa de Junção(B_i) **(Alg. 5)**
-

O objetivo do algoritmo em cada nó i é produzir uma estrutura que chamaremos de caixa e denotaremos por N_i . Cada *caixa* é uma matriz formada por linhas e colunas associadas à matriz de entrada M . Cada vértice de G corresponde a uma linha de M , e, portanto, sempre que nos referimos a um vértice $v \in V$, estamos nos referindo a uma linha ou coluna de M . Se o nó é uma folha, o algoritmo inicializa uma caixa para o vértice correspondente, caso contrário, produz uma nova caixa baseada nas caixas transmitidas pelos seus filhos e em informações extraídas de M . Em cada etapa, o algoritmo também pode produzir elementos diagonais de uma matriz congruente a M . Os elementos diagonais são armazenados na forma (v, α) , onde v refere-se à linha e coluna correspondentes, e α representa o valor da entrada. Esses elementos diagonais não são transmitidos de uma caixa para outra, mas são armazenados para produzir D ao final do processo. Ao final de cada etapa a caixa a ser transmitida para o seu pai deve ter a seguinte aparência:

$$N_i = \left[\begin{array}{c|c} \begin{array}{c} 0 \\ \hline 0 \end{array} & \begin{array}{c} \text{blue triangle} \\ \hline 0 \end{array} \\ \hline \begin{array}{c} \text{blue triangle} \\ \hline 0 \end{array} & \begin{array}{c} * \\ \hline * \\ * \\ * \\ * \\ * \\ * \end{array} \end{array} \right] = \left[\begin{array}{c|c} N_i^{(0)} & N_i^{(1)} \\ \hline N_i^{(1)\top} & N_i^{(2)} \end{array} \right]$$

Figura 4.1: Modelo de caixa produzida pelo algoritmo.

Cada submatriz da Figura 4.1 possui uma estrutura bem definida. $N_i^{(0)}$ é uma matriz de dimensões $k'_i \times k'_i$ composta somente por zeros, $N_i^{(2)}$ é uma matriz simétrica de ordem $k''_i \times k''_i$ e $N_i^{(1)}$ é uma matriz de dimensões $k'_i \times k''_i$ na forma escalonada. Isso significa que todas as linhas não nulas aparecem antes das linhas nulas e, na mesma ordem, em todas as colunas. Além disso, para qualquer coluna que contenha um elemento não nulo, todos os elementos acima dele também são não nulos.

É importante notar que, ao final de cada etapa do algoritmo, a caixa a ser transmitida para o nó pai deve estar na forma da Figura 4.1, ou seja, ela deve

ser composta pela matriz $N_i^{(0)}$ na diagonal superior esquerda, pela matriz $N_i^{(2)}$ na diagonal inferior direita e pela matriz $N_i^{(1)}$ na parte superior direita. Essa estrutura é essencial para o bom funcionamento do algoritmo e garante que a matriz diagonal final seja congruente a matriz de entrada M .

Cada linha e coluna de N_i está associada a um vértice de G , de modo que $V(N_i)$ é o conjunto de vértices associados a essas linhas e colunas. Podemos classificar os vértices em dois tipos: vértices do tipo I (ou linhas do tipo I), que correspondem às k'_i primeiras linhas, e vértices do tipo II, que correspondem às k''_i linhas restantes. Os vértices do tipo I representam vértices temporários que estão esperando para serem diagonalizados. Eles são específicos dos nós em algum ramo da árvore enraizada que foram esquecidos no processo. Os vértices do tipo II são aqueles contidos na bolsa B_i correspondente de \mathcal{T} e que ainda não foram esquecidos em nenhum passo. As linhas $V_2(N_i)$ estão associadas aos vértices de B_i , ordenados por rótulo crescente. Definimos k'_i como o tamanho do conjunto $V_1(N_i)$ e k''_i como o tamanho de $V_2(N_i)$. É importante notar que $V(N_i) = V_1(N_i) \cup V_2(N_i)$ é uma partição dos vértices da caixa.

Iremos agora apresentar em detalhes o funcionamento de cada possível passo do algoritmo, começando quando o nó i é uma folha. Quando i é um nó de \mathcal{T} do tipo como **folha** com uma bolsa $|B_i| = b_i$ nós executamos o procedimento *Caixa de Folhas* (*Algoritmo 2*), que simplesmente constrói uma matriz nula de dimensão b_i .

Algoritmo 2: Caixa de Folhas

Entrada: Bolsa B_i de tamanho b_i

Saída: Caixa Padrão N_i

1 Defina $N_i^{(1)} = \emptyset$

2 Defina $N_i^{(2)}$ uma matriz com entradas nulas de dimensão $b_i \times b_i$

O procedimento inicializa a caixa com $k' = 0$ e $k'' = b_i$ onde $N_i^{(2)} = \mathbf{0}_{k'' \times k''}$. As linhas da caixa são rotuladas pelos vértices que estão presentes na bolsa B_i e a criação desta caixa tem como objetivo a inicialização do algoritmo.

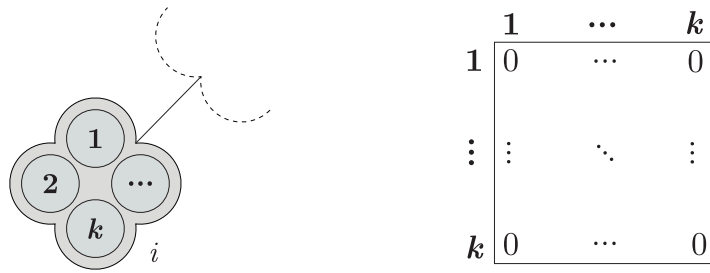


Figura 4.2: Nó folha com sua respectiva caixa.

Quando o nó i é um vértice do tipo **introdução** que insere v , ocorre $B_i = B_j \cup \{v\}$, onde j denota o filho de i e B_j não contém v . Para a construção da caixa N_i , executamos o algoritmo *Caixa de Introdução (Algoritmo 3)*, que utiliza como entrada o vértice adicionado v e a caixa transmitida pelo seu filho j que já foi produzida. Então o algoritmo adiciona uma linha e uma coluna de zeros rotuladas por v respeitando a ordem crescente dos rótulos das linhas $V_2(N_j)$ (vértices que pertencem a bolsa B_i), para produzir N_i .

Algoritmo 3: Caixa de Introdução

Entrada: Vértice v e caixa N_j do filho de i

Saída: Caixa Padrão N_i

- 1 Adicione uma linha e coluna de zeros associadas com v em N_j de forma que a linha adicionada esteja em ordem crescente com as linhas do tipo II.
-

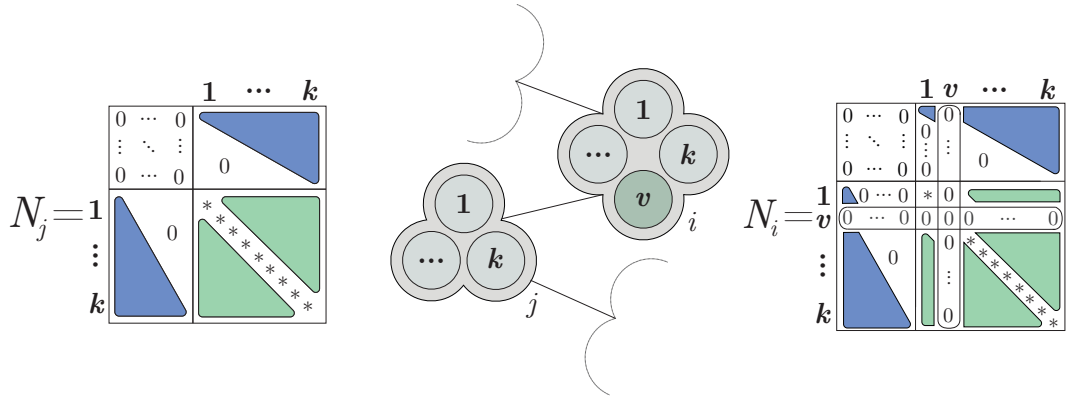


Figura 4.3: Nó do tipo introdução e sua respectiva caixa.

O procedimento *Caixa de Junção (Algoritmo 4)* é acionado quando i é um nó do tipo **junção**. Começamos o processo com as caixas N_j e N_ℓ já produzidas pelos filhos j e ℓ de i . Como $B_\ell = B_j$, temos que $V_2(N_\ell) = V_2(N_j)$. De acordo com o item (3) da definição de decomposição arbórea, um vértice esquecido em um ramo de \mathcal{T} não pode aparecer em outro ramo. Portanto, temos que $V_1(N_j) \cap V_1(N_\ell) = \emptyset$.

A operação de junção cria uma caixa intermediária chamada N_i^* . Intuitivamente, essa caixa será a soma dos blocos $N_j^{(2)}$ e $N_\ell^{(2)}$, que possuem os mesmos conjuntos de rótulos neste momento, e o empilhamento das matrizes $N_j^{(1)}$ e $N_\ell^{(1)}$, que têm rótulos diferentes. Os vértices da nova caixa são particionados em $V(N_i) = V_1(N_j) \cup V_1(N_\ell) \cup V_2(N_j)$, onde $|V_1(N_j)| = r$, $|V_1(N_\ell)| = s$, e $|V_2(N_j)| = |V_2(N_\ell)| = |B_i| = t$.

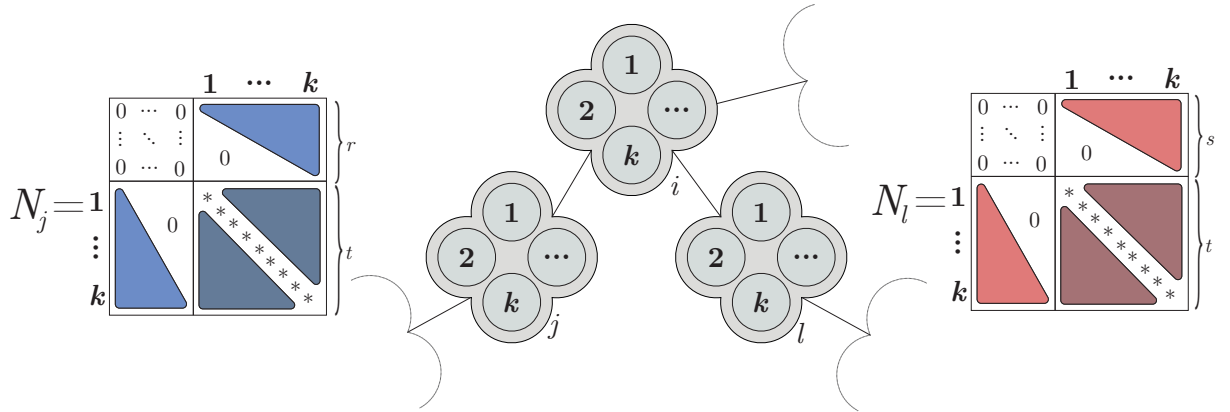


Figura 4.4: Caixas produzidas pelos filhos do nó junção.

A matriz N_i^* é definida com o seguinte formato: $N_i^{(0)}$ é uma matriz com entradas nulas de tamanho $(s+r) \times (s+r)$, $N_i^{(1)}$ é o empilhamento das submatrizes $N_j^{(1)}$ e $N_\ell^{(1)}$ resultando em uma matriz de tamanho $(r+s) \times t$, e $N_i^{(2)} = N_j^{(2)} + N_\ell^{(2)}$ é a soma das submatrizes $N_j^{(2)}$ e $N_\ell^{(2)}$, que possuem o mesmo conjunto de rótulos. Note que a operação de soma é realizada somando as entradas correspondentes das matrizes.

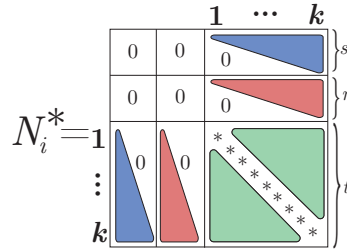


Figura 4.5: Caixa intermediária do nó junção.

Nosso objetivo é transformar N_i^* em uma caixa no formato padrão. Para isso, é necessário inserir as linhas de $V_1(N_\ell)$ na matriz $N_j^{(1)}$ para produzir uma única matriz na forma escalonada. Esse processo é necessário porque somente o bloco $N_i^{(1)}$ não está no formato adequado para transmitir a caixa N_i para seu pai. Para realizar essa transformação, enquanto houver um pivô α_c em uma coluna c de algum vértice $w \in V_1(N_j)$ na mesma coluna que o pivô β_c de alguma linha $v \in V_1(N_\ell)$, utilizamos

v para eliminar esse pivô por meio das seguintes operações elementares:

$$L_w \longleftarrow L_w - \frac{\alpha_c}{\beta_c} L_v \quad C_w \longleftarrow C_w - \frac{\alpha_c}{\beta_c} C_v. \quad (4.1)$$

A notação $L_w \longleftarrow L_w + \alpha L_v$ indica que a linha w é substituída pela linha w acrescida do valor α multiplicado pela linha v . De maneira análoga, a notação para as colunas segue o mesmo princípio.

Após não existir mais pivôs em posições iguais, a matriz $N_i^{(1)}$ é obtida juntando todas linhas que ainda tem pivô dentre $N_j^{(1)}$ e $N_\ell^{(1)}$ de modo que o resultado fique na forma escalonada.

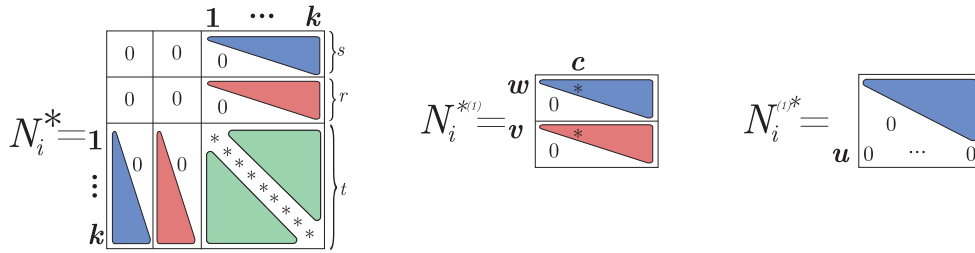


Figura 4.6: Passos intermediários para transformação de N_i^* para N_i .

Durante o processo de transformação de N_i^* em N_i , caso alguma linha se torne nula, o algoritmo armazena a linha correspondente a um vértice $u \in V_1(N_i)$ na forma $(u, 0)$ e remove essa linha da matriz para produzir N_i . Lembre-se de que a entrada relacionada ao vértice u é igual a zero. Dessa forma, ao final do processo obtemos uma caixa no padrão desejado.

Algoritmo 4: Caixa de Junção

Entrada: Caixas N_j e N_ℓ produzidas pelos filhos de i

Saída: Caixa Padrão N_i e possíveis pares diagonalizados

- 1 Defina uma matriz N_i^* como na figura 4.5
 - 2 Faça operações na submatriz $N_i^{*(1)}$ para reduzir à forma escalonada como em (4.1)
 - 3 Para cada linha nula $u \in N_i^{*(1)}$ armazene $(u, 0)$ e retire a linha/coluna associada para obter N_i
-

Se o vértice i é um nó do tipo **esquecimento**, executamos o processo *Caixa de Esquecimento* (Algoritmo 5). A construção da caixa N_i começa com uma caixa intermediária N_i^* , assim como é feito para os nós de junção. Inicialmente, resgatamos a caixa produzida pelo seu filho j e revelamos as adjacências entre o vértice esquecido v e os vértices que estão na bolsa B_i . Simbolicamente, para toda entrada uv , temos que $N_i[u, v] = N_j^{(2)}[u, v] + m_{uv}$ para todo $u \in B_j$, enquanto as outras entradas permanecem inalteradas.

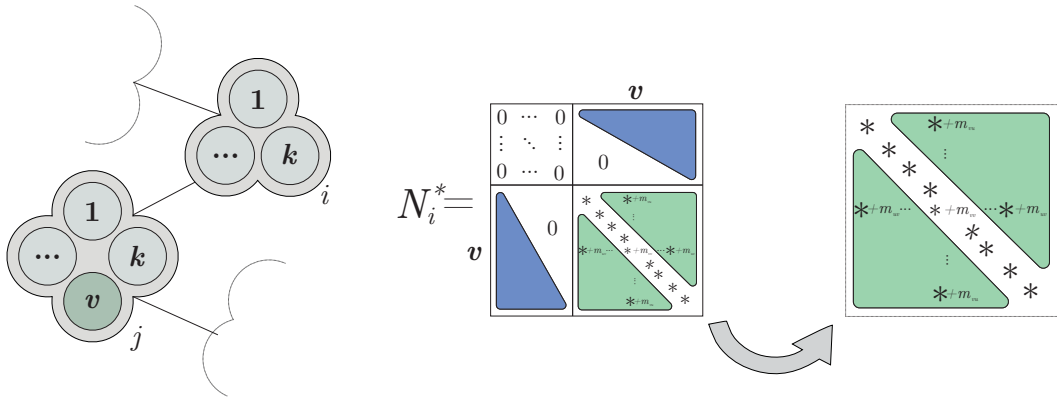


Figura 4.7: Passos intermediários para transformação de N_i^* para N_i .

A linha v é do tipo II na caixa N_j , portanto, após a introdução das entradas de M , v será vista como uma linha do tipo I na caixa intermediária N_i^* . Para facilitar a visualização, podemos permutar as linhas e colunas de N_i^* de modo que a linha associada a v ocupe a primeira posição. Assim, N_i^* se torna:

$$N_i^* = \begin{array}{c} \mathbf{1} \\ \vdots \\ \mathbf{k} \end{array} \begin{array}{c|cc} \mathbf{v} & \mathbf{1} & \cdots & \mathbf{k} \\ \hline \mathbf{v} & d_v & \mathbf{x}_v & \mathbf{y}_v \\ \mathbf{x}_v & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{1} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{k} & \vdots & \mathbf{y}_v & \vdots \end{array}$$

Figura 4.8: Permutação de linhas e colunas para v ocupar a primeira posição.

Agora, a primeira linha e coluna da caixa N_i^* estão associadas ao vértice a ser esquecido v . Denominamos por d_v o valor da entrada vv da caixa. O vetor \mathbf{x}_v relaciona o vértice v com os vértices que já foram esquecidos em etapas anteriores, ou seja, as entradas uv tal que $u \notin B_i$. Já o vetor \mathbf{y}_v relaciona v com as entradas uv tal que $u \in B_i$. Esses vetores serão úteis para facilitar a compreensão dos próximos passos e a execução dos processos.

Caso 1. \mathbf{x}_v é vazio ou $\mathbf{x}_v = [0 \dots 0]$.

Caso 1a. Se \mathbf{y}_v é vazio ou \mathbf{y}_v é um vetor formado de zeros, o algoritmo constata que a linha v já está diagonalizada e armazenamos o par (v, d_v) , e removemos a linha e coluna v produzindo de N_i^* a caixa N_i .

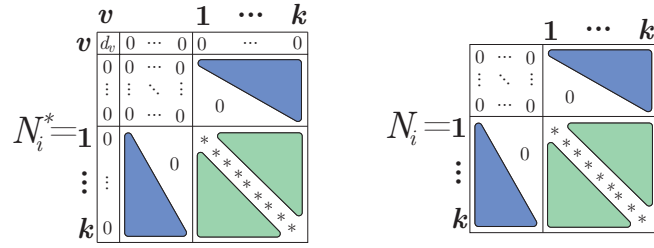


Figura 4.9: Caso 1a: Vetores nulos ou vazios.

Se $\mathbf{y}_v \neq [0 \dots 0]$ podemos ter dois casos possíveis de procedimentos:

Caso 1b. Se $d_v \neq 0$ usamos essa entrada para anular todas as entradas em \mathbf{y}_v e diagonalizamos a linha correspondente a v . Para cada elemento $u \in B_i$ realizamos as operações elementares

$$L_u \leftarrow L_u - \frac{\alpha_v}{d_v} L_v \quad C_u \leftarrow C_u - \frac{\alpha_v}{d_v} C_v.$$

Após anular todas as entradas em \mathbf{y}_v , adicionamos (v, d_v) e retiramos a linha/coluna v transformando N_i^* para N_i .

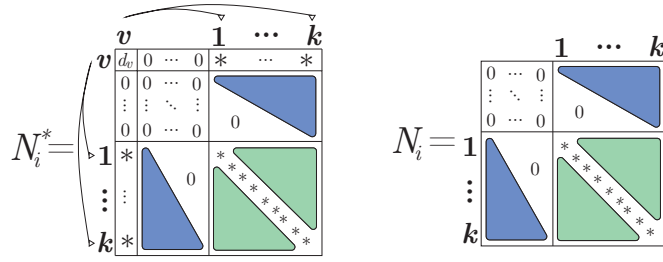


Figura 4.10: Caso 1b: $d_v \neq 0$ e $\mathbf{y}_v \neq [0 \dots 0]$.

Caso 1c. Quando $d_v = 0$, o objetivo é tornar a linha v do tipo I e deixar a caixa na forma padrão. Para isso, é necessário adicionar o vetor \mathbf{y}_v à submatriz $N_i^{(1)}$ para obter uma matriz escalonada. As linhas em $V(N_i^{(1)*})$ serão usadas para eliminar as entradas da linha associada a v .

Seja α_j o pivô do vetor \mathbf{y}_v para $j \in V(N_i^{(1)*})$. Se existir outro pivô β_j em alguma linha de $N_1^{*(1)}$, é utilizado para anular a entrada α_j realizando as operações:

$$L_v \leftarrow L_v - \frac{\alpha_j}{\beta_j} L_u \quad C_v \leftarrow C_v - \frac{\alpha_j}{\beta_j} C_u. \quad (4.2)$$

Esse processo é repetido até que não haja mais pivôs para anular posições na linha v , momento em que a linha v é inserida em $N_i^{(1)}$. Caso a linha associada a v se torne nula, armazenamos o par $(v, 0)$ e retiramos a linha v para produzir N_i .

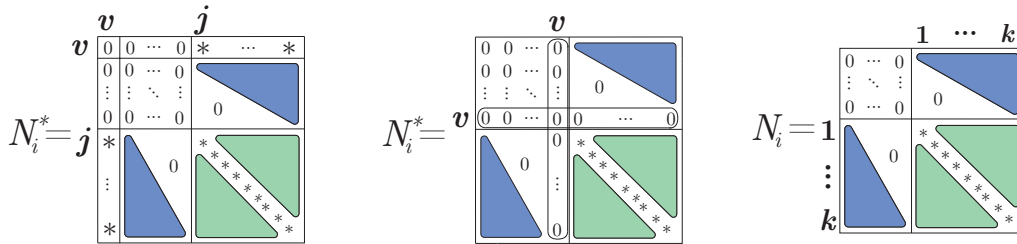


Figura 4.11: Caso 1c: $d_v = 0$ e $\mathbf{y}_v \neq [0 \dots 0]$.

Caso 2. Se $\mathbf{x}_v \neq \emptyset$ ou $\mathbf{x}_v = [0 \dots 0]$.

Seja j o vértice associado à última entrada não nula de \mathbf{x}_v . Seja α_j essa entrada. O algoritmo utiliza α_j para eliminar todas outras entradas não nulas em \mathbf{x}_v

da direita para a esquerda. Seja $w \in V_1(N_i^{*(1)})$ um vértice associado a uma entrada $\alpha_w \neq 0$. O algoritmo executa

$$L_w \leftarrow L_w - \frac{\alpha_j}{\alpha_w} L_j, \quad C_w \leftarrow C_w - \frac{\alpha_j}{\alpha_w} C_j. \quad (4.3)$$

Se $d_v \neq 0$ anulamos essa entrada realizando as operações:

$$L_v \leftarrow L_v - \frac{d_v}{2\alpha_j} L_j, \quad C_v \leftarrow C_v - \frac{d_v}{2\alpha_j} C_j. \quad (4.4)$$

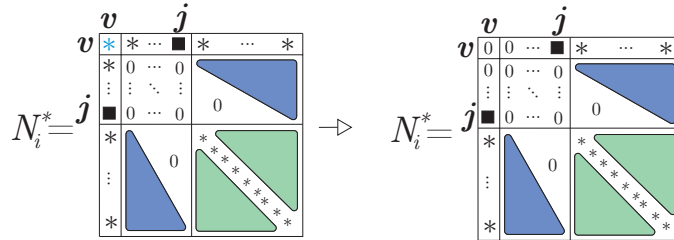


Figura 4.12: Passo intermediário para a execução do caso 2.

Neste ponto obtemos uma submatriz nas $(k' + 1) \times (k' + 1)$ primeiras entradas, na qual somente as entradas $N_i^*[v, j]$ e $N_i^*[j, v]$ não são nulas, ademais são α_j . Nosso objetivo é tornar as linhas associadas aos vértices v e j aptas para a diagonalização, isto é, desejamos substituir essas duas entradas por elementos não-nulos nas respectivas diagonais. Realiza as operações:

$$L_u \leftarrow L_u + \frac{1}{2} L_v, C_u \leftarrow C_u + \frac{1}{2} C_v, L_v \leftarrow L_v - L_u, C_v \leftarrow C_v - C_u \quad (4.5)$$

As entradas relevantes da submatriz são modificadas da seguinte forma:

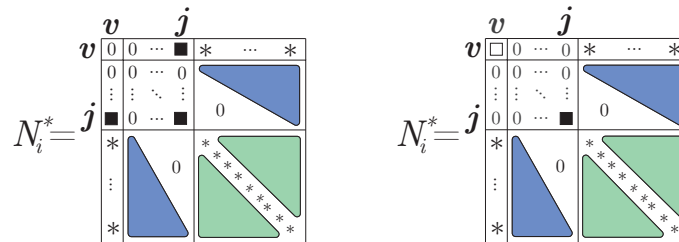


Figura 4.13: Tornando elementos aptos para diagonalização.

Neste ponto, estamos aptos a usar os elementos obtidos na diagonal para diagonalizar as linhas associadas a v e j , como no Caso 1 quando $\mathbf{x}_v = [0 \dots 0]$ e $d_v \neq 0$. Além disso, é fácil ver que os pares produzidos nessa etapa são justamente $(v, -\alpha_j)$ e (j, α_j) .

Algoritmo 5: Caixa de Esquecimento

Entrada: Caixa N_j , vértice esquecido v

Saída: Caixa Padrão N_i e Pares Diagonalizados

- 1 $N_i^{*(1)} = N_j^{(1)}$
- 2 $N_i^{*(2)}[u, w] = N_j^{(2)}[u, w], \forall u, w \in B_j$ com $v \notin \{u, w\}$
- 3 $N_i^{*(2)}[u, v] = N_i^{*(2)}[v, u] = N_j^{(2)}[u, v] + m_{uv}, \forall u \in B_j$
- 4 **se** x_v *é vazio ou 0* **então**
 - 5 **se** y_v *ou 0* **então** Caso 1a
 - 6 anexe o par $(v, 0)$ em D
 - 7 retire a linha v de N_i^* para formar N_i
 - 8 **senão**
 - 9 **se** $d_v \neq 0$ **então** Caso 1b
 - 10 use d_v para diagonalizar a linha/coluna v
 - 11 anexe o par (v, d_v) em D
 - 12 remova a linha v de N_i^* para formar N_i
 - 13 **senão** Caso 1c
 - 14 faça as operações de linha/coluna como na equação 4.2
 - 15 **se** *obteve uma linha nula* **então**
 - 16 anexe o par $(v, 0)$ em D
 - 17 retire a linha de v de N_i^* para formar N_i
 - 18 **senão**
 - 19 insira v em $N_i^{(1)}$
 - 20 **senão** Caso 2
 - 21 use as operações apresentadas nas equações 4.3, 4.4 e 4.5 para diagonalizar a linha j e v
 - 22 anexe (v, d_v) e (u, d_j) em D e remova linha j e v de N_i^* para produzir N_i

Para finalizar esta seção, apresentaremos um exemplo concreto detalhado para ilustrar o funcionamento do algoritmo.

Exemplo 4.1. Considere G o grafo descrito na Figura 3.2 e sua decomposição agradável da Figura 4.14 com a matriz associada

$$M = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} & \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 6 & 0 & -2 & 0 \\ 0 & 0 & 6 & -5 & -3 & 2 & 0 \\ 0 & 0 & 0 & -3 & -4 & -1 & 2 \\ 0 & 0 & -2 & 2 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 2 & 1 & -1 \end{bmatrix} \end{matrix}$$

Nosso objetivo é encontrar uma matriz diagonal D que é congruente a M usando \mathcal{T} como entrada. Note que a matriz M representa um grafo não direcionado G , onde os pesos são atribuídos às arestas e vértices. Para inicialização do algoritmo, devemos ordenar os nós de \mathcal{T} de modo que o pai de algum nó i só é processado quando seu filho j já tiver sido. Para isso, suponha que os nós de \mathcal{T} estejam rotulados da seguinte forma: os nós 1 a 6 estão no ramo superior em ordem crescente em direção da raiz, os nós 7 a 10 estão no ramo inferior, e os nós 11 a 13 partem da junção da bifurcação até a raiz da árvore.

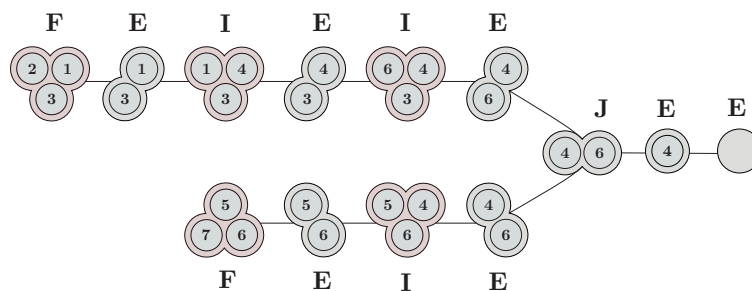
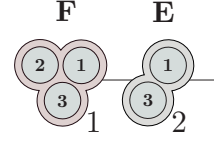


Figura 4.14: Decomposição agradável do exemplo 3.2.

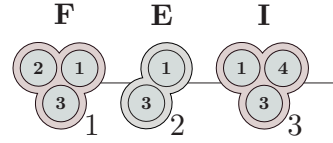
Quando $i = 1$ o nó é do tipo folha, chamando o procedimento *Caixa de Folhas*. Neste passo, o nó $i = 1$ possui bolsa $B_1 = \{1, 2, 3\}$ que produz um caixa $\mathbf{0}_{3 \times 3}$ com linhas rotuladas pelos vértices de B_1 .

$$N_1 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix}$$



A caixa N_1 não possui linhas do tipo I e possui três linhas do tipo II, isto é, $N_1^{(0)} = N_1^{(1)} = \emptyset$ e $N_1^{(2)} = N_1$. Os números representados acima e ao lado de cada caixa representam os respectivos rótulos dos vértices do grafo. Quando $i = 2$, o nó esquece o vértice $v = 2$ chamando o procedimento *Caixa de Esquecimento* e que recebe a caixa N_1 já produzida por seu filho 1. Nosso objetivo é produzir uma caixa auxiliar N_2^* que captura as entradas m_{22}, m_{i2}, m_{2j} , da matriz de entrada M onde $i, j \in \{1, 3\} = B_2$, ou seja, revelamos as adjacências entre o vértice esquecido e os vértices que estão na bolsa. Como o vértice $v = 2$ não ocupa a primeira posição no ordenamento dos vértices, trocamos as linhas e colunas obtendo:

$$N_2^* = N_2^{*(2)} = \begin{matrix} & \begin{matrix} 2 & 1 & 3 \end{matrix} \\ \begin{matrix} 2 \\ 1 \\ 3 \end{matrix} & \begin{bmatrix} -1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \end{matrix}$$



Na configuração obtida de N_2^* temos que o vetor \mathbf{x}_v é vazio, $\mathbf{y}_v = [1 \ 1] \neq [0 \ 0]$. Como $d_v = -1 \neq 0$ e \mathbf{x}_v é vazio estamos no Subcaso 1b do procedimento, e o algoritmo diagonaliza as linhas e colunas correspondentes a $v = 2$. Identificaremos por L_v e C_v a linha e coluna indexada pelo vértice v . Para isso, executamos as operação $L_1 \leftarrow L_1 + L_2$ seguida de $C_1 \leftarrow C_1 + C_2$ para anularmos a entrada da linha correspondente a $v = 1$ e as operações $L_3 \leftarrow L_3 + L_2$ seguida $C_1 \leftarrow C_3 + C_2$ para anular a posição $v = 3$. Isto leva a

$$N_2^* = \begin{matrix} & \begin{matrix} 2 & 1 & 3 \end{matrix} \\ \begin{matrix} 2 \\ 1 \\ 3 \end{matrix} & \begin{bmatrix} -1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{matrix}, \quad N_2^* = \begin{matrix} & \begin{matrix} 2 & 1 & 3 \end{matrix} \\ \begin{matrix} 2 \\ 1 \\ 3 \end{matrix} & \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \end{matrix}.$$

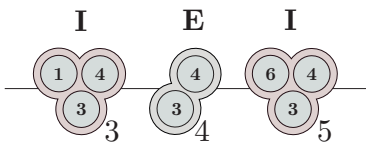
Dessa forma a linha e coluna correspondente ao vértice $v = 2$ está diagonalizada. O passo $i = 2$ armazena o par $(v, d_v) = (2, -1)$ para produzir D ao final do processo e transmite a caixa N_2 com $N_2^{(0)} = N_2^{(1)} = \emptyset$ e $N_2^{(2)} = N_2$.

$$N_2^* = \begin{array}{c} 2 \\ 1 \\ 3 \end{array} \begin{array}{c|cc} 2 & 1 & 3 \\ \hline -1 & 0 & 0 \\ \hline 0 & 1 & 1 \\ \hline 0 & 1 & 1 \end{array} \quad N_2 = \begin{array}{c} 1 \\ 3 \end{array} \begin{array}{c|c} 1 & 3 \\ \hline 1 & 1 \\ \hline 1 & 1 \end{array}.$$

Na iteração $i = 3$ é introduzido o vértice $v = 4$, produzindo uma caixa N_3 a partir de N_2 , onde $N_3^{(0)} = N_3^{(1)} = \emptyset$ e é adicionada uma linha repleta de zeros em ordem crescente com os demais vértices do tipo II.

$$N_3 = N_3^{(2)} = \begin{array}{c} 1 \\ 3 \\ 4 \end{array} \begin{array}{c|cc} 1 & 3 & 4 \\ \hline 1 & 1 & 0 \\ \hline 1 & 1 & 0 \\ \hline 0 & 0 & 0 \end{array}$$


No passo $i = 4$ temos um nó que esquece o vértice $v = 1$, portanto atualizamos as entradas m_{1i} e m_{1j} de forma que $i, j \in \{3, 4\}$ que são vértices que continuam na bolsa B_4 . A matriz auxiliar N_4^* é definida como

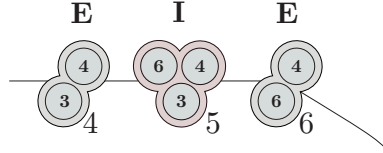
$$N_4^* = \begin{array}{c} 1 \\ 3 \\ 4 \end{array} \begin{array}{c|cc} 1 & 3 & 4 \\ \hline 1+1 & 1+1 & 0 \\ \hline 1+1 & 1 & 0 \\ \hline 0 & 0 & 0 \end{array} = \begin{array}{c} 1 \\ 3 \\ 4 \end{array} \begin{array}{c|cc} 1 & 3 & 4 \\ \hline 2 & 2 & 0 \\ \hline 2 & 1 & 0 \\ \hline 0 & 0 & 0 \end{array}$$


Como \mathbf{x}_v é vazio e $d_v = 2 \neq 0$ novamente estamos no Subcaso 1b, onde utilizaremos a d_v para aniquilar as entradas não nulas do vetor \mathbf{y}_v . Realizando a operação $L_3 \leftarrow L_3 - L_1$ seguida de $C_3 \leftarrow C_3 - C_1$ temos que

$$N_4^* = \begin{array}{c} 1 \\ 3 \\ 4 \end{array} \begin{array}{c|cc} 1 & 3 & 4 \\ \hline 2 & 0 & 0 \\ \hline 0 & -1 & 0 \\ 0 & 0 & 0 \end{array} \quad N_4 = \begin{array}{c} 3 \\ 4 \end{array} \begin{array}{c|c} 3 & 4 \\ \hline -1 & 0 \\ 0 & 0 \end{array}.$$

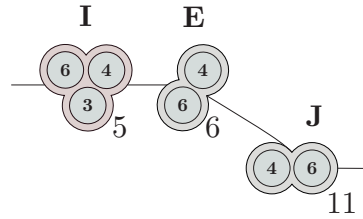
A linha correspondente ao vértice $v = 1$ está diagonalizada, produzindo o par $(v, d_v) = (1, 2)$. Por fim, retiramos a linha e coluna associadas a $v = 1$ para produzir N_5 , na qual $N_5^{(0)} = N_5^{(1)} = \emptyset$ e $N_5^{(2)} = N_5$. Para $i = 5$, é introduzido o vértice $v = 6$ e construímos a caixa N_5 com todas linhas do tipo II em ordem crescente.

$$N_5 = \begin{array}{c} 3 \\ 4 \\ 6 \end{array} \begin{array}{c|cc} 3 & 4 & 6 \\ \hline -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}$$



Quando $i = 6$ o vértice $v = 3$ é esquecido. Como realizado anteriormente, capturamos as entradas em M das adjacências entre o vértice $v = 3$ e os vértices que permanecem na bolsa. Nesta caixa, não há necessidade de trocar as linhas e colunas devido ao fato do vértice 3 já ocupar a primeira posição da caixa.

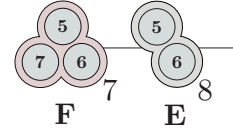
$$N_6^* = \begin{array}{c} 3 \\ 4 \\ 6 \end{array} \begin{array}{c|cc} 3 & 4 & 6 \\ \hline -1 + 1 & 6 & -2 \\ 6 & 0 & 0 \\ -2 & 0 & 0 \end{array} = \begin{array}{c} 3 \\ 4 \\ 6 \end{array} \begin{array}{c|cc} 3 & 4 & 6 \\ \hline 0 & 6 & -2 \\ 6 & 0 & 0 \\ -2 & 0 & 0 \end{array}$$



Note que $d_v = 0$, \mathbf{x}_v é vazio e $\mathbf{y}_v = [6 \ -2] \neq [0 \ 0]$. Estamos no subcaso 1c, nosso objetivo é tornar a linha v como uma linha do tipo I. Como não há nenhuma linha do tipo I em N_6^* , ela é simplesmente utilizada para produzir N_6 . Aqui temos que o $k'_6 = 1$ e $k''_6 = 2$, os respectivos tamanhos dos conjuntos das linhas do tipo I e tipo II.

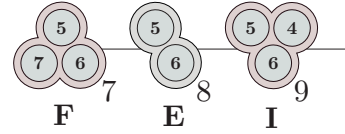
Agora quando $i = 7$ estamos em outro ramo da árvore \mathcal{T} . Como $i = 7$ é uma folha, uma caixa rotulada pelos vértices da bolsa com todas entradas nulas é criada como no início do processo.

$$N_7 = \begin{matrix} & \begin{matrix} 5 & 6 & 7 \end{matrix} \\ \begin{matrix} 5 \\ 6 \\ 7 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix}$$



O nó $i = 8$ esquece o vértice $v = 7$. Remanejamos as linhas de forma que v ocupe a primeira posição e atualizamos as entradas entre v e os vértices que estão na bolsa B_8 . Obtemos a matriz auxiliar

$$N_8^* = \begin{matrix} & \begin{matrix} 7 & 5 & 6 \end{matrix} \\ \begin{matrix} 7 \\ 5 \\ 6 \end{matrix} & \begin{bmatrix} -1 & 2 & 1 \\ 2 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \end{matrix}$$

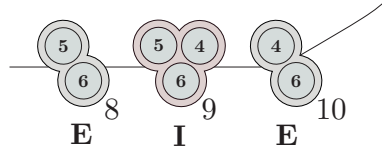


Como $d_v = -1 \neq 0$ e \mathbf{x}_v é vazio, estamos novamente no subcaso 1b onde usamos d_v para eliminar as posições do vetor \mathbf{y}_v . Realizando a sequência de operações $L_5 \leftarrow L_5 + 2L_7$, $C_5 \leftarrow C_5 + 2C_7$, $L_6 \leftarrow L_6 + L_7$, $C_6 \leftarrow C_6 + C_7$, obtemos a diagonalização de $v = 7$.

$$N_8^* = \begin{matrix} & \begin{matrix} 7 & 5 & 6 \end{matrix} \\ \begin{matrix} 7 \\ 5 \\ 6 \end{matrix} & \begin{bmatrix} -1 & 0 & 0 \\ 0 & 4 & 2 \\ 0 & 2 & 1 \end{bmatrix} \end{matrix} \quad N_8 = \begin{matrix} & \begin{matrix} 5 & 6 \end{matrix} \\ \begin{matrix} 5 \\ 6 \end{matrix} & \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \end{matrix}$$

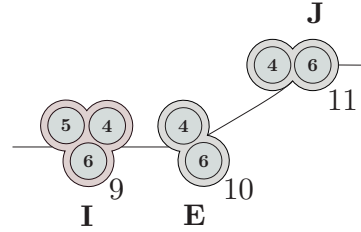
Nesta etapa, o algoritmo armazena o par $(7, -1)$ e retira a linha e a coluna relacionadas a v para produzir N_8 . Quando $i = 9$, introduzimos o vértice $v = 4$ para produzir

$$N_9 = \begin{array}{c} 4 \\ 5 \\ 6 \end{array} \begin{array}{c|cc} 4 & 5 & 6 \\ \hline 0 & 0 & 0 \\ \hline 0 & 4 & 2 \\ \hline 0 & 2 & 1 \end{array}$$



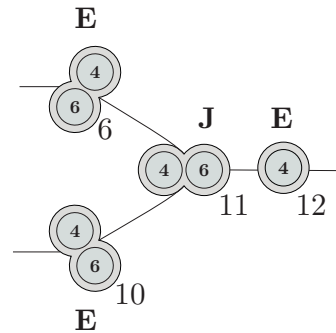
No nó $i = 10$, o vértice $v = 5$ é esquecido. Resgatando as adjacências entre v e os vértice na bolsa B_{10} obtemos a caixa auxiliar

$$N_{10}^* = \begin{array}{c} 5 \\ 4 \\ 6 \end{array} \begin{array}{c|cc} 5 & 4 & 6 \\ \hline 4-4 & -3 & 2-1 \\ \hline -3 & 0 & 0 \\ \hline 2-1 & 0 & 1 \end{array} = \begin{array}{c} 5 \\ 4 \\ 6 \end{array} \begin{array}{c|cc} 5 & 4 & 6 \\ \hline 0 & -3 & 1 \\ \hline -3 & 0 & 0 \\ \hline 1 & 0 & 1 \end{array}$$



Como N_{10}^* não tem linhas do tipo I e $d_v = 0$, estamos no Subcaso 1c e nosso objetivo é tornar v como uma linha do tipo I. Para isso, devemos deixar a submatriz $N_{10}^{*(1)}$ na forma escalonada, contudo N_{10}^* já está no formato requerido. Portanto temos que $N_{10}^* = N_{10}$ é a caixa transmitida para seu pai. O nó $i = 11$ é do tipo junção. Nós chamamos o processo *Caixa de Junção* que utiliza como entrada as caixas já produzidas pelos seus filhos N_6 e N_{10} . O algoritmo constrói uma caixa N_{11}^* auxiliar onde as linhas do tipo I vem de diferentes ramos de \mathcal{T} e as matrizes na forma escalonada dos filhos $N_6^{(1)}$ e $N_{10}^{(1)}$ ficam uma ao topo da outra, e as linhas do tipo II são somadas em suas respectivas posições. Dessa forma a caixa N_{11}^* se torna

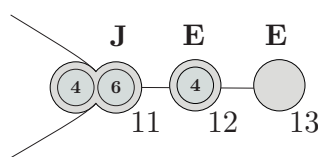
$$N_{11}^* = \begin{array}{c} 3 \\ 5 \\ 4 \\ 6 \end{array} \begin{array}{c|cc|cc} 3 & 5 & 4 & 6 \\ \hline 0 & 0 & 6 & -2 \\ \hline 0 & 0 & -3 & 1 \\ \hline 6 & -3 & 0 & 0 \\ \hline -2 & 1 & 0 & 1 \end{array}$$



Nosso objetivo é tornar N_{11}^* em uma caixa no formato padrão para transmitir para seu pai. Para isso, enquanto existir dois pivôs na mesma coluna de alguma linhas do tipo I , usamos o primeiro pivô para deixar a submatriz $N_{11}^{*(1)}$ na forma escalonada. Sendo assim, realizando as operações $L_5 \leftarrow L_5 + 1/2L_3$ seguida por $C_5 \leftarrow C_5 + 1/2C_3$ a caixa N_{11}^* se torna

$$N_{11}^* = \begin{array}{c} 3 \\ 5 \\ 4 \\ 6 \end{array} \begin{array}{c} 3 \quad 5 \quad 4 \quad 6 \\ \left[\begin{array}{c|c|c|c} 0 & 0 & 6 & -2 \\ \hline 0 & 0 & 0 & 0 \\ \hline 6 & 0 & 0 & 0 \\ \hline -2 & 0 & 0 & 1 \end{array} \right] \end{array} \quad N_{11} = \begin{array}{c} 3 \\ 4 \\ 6 \end{array} \begin{array}{c} 3 \quad 4 \quad 6 \\ \left[\begin{array}{c|c|c} 0 & 6 & -2 \\ \hline 6 & 0 & 0 \\ \hline -2 & 0 & 1 \end{array} \right] \end{array}.$$

Ao realizar as operações obtemos uma linha de zero do tipo I na caixa. O algoritmo adiciona o par $(5, 0)$ na lista de elementos diagonalizados para formar D e retira a linha e a coluna respectiva da caixa N_{11}^* para formar N_{11} . O nó $i = 12$ esquece o vértice $v = 6$. Como nos casos anteriores trocamos as linhas para visualizar o vértice 6 na primeira posição e resgatamos as adjacências de $v = 6$ e o vértice $w = 4$. A caixa auxiliar N_{12} é construída como

$$N_{12}^* = \begin{array}{c} 6 \\ 3 \\ 4 \end{array} \begin{array}{c} 6 \quad 3 \quad 4 \\ \left[\begin{array}{c|c|c} 1+1 & -2 & 2 \\ \hline -2 & 0 & 6 \\ \hline 2 & 6 & 0 \end{array} \right] \end{array} = \begin{array}{c} 6 \\ 3 \\ 4 \end{array} \begin{array}{c} 6 \quad 3 \quad 4 \\ \left[\begin{array}{c|c|c} 2 & -2 & 2 \\ \hline -2 & 0 & 6 \\ \hline 2 & 6 & 0 \end{array} \right] \end{array}$$


Como $\mathbf{x}_v = [-2] \neq []$ estamos no subcaso 2. Como $d_v = 2 \neq 0$ o primeiro passo a ser realizado é tornar essa entrada nula. Utilizando o vértice mais a direita de \mathbf{x}_v , realizamos as operações $L_6 \leftarrow L_6 + 1/2L_3$ e $C_6 \leftarrow C_6 + 1/2C_3$ obtemos a matriz auxiliar N_{12}^* da forma

$$N_{12}^* = \begin{array}{c} 6 \\ 3 \\ 4 \end{array} \begin{array}{c} 6 \quad 3 \quad 4 \\ \left[\begin{array}{c|c|c} 0 & -2 & 5 \\ \hline -2 & 0 & 6 \\ \hline 5 & 6 & 0 \end{array} \right] \end{array}.$$

Nosso próximo passo é tornar a submatriz rotulado pelos vértice $\{3, 6\}$ com elementos não nulos na diagonal. Para isso vamos executar as operações $L_3 \leftarrow L_3 + 1/2L_6$, $C_3 \leftarrow C_3 + 1/2C_6$, $L_6 \leftarrow L_6 - L_3$ e $C_6 \leftarrow C_6 - C_3$ obtemos o formato desejado:

$$N_{12}^* = \begin{array}{c} \begin{array}{ccc} 6 & 3 & 4 \\ \hline 0 & -2 & 5 \\ \hline -2 & -2 & 17/2 \\ \hline 5 & 17/2 & 0 \end{array} \\ \begin{array}{c} 6 \\ 3 \\ 4 \end{array} \end{array} \quad N_{12}^* = \begin{array}{c} \begin{array}{ccc} 6 & 3 & 4 \\ \hline 2 & 0 & -7/2 \\ \hline 0 & -2 & 17/2 \\ \hline -7/2 & 17/2 & 0 \end{array} \\ \begin{array}{c} 6 \\ 3 \\ 4 \end{array} \end{array} .$$

Agora podemos utilizar os pivôs não nulos dos vértice $v = 3$ e $v = 6$ para diagonalizar esses dois pares. Para isso, realizando as operações $L_4 \leftarrow L_4 + 7/4L_6$, $C_4 \leftarrow C_4 + 7/4C_6$, $L_4 \leftarrow L_4 + 17/4L_3$, $C_4 \leftarrow C_4 + 17/4C_3$ produzindo a caixa

$$N_{12}^* = \begin{array}{c} \begin{array}{ccc} 6 & 3 & 4 \\ \hline 2 & 0 & 0 \\ \hline 0 & -2 & 17/2 \\ \hline 0 & 17/2 & -49/8 \end{array} \\ \begin{array}{c} 6 \\ 3 \\ 4 \end{array} \end{array} \quad N_{12}^* = \begin{array}{c} \begin{array}{ccc} 6 & 3 & 4 \\ \hline 2 & 0 & 0 \\ \hline 0 & -2 & 0 \\ \hline 0 & 0 & 30 \end{array} \\ \begin{array}{c} 6 \\ 3 \\ 4 \end{array} \end{array} \quad N_{12} = \begin{array}{c} 4 \\ \left[30 \right] \end{array}$$

Neste passo os pares $(6, 2)$ e $(3, -2)$ são obtidos e armazenados para produção de D e as linhas associadas a eles são retiradas para formar N_{12} . Por fim o nó $i = 13$ esquece o vértice $v = 4$. Dessa forma a única entrada da caixa N_{13} se torna $0 + m_{44}$ e o algoritmo armazena o valor $(4, m_{44})$ para produzir D .

$$N_{13}^* = \begin{array}{c} 4 \\ \left[30 \right] \end{array} \quad N_{13}^* = \begin{array}{c} 4 \\ \left[30-5 \right] \end{array} \quad N_{13} = \begin{array}{c} 4 \\ \left[25 \right] \end{array} \quad \begin{array}{cc} \mathbf{E} & \mathbf{E} \\ \text{---} \textcircled{4} \text{---} & \text{---} \textcircled{} \text{---} \\ & 12 \quad 13 \end{array}$$

Dessa forma o algoritmo produz o último par $(4, 25)$ e produz a matriz diagonal D que é congruente a M como segue:

$$D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Pela Lei de Inércia de Sylvester, podemos concluir que o algoritmo apresentado foi capaz de determinar que M possui um autovalor nulo com multiplicidade 1, 3 autovalores positivos e 3 autovalores negativos. No próximo capítulo, analisaremos como esse algoritmo se relaciona com outros algoritmos desenvolvidos para a mesma finalidade, mas com entradas específicas para determinadas classes de grafos.

5 LOCALIZANDO AUTOVALORES EM ÁRVORES

Em 2011, Jacobs e Trevisan [19] desenvolveram um algoritmo para localizar os autovalores da matriz de adjacência $A(T)$ de uma árvore T . Este algoritmo, que chamaremos de *Algoritmo de Árvore*, utiliza como entradas um número real α e uma árvore T e é executado diretamente sobre os vértices de T . Neste capítulo, apresentaremos o funcionamento do *Algoritmo de Árvore* e estabeleceremos um paralelo com o *Algoritmo Diagonal Arbórea* apresentado no capítulo anterior. Antes de discutirmos a correspondência entre eles, mostraremos como o *Algoritmo de Árvore* localiza os autovalores de uma árvore T , apresentando uma breve descrição seguida de um exemplo. Dada uma árvore T , escolhe-se um vértice arbitrário para ser a raiz da árvore. A partir da raiz, podemos definir a relação de ordem parcial \preceq_T em $V(T)$ da seguinte forma: dado um par de vértices $a, b \in V(T)$, dizemos que $a \preceq_T b$ se $a = b$ ou se o caminho de a até a raiz de T passa por b (ou seja, b é um ancestral de a em T). Ao final do algoritmo, cada vértice receberá um valor x_i que corresponderá precisamente ao valor da entrada ii da matriz diagonal D produzida no final do processo. Nas figuras, o valor de x_i será exibido no interior dos vértices.

Em resumo, o algoritmo atribui valores aos vértices da árvore de forma recursiva. Quando um vértice é uma folha, o valor atribuído é o valor de entrada α . Caso o vértice tenha filhos com valores diferentes de zero, seu novo valor é calculado como a soma do valor de entrada α com a soma dos inversos negativos dos valores dos filhos. Se um vértice tiver filhos com valor zero, um desses filhos é escolhido para receber o valor 2, e o valor do vértice processado é definido como $-1/2$. Além disso, se o vértice processado estiver ligado a seu pai por uma aresta, essa aresta é removida durante o processo.

Algoritmo 6: Algoritmo Árvore

Entrada: Escalar α , árvore T com vértices $\{1, 2, \dots, n\}$
estendendo a ordem \preceq_T .

Saída: Matriz diagonal D congruente a $A + \alpha I$

1 Inicialize $x_v = \alpha$ para todo vértice v

2 **para** $k = 1$ até n **faça**

3 **se** k é uma folha **então** continue

4 **senão se** $x_c \neq 0$ para todo filho c de k **então**

5 | $x_k := x_k - \sum_c 1/x_c$

6 **senão**

7 | Selecione um filho j de k para qual $x_j = 0$ e defina

8 | $x_k := 2$

9 | $x_j := -1/2$

10 | **se** k tem um pai l **então**

11 | | remova a aresta que conecta os vértices k e l

Combinando o Teorema 2.5 com a corretude do *Algoritmo de Árvore* o Teorema 5.1 é obtido:

Teorema 5.1. *Seja D a matriz diagonal produzida pelo Algoritmo de Árvore com entrada uma árvore T e um escalar $-\alpha$ então:*

(a) *O número de entradas positivas em D é o número autovalores de T maiores que α .*

(b) *O número de entradas negativas em D é o número autovalores de T menores que α .*

(c) *Se há r entradas nulas na diagonal de D , então α é um autovalor de T com multiplicidade r .*

Exemplo 5.1. *Considere a árvore T da Figura 5.1. Executaremos o Algoritmo de Árvore para obter o número de autovalores de T que são maiores, menores e iguais a -1 .*

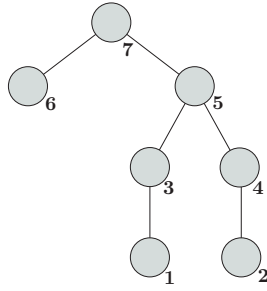


Figura 5.1: Ordenamento dos vértices de T que estende \preceq_T .

Inicialmente uma raiz é fixada para T e é realizado o ordenamento das folhas em direção à raiz. Para obter o número de autovalores maiores, menores ou iguais a -1 , inicializamos todos os vértices de T atribuindo o valor $\alpha = 1$. O Algoritmo 6 não executa nenhuma operação para folhas, dessa forma os vértices **1** e **2** são processados com valores atribuídos $x_1 = x_2 = 1$ e podemos seguir para seus pais. Ao processarmos o vértice **3** que é pai do vértice **1**, observamos que não há filhos com valor nulo e portanto o valor do vértice **3** se torna

$$x_3 = \alpha - \frac{1}{x_1} = 1 - \frac{1}{1} = 0.$$

Repetindo o processo para o vértice **4** obtemos $x_4 = 0$.

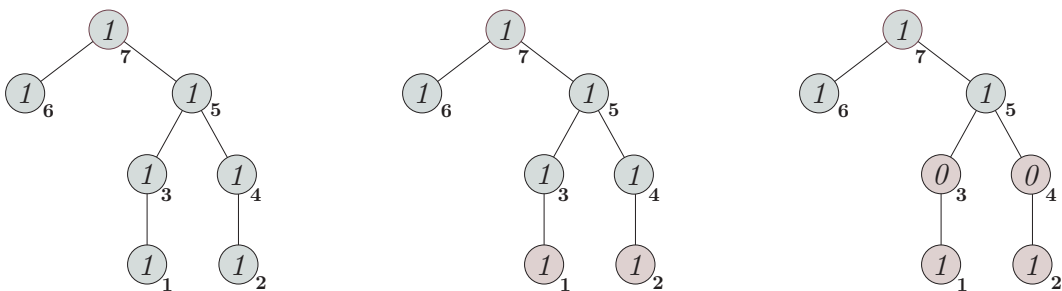


Figura 5.2: Processamentos dos vértice **1** a **4** de T .

Representamos os vértices com preenchimento vermelho aqueles que já foram processados. Ao processar o vértice **5** podemos notar que pelo menos um de seus filhos tem valor nulo. Neste caso é selecionado algum destes filhos para receber

o novo valor de 2 enquanto que o vértice **5** recebe o valor de processamento $-1/2$ e a ligação com seu pai, o vértice **6** é eliminada. Por fim o vértice **6** se tornou uma folha e portanto possui valor atribuído $x_6 = \alpha$. O vértice **7** é a raiz de T , e na ausência de filhos com valores nulos sua entrada se torna

$$x_7 = \alpha - \frac{1}{x_6} = 1 - \frac{1}{-1} = 0.$$

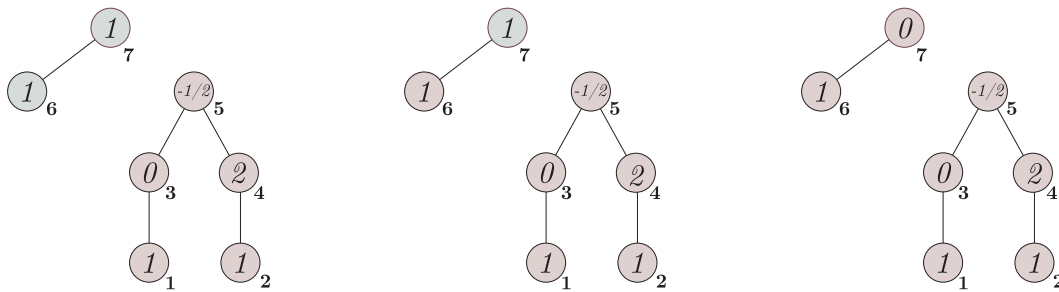


Figura 5.3: Processamentos dos vértice **5** a **7** de T .

Como o vértice **7** é raiz o algoritmo, o laço é encerrado e os valores produzidos em D são os valores contidos nos vértices da floresta produzida acima. Portanto o algoritmo indica que há dois autovalores iguais a -1 pela indicação de dois 0 nos vértices da floresta, quatro autovalores maiores que -1 e um autovalor menor que -1 .

O *Algoritmo de Árvore* foi uma das ferramentas fundamentais para resolver conjecturas importantes em Teoria Espectral de Grafos, muitas delas envolvendo a busca de valores de mínimo ou máximo de algum parâmetro espectral como em [11], [18] e [26]. No entanto, sua aplicabilidade é limitada à classe de grafos para a qual ele foi desenvolvido, ou seja, árvores. Já o *Algoritmo Diagonal Arbórea* é aplicável em diversas classes de grafos, inclusive em árvores. Para ilustrar a diferença entre esses algoritmos, mostraremos como o *Algoritmo Diagonal Arbórea* funciona quando o grafo de entrada é uma árvore, comparando-o com o algoritmo

apresentado neste capítulo. Para isso, fixamos um enraizamento e um ordenamento de T e construímos uma decomposição agradável \mathcal{T} , de modo que os nós esquecidos estejam na mesma ordem e sequência que os nós que são processados em T na execução do *Algoritmo de Árvore*. Em seguida, apresentaremos um lema que descreve a transição de T para \mathcal{T} .

Lema 5.2. *Dada uma árvore T enraizada, é possível definir um ordenamento de T , construir \mathcal{T} e ordenar os nós de tal modo que a sequência de diagonalização se equivalem.*

Demonstração. Seja uma árvore T e \mathbf{r} um vértice arbitrário que será a raiz de T . Para construir \mathcal{T} , ordenamos os vértices de T de forma que todo filho tenha um rótulo menor que seu pai. A construção é feita a partir das folhas de T em direção à raiz \mathbf{r} .

Começamos considerando a distância d do caminho mais longo de \mathbf{r} até uma folha de T . Para cada folha que esteja a uma distância d de \mathbf{r} , construímos uma bolsa que contém a folha e seu pai. Em seguida, adjacente a cada bolsa, criamos uma nova bolsa que esquece o vértice que é a folha de T .

Ao final deste passo, teremos uma união disjunta de árvores isomorfas a K_2 , em que cada árvore possui um nó cuja bolsa tem tamanho 1. Em seguida, no segundo passo, enquanto houver componentes enraizadas em nós com bolsas iguais, combinamos duas dessas componentes em uma única componente através de um nó junção. Essa transição pode ser visualizada no diagrama abaixo:

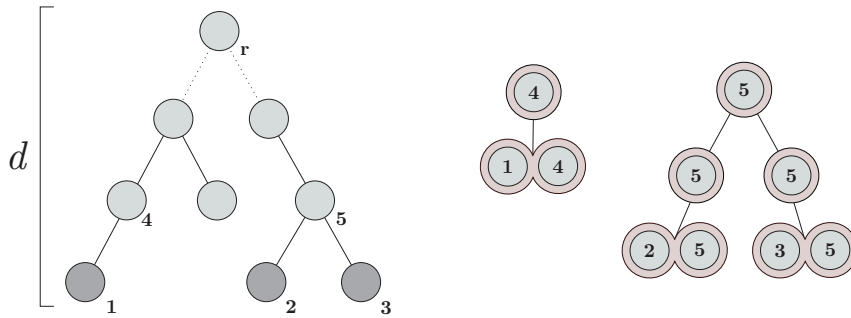


Figura 5.4: A floresta obtida após a realização dos dois primeiros passos na construção de \mathcal{T} .

No próximo passo, consideramos a árvore T' , que é obtida removendo todas as folhas de T que têm distância máxima d da raiz \mathbf{r} . Para cada vértice que é folha tanto em T quanto em T' , construímos uma bolsa que contém a folha e seu respectivo pai, e adicionamos uma nova bolsa adjacente que esquece o vértice que é folha em T , conforme descrito anteriormente. Para os vértices que não são folhas em T , mas que são folhas em T' , anexamos um nó introdução à subárvore que é vizinha do vértice no passo anterior e realizamos o esquecimento do vértice advindo da subárvore. A figura 5.5 a seguir ilustra esse passo.

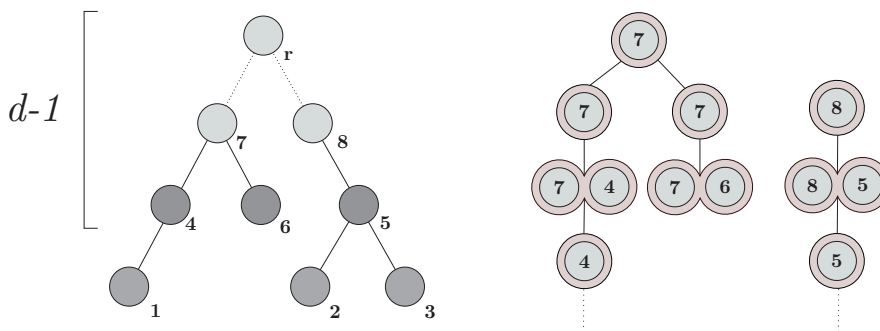


Figura 5.5: Transição de T para \mathcal{T} .

Ao seguir esse processo iterativamente até chegarmos nos filhos da raiz \mathbf{r} , podemos construir uma árvore \mathcal{T} que é uma decomposição agradável de T , com a mesma sequência de processamento. Em outras palavras, a ordem dos nós que são

processados no *Algoritmo Diagonal Arbórea* é exatamente a mesma que na execução do *Algoritmo de Árvore* na árvore original T . \square

Iniciaremos o paralelo entre os *Algoritmo de Árvore* e *Algoritmo Diagonal Arbórea* fixando a matriz de entrada $M = A(T) + \alpha I$, onde $A(T)$ é a matriz de adjacência da árvore T e α é um escalar real arbitrário. Vamos analisar a semelhança entre os Algoritmos de Árvore e Diagonal Arbórea em dois passos principais que dependem da posição que os vértices ocupam na árvore. Seja \mathbf{j} um vértice quase pendente de T . Denotamos por $\{\mathbf{j}_1, \dots, \mathbf{j}_k\}$ o conjunto dos $k \geq 1$ filhos de \mathbf{j} .

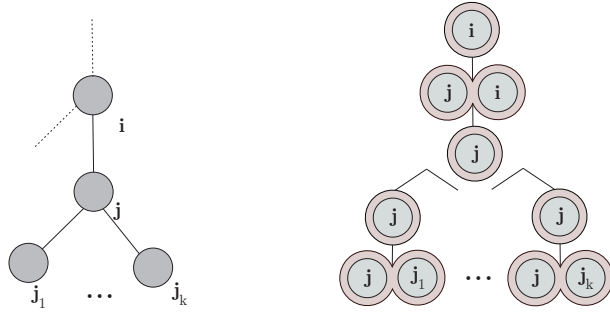


Figura 5.6: Representação de uma transição de T para \mathcal{T} .

Abordaremos nossa análise em dois casos. No primeiro caso, vamos analisar as caixas produzidas pelo *Algoritmo Diagonal Arbórea* onde todos os filhos do vértice \mathbf{j} foram diagonalizados quando seu nó foi esquecido. Seja \mathbf{j}_ℓ um filho de \mathbf{j} para algum $\ell \in \{1, \dots, k\}$. Por construção criou-se uma caixa rotulada por \mathbf{j} e \mathbf{j}_ℓ quando estamos em um nó folha, uma caixa intermediária no nó que ocorre o esquecimento de \mathbf{j}_ℓ e por fim a caixa resultante da operação. As caixas mencionadas podem ser observadas abaixo:

$$\begin{array}{ccc}
 \begin{array}{c} \mathbf{j} \quad \mathbf{j}_\ell \\ \mathbf{j} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{array} &
 \begin{array}{c} \mathbf{j}_\ell \quad \mathbf{j} \\ \mathbf{j}_\ell \begin{bmatrix} x_{\mathbf{j}_\ell} & 1 \\ 1 & 0 \end{bmatrix} \end{array} &
 \begin{array}{c} \mathbf{j} \\ \mathbf{j} \begin{bmatrix} -1/x_{\mathbf{j}_\ell} \end{bmatrix} \end{array}
 \end{array}$$

Portanto as caixas deixadas para os próximos passos possuem um único vértice que contém o inverso negativo do valor de atribuição ao qual foram diago-

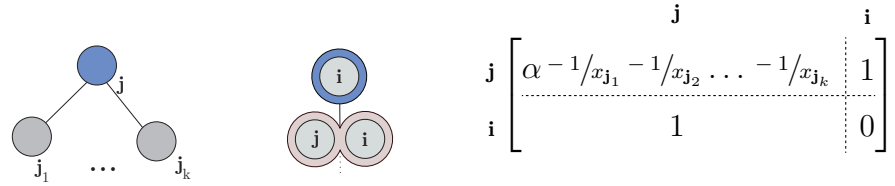
nalizados. Sejam $x_{j_1}, x_{j_2}, \dots, x_{j_k}$ os k valores diagonalizados. As caixas possuem o formato:

$$\begin{matrix} & \mathbf{j} & & \mathbf{j} & & \mathbf{j} \\ \mathbf{j} & \left[-1/x_{j_1} \right] & , & \mathbf{j} & \left[-1/x_{j_2} \right] & , \dots , & \mathbf{j} & \left[-1/x_{j_k} \right] \end{matrix}$$

As operações de junção concatenarão as entradas relacionadas ao vértice \mathbf{j} duas a duas até restar uma única caixa obtendo dessa forma:

$$\begin{matrix} & & & \mathbf{j} & & & \\ \mathbf{j} & \left[-1/x_{j_1} - 1/x_{j_2} - \dots - 1/x_{j_k} \right] \end{matrix}$$

Como o próximo passo em \mathcal{T} é a introdução do vértice \mathbf{i} incluímos uma linha nula. O próximo nó a ser processado esquece o vértice \mathbf{j} e obtemos a caixa intermediária da forma:



Neste ponto as entradas do *Algoritmo Diagonal Arbórea* e o *Algoritmo de Árvore* coincidem e são exatamente α subtraído da soma dos inversos dos seus filhos como no *Algoritmo de Árvore*. Caso $\alpha - 1/x_{j_1} \dots - 1/x_{j_k} \neq 0$ então a entrada relacionada ao vértice \mathbf{j} é diagonalizada, caso contrário esta caixa ficará de herança para o processamento dos nós restantes tornando o vértice \mathbf{j} do tipo I.

Já no segundo caso, algum filho \mathbf{j}_ℓ de \mathbf{j} não foi diagonalizado quando ocorreu seu esquecimento. Neste ponto a caixa produzida pelo nó que esquece \mathbf{j}_ℓ possui a estrutura abaixo:

$$\begin{matrix} & \mathbf{j}_\ell & \mathbf{j} \\ \mathbf{j}_\ell & \left[\begin{array}{c|c} 0 & 1 \\ \hline 1 & * \end{array} \right] \end{matrix}$$

Ao realizar o processo de junção, obteremos nas caixas do caso anterior o mesmo processo de soma. Iremos explicar o que acontece quando juntamos duas caixas que possuem vértices do tipo I. Sem perda de generalidade supomos que \mathbf{j}_a e

\mathbf{j}_b são filhos que não foram diagonalizados quando processados. A junção transforma a nova caixa intermediária em

$$\begin{array}{c} \mathbf{j}_a \\ \mathbf{j}_b \\ \mathbf{j} \end{array} \left[\begin{array}{cc|c} \mathbf{j}_a & \mathbf{j}_b & \mathbf{j} \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ \hline 1 & 1 & *_{a} + *_{b} \end{array} \right]$$

O procedimento de junção realiza o empilhamento das linhas do tipo I e realiza a soma das entrada dos vértice do tipo II com o mesmo rótulo. Para deixar a caixa na forma padrão devemos deixar a submatriz relacionado aos vértice de tipo I na forma escalonada. Realizando as operações elementares $L_{\mathbf{j}_b} \leftarrow L_{\mathbf{j}_b} - L_{\mathbf{j}_a}$ seguida de $C_{\mathbf{j}_b} \leftarrow C_{\mathbf{j}_b} - C_{\mathbf{j}_a}$. Estas operações produzem a caixa

$$\begin{array}{c} \mathbf{j}_a \\ \mathbf{j}_b \\ \mathbf{j} \end{array} \left[\begin{array}{cc|c} \mathbf{j}_a & \mathbf{j}_b & \mathbf{j} \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \hline 1 & 0 & *_{a} + *_{b} \end{array} \right]$$

e não são capazes de afetar a linha que relaciona ao vértice \mathbf{j} . Este processo é repetido até restar somente uma caixa neste formato que deixará como herança para seu pai a caixa do formato

$$\begin{array}{c} \mathbf{j}_k \\ \mathbf{j} \end{array} \left[\begin{array}{c|c} & \mathbf{j} \\ 0 & 1 \\ \hline 1 & *_{k} \end{array} \right].$$

Nos próximos passos em \mathcal{T} introduzimos uma linha nula rotulada por \mathbf{i} e esquecemos o vértice \mathbf{j} .

$$\begin{array}{c} \mathbf{j} \\ \mathbf{j}_k \\ \mathbf{i} \end{array} \left[\begin{array}{cc|c} \mathbf{j} & \mathbf{j}_k & \mathbf{i} \\ *_{j} & 1 & 1 \\ 1 & 0 & 0 \\ \hline 1 & 0 & *_{i} \end{array} \right]$$

Observe que possuímos uma linha do tipo I e que o vetor $\mathbf{x}_v \neq 0$. As operações a serem realizadas seguem do subcaso 2 do *Algoritmo Diagonal Arbórea*.

Se $*_j \neq 0$ realiza-se as operações $L_j \leftarrow L_j - *_j/2L_{j_k}$ seguida de $C_j \leftarrow C_j - *_j/2C_{j_k}$ para eliminar a entrada diagonal de \mathbf{j} .

$$\begin{array}{c} \mathbf{j} \quad \mathbf{j}_k \quad \mathbf{i} \\ \mathbf{j} \left[\begin{array}{cc|c} 0 & 1 & 1 \\ 1 & 0 & 0 \\ \hline 1 & 0 & *_i \end{array} \right] \\ \mathbf{j}_k \\ \mathbf{i} \end{array}$$

Note que as operações anteriores não modificam nenhuma entrada da caixa. Em seguida, são realizadas as operações $L_{j_k} \leftarrow L_{j_k} + 1/2L_j$, $C_{j_k} \leftarrow C_{j_k} + 1/2C_j$, $L_j \leftarrow L_j - L_{j_k}$ e $C_j \leftarrow C_j - C_{j_k}$ produzindo a caixa da forma:

$$\begin{array}{c} \mathbf{j} \quad \mathbf{j}_k \quad \mathbf{i} \\ \mathbf{j} \left[\begin{array}{cc|c} -1 & 0 & 1/2 \\ 0 & 1 & 1/2 \\ \hline 1/2 & 1/2 & *_i \end{array} \right] \\ \mathbf{j}_k \\ \mathbf{i} \end{array}$$

Neste ponto existem pivôs nas diagonais associadas aos vértices \mathbf{i} e \mathbf{j}_k aptos para diagonalizar suas respectivas linhas. Realizando as operações elementares $L_i \leftarrow L_i + 1/2L_j$, $C_i \leftarrow C_i + 1/2C_j$, $L_i \leftarrow L_i - 1/2L_{j_k}$, $C_i \leftarrow C_i - 1/2C_{j_k}$ obtemos os pares diagonalizados $(\mathbf{j}, -1)$ e $(\mathbf{j}_k, 1)$ e a caixa resultante como:

$$\mathbf{i} \left[\begin{array}{c} \mathbf{i} \\ *_i \end{array} \right].$$

Podemos então concluir que os dois algoritmos apresentam semelhanças no tratamento de vértices quase pendentos, quando todos os seus filhos já foram diagonalizados. Nesses casos, ambos os algoritmos eliminam as ligações entre o vértice e seus filhos e prosseguem operando na subárvore restante. O *Algoritmo de Árvore* também produz esse resultado quando processa um vértice j com um filho j_ℓ nulo. No entanto, as entradas obtidas pelos dois algoritmos diferem. No *Algoritmo de Árvore*, quando um vértice é nulo, o algoritmo define valores de $-1/2$ e 2 , enquanto no

Algoritmo Diagonal Arbórea, os valores passam a ser $+1$ e -1 . No entanto, o sinal das entradas é o que realmente importa para a saída do algoritmo, independentemente dos valores serem diferentes para ambos os algoritmos. Agora, o próximo vértice a ser processado em \mathcal{T} herda as caixas resultantes das operações anteriores, seguindo o mesmo padrão dos casos apresentados. Nesse estágio, o vértice é considerado quase pendente. Esse processo é repetido até que a raiz \mathbf{r} seja esquecida e o algoritmo seja encerrado.

A seguir, apresentaremos uma especialização do *Algoritmo Diagonal Arbóreo* para uma classe de grafos conhecida como *cactos*. Além disso, mostraremos uma aplicação dessa classe, utilizando o algoritmo para sua execução.

6 GRAFOS CACTOS

Neste capítulo, estudaremos como o *Algoritmo Diagonal Arbórea* é capaz de diagonalizar uma matriz $M = A(G) + D$, em que D é uma matriz diagonal arbitrária e $A(G)$ é a matriz de adjacências de G quando aplicado a um grafo cacto. Veremos também uma aplicação específica do algoritmo para esta classe de grafos.

Definição 6.1. Um *cacto* é um grafo conexo em que quaisquer dois ciclos possuem no máximo um vértice em comum, ou, equivalentemente, no qual toda aresta pertence a no máximo um ciclo do grafo.

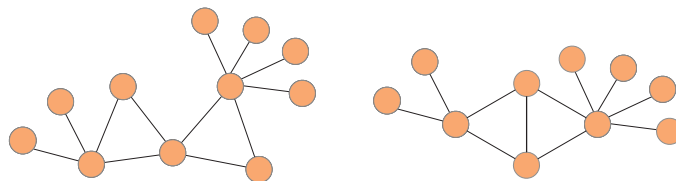


Figura 6.1: Um grafo cacto à esquerda e um grafo que não é cacto à direita.

Os grafos cactos foram inicialmente estudados sob o nome de Árvore Husimi em [15], em homenagem a um trabalho anterior de Kôdi Husimi [17]. Entretanto, o termo cactos era originalmente aplicado apenas a grafos em que cada ciclo era um triângulo. Atualmente, a nomenclatura se estende a grafos com ciclos de qualquer tamanho. Neste capítulo, vamos aprofundar nosso conhecimento sobre como o *Algoritmo Diagonal Arbórea* é executado para essa classe de grafos. Em seguida, aplicaremos o algoritmo para obter resultados sobre grafos cactos *reflexivos*, uma vertente que tem sido pesquisada nos últimos anos por um grupo de pesquisadores sérvios [29].

6.1 Aplicando o Algoritmo Diagonal Arbórea a grafos Cactos

Abordaremos os casos de processamento dos vértices em etapas. Iniciaremos analisando como o *Algoritmo Diagonal Arbórea* lida com apenas um ciclo C_b e seus possíveis desdobramentos. Em seguida, veremos como o algoritmo trata os vértices de um ciclo que podem conter árvores e ciclos pendurados. Em cada etapa, demonstraremos como o *Algoritmo Diagonal Arbórea* pode ser executado diretamente no grafo, apresentando alguns exemplos para auxiliar na compreensão do procedimento. Fixaremos, para essa análise, um ciclo C_b com b vértices e uma decomposição arbórea \mathcal{T} .

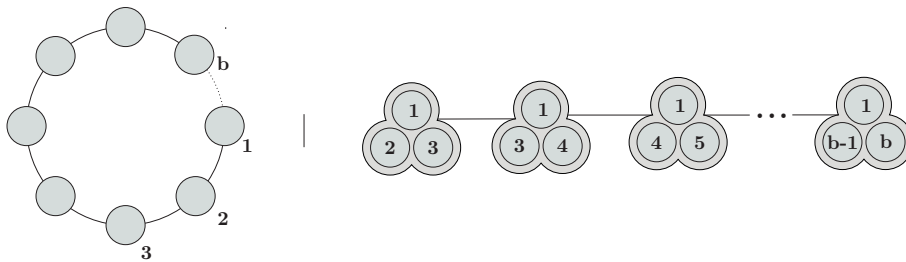


Figura 6.2: Ciclo C_b e uma decomposição arbórea \mathcal{T} .

No Capítulo 3, discutimos que uma árvore tem largura arbórea igual a 1, enquanto um grafo conexo com ciclo tem largura arbórea maior ou igual a 2. Para um ciclo C_b , uma decomposição arbórea com largura ótima tem largura arbórea igual a 2. Ao transformar uma decomposição arbórea da Figura 6.2 para uma decomposição arbórea agradável, a sequência de esquecimentos e introduções inicia com o esquecimento do vértice **2**, a introdução do vértice **4** e termina com o esquecimento do vértice **b** seguido do vértice inicial do ciclo..

Durante o procedimento utilizaremos $M = A(G) + D$ como entrada do algoritmo onde D é uma matriz diagonal com valores arbitrários em sua diagonal e $A(G)$ é a matriz de adjacências de G . Denotaremos por x_v o valor da entrada

m_{vv} somado com a entrada vv obtida das caixas do algoritmo para algum vértice $\mathbf{v} \in [b]$. O algoritmo começa com um nó folha, criando uma caixa $\mathbf{0}_{3 \times 3}$ nula rotulada com os vértices da bolsa $B_1 = \{\mathbf{1}, \mathbf{2}, \mathbf{3}\}$ e com entradas nulas para todos os vértices. Em seguida, no nó seguinte da decomposição arbórea \mathcal{T} , ocorre o esquecimento do vértice $\mathbf{2}$, e as adjacências entre este vértice e os vértices que continuam na bolsa são incorporadas na caixa.

Vamos dividir nossa análise em dois casos para o processamento dos vértices em um grafo cacto que é apenas um ciclo C_b . Esses casos serão denominados de 1a e 1b. No caso 1a, mostraremos como um ciclo é processado sem ocorrer entradas de diagonalização nula na caixa. Já no caso 1b, vamos analisar o que ocorre quando um vértice é esquecido e não é diagonalizado.

- Caso 1a: $x_v \neq 0, \forall v \in \{1, \dots, b\}$.

Na ilustração abaixo representamos o esquecimento vértice $\mathbf{2}$ e a saída do algoritmo.

	$\mathbf{2}$	$\mathbf{1}$	$\mathbf{3}$		$\mathbf{2}$	$\mathbf{1}$	$\mathbf{3}$		\mathbf{F}	\mathbf{E}	\mathbf{I}
$\mathbf{2}$	x_2	1	1		x_2	0	0		$\begin{matrix} \textcircled{1} \\ \textcircled{2} \textcircled{3} \end{matrix}$	$\begin{matrix} \textcircled{1} \\ \textcircled{3} \end{matrix}$	$\begin{matrix} \textcircled{1} \\ \textcircled{3} \textcircled{4} \end{matrix} \dots$
$\mathbf{1}$	1	0	0		0	$-1/x_2$	$-1/x_2$				
$\mathbf{3}$	1	0	0		0	$-1/x_2$	$-1/x_2$				

Como $x_2 \neq 0$, esta entrada é utilizada para diagonalizar a linha associada ao vértice $\mathbf{2}$. O próximo nó de \mathcal{T} introduz o vértice $\mathbf{4}$. Em seguida, esquecemos o vértice $\mathbf{3}$, obtendo a caixa representada a seguir, que é a caixa produzida pelo nó.

	$\mathbf{3}$	$\mathbf{1}$	$\mathbf{4}$		$\mathbf{3}$	$\mathbf{1}$	$\mathbf{4}$
$\mathbf{3}$	x_3	$-1/x_2$	1		x_3	0	0
$\mathbf{1}$	$-1/x_2$	$-1/x_2$	0		0	$-\frac{1}{x_2} - \frac{1}{x_2^2 \cdot x_3}$	$\frac{1}{x_2 \cdot x_3}$
$\mathbf{4}$	1	0	0		0	$\frac{1}{x_2 \cdot x_3}$	$-\frac{1}{x_3}$

Pela condição imposta e repetindo o processo para todos os vértices, ao esquecer o vértice \mathbf{b} de C_b , a penúltima caixa se torna:

$$\begin{array}{c} \mathbf{b} \\ \mathbf{1} \end{array} \left[\begin{array}{c|c} \mathbf{b} & \mathbf{1} \\ \hline x_{\mathbf{b}} & 1/x_2 \cdot x_3 \dots x_{\mathbf{b}-1} + 1 \\ \hline 1/x_2 \cdot x_3 \dots x_{\mathbf{b}-1} + 1 & f(x_2, x_3, \dots, x_{\mathbf{b}-1}) \end{array} \right]$$

onde

$$f(x_2, x_3, x_4, \dots, x_{\mathbf{b}-1}) = -\frac{1}{x_2} - \frac{1}{x_2^2 x_3} - \frac{1}{x_2^2 x_3^2 x_4} - \frac{1}{x_2^2 x_3^2 x_4^2 x_5} \dots - \frac{1}{x_2^2 x_3^2 \dots x_{\mathbf{b}-1}}.$$

O valor dessa diagonal é importante para calcular o valor dos novos pesos que conectam o vértice $\mathbf{1}$ ao vértice final do ciclo \mathbf{b} . Se $\frac{1}{x_2 \cdot x_3 \dots x_{\mathbf{b}-1}} + 1 = 0$, então o vértice \mathbf{b} pode ser diagonalizado com o par $(b, x_{\mathbf{b}})$ sem a necessidade de nenhuma operação sobre as linhas/colunas da caixa. Ao esquecer o vértice $\mathbf{1}$, a entrada da caixa se torna:

$$a_{\mathbf{1}} = -\frac{1}{x_2} - \frac{1}{x_2^2 x_3} - \frac{1}{x_2^2 x_3^2 x_4} \dots - \frac{1}{x_2^2 x_3^2 \dots x_{\mathbf{b}-1}} + m_{11},$$

e a última entrada diagonalizada é $(\mathbf{1}, a_{\mathbf{1}})$.

Se esse não for o caso, as operações que o algoritmo realiza tornam a entrada do vértice $\mathbf{1}$ como:

$$-\frac{1}{x_2} - \frac{1}{x_2^2 x_3} - \frac{1}{x_2^2 x_3^2 x_4} \dots - \frac{1}{x_2^2 x_3^2 \dots x_{\mathbf{b}-1}} - \left(\frac{1}{x_2 \cdot x_3 \dots x_{\mathbf{b}-1}} + 1 \right)^2 \cdot \frac{1}{x_b} + m_{11}.$$

A expressão acima fornece o cálculo do valor do último vértice do ciclo a ser esquecido, o vértice $\mathbf{1}$.

Os casos apresentados neste capítulo possuem semelhanças com o *Algoritmo de Unicíclicos* [4]. Durante o processamento dos ciclos, a entrada relacionada ao vértice $\mathbf{1}$ é atualizada a cada vez que os demais vértices são esquecidos, seguindo

a mesma lógica do algoritmo unicíclico, onde a entrada é atualizada a cada etapa. A dificuldade está em determinar o valor final dessa entrada ao término do processo.

No exemplo a seguir, demonstraremos como diagonalizar um ciclo de quatro vértices usando a matriz $M = A(G) - 2I$ com $D = -2I$. Mostraremos como o algoritmo gera as caixas e os valores diagonais, e posteriormente como podemos simular o processo diretamente no ciclo.

Exemplo 6.1. Considere o grafo $G = C_4$ com sua decomposição agradável \mathcal{T} ilustrada na Figura 6.3, e matriz $M = A(G) - 2I$.

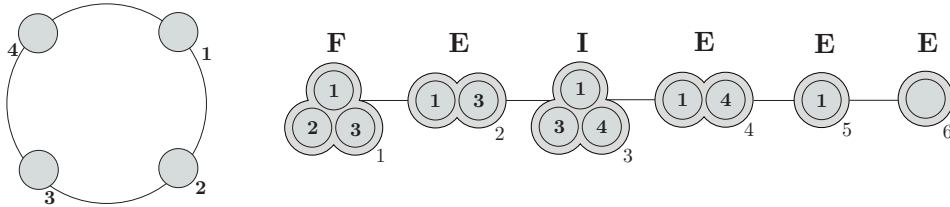


Figura 6.3: Grafo C_4 e uma decomposição agradável \mathcal{T} .

Quando $i = 1$, o nó é do tipo folha e uma caixa $\mathbf{0}^{3 \times 3}$ é criada e rotulada com os vértices da bolsa $B_1 = \{1, 2, 3\}$. No nó $i = 2$, o vértice $\mathbf{2}$ é esquecido e a seguinte sequência de operações é realizada:

$$N_2^* = \begin{array}{c} \mathbf{2} \\ \mathbf{1} \\ \mathbf{3} \end{array} \begin{array}{c|cc} \mathbf{2} & \mathbf{1} & \mathbf{3} \\ \hline -2 & 1 & 1 \\ \hline 1 & 0 & 0 \\ \hline 1 & 0 & 0 \end{array} \longrightarrow \begin{array}{c} \mathbf{2} \\ \mathbf{1} \\ \mathbf{3} \end{array} \begin{array}{c|cc} \mathbf{2} & \mathbf{1} & \mathbf{3} \\ \hline x_2 = -2 & 0 & 0 \\ \hline 0 & 1/2 & 1/2 \\ \hline 0 & 1/2 & 1/2 \end{array} \longrightarrow \begin{array}{c} \mathbf{1} \\ \mathbf{3} \end{array} \begin{array}{c|cc} \mathbf{1} & \mathbf{3} \\ \hline 1/2 & 1/2 \\ \hline 1/2 & 1/2 \end{array}$$

As operações realizadas no algoritmo diagonalizam o vértice $\mathbf{2}$ produzindo o par $(\mathbf{2}, -2)$. O nó $i = 3$ introduz o vértice $v = \mathbf{4}$ adicionando uma linha nula em ordem crescente com as linhas do tipo II que estão na caixa. Já o nó $i = 4$ esquece o vértice $\mathbf{3}$. Para diagonalização do vértice $\mathbf{3}$, são realizadas modificações na

caixa N_4^* , como apresentado abaixo:

$$N_4^* = \begin{array}{c} \mathbf{3} \\ \mathbf{1} \\ \mathbf{4} \end{array} \begin{array}{c} \mathbf{3} \quad \mathbf{1} \quad \mathbf{4} \\ \left[\begin{array}{c|cc} -2 + 1/2 & 1/2 & 1 \\ \hline 1/2 & 1/2 & 0 \\ 1 & 0 & 0 \end{array} \right] \end{array} \longrightarrow \begin{array}{c} \mathbf{3} \\ \mathbf{1} \\ \mathbf{4} \end{array} \begin{array}{c} \mathbf{3} \quad \mathbf{1} \quad \mathbf{4} \\ \left[\begin{array}{c|cc} x_{\mathbf{3}} = -3/2 & 0 & 0 \\ \hline 0 & 2/3 & 1/3 \\ 0 & 1/3 & 2/3 \end{array} \right] \end{array} \longrightarrow \begin{array}{c} \mathbf{1} \\ \mathbf{4} \end{array} \begin{array}{c} \mathbf{1} \quad \mathbf{4} \\ \left[\begin{array}{cc} 2/3 & 1/3 \\ 1/3 & 2/3 \end{array} \right] \end{array}$$

As operações produzem o par $(\mathbf{3}, -3/2)$. O nó $i = 5$ esquece o vértice $\mathbf{4}$, produzindo as caixas

$$N_5^* = \begin{array}{c} \mathbf{4} \\ \mathbf{1} \end{array} \begin{array}{c} \mathbf{1} \quad \mathbf{4} \\ \left[\begin{array}{cc} -2 + 2/3 & 4/3 \\ 4/3 & 2/3 \end{array} \right] \end{array} \longrightarrow \begin{array}{c} \mathbf{4} \\ \mathbf{1} \end{array} \begin{array}{c} \mathbf{1} \quad \mathbf{4} \\ \left[\begin{array}{cc} x_{\mathbf{4}} = -4/3 & 0 \\ 0 & 2 \end{array} \right] \end{array} \longrightarrow \mathbf{1} \begin{array}{c} \mathbf{1} \\ \left[2 \right] \end{array}.$$

Neste passo o par $(\mathbf{4}, -4/3)$ é obtido. Por fim, no nó $i = 6$ é atualiza o valor do vértice $\mathbf{1}$, produzindo o último valor diagonal $x_{\mathbf{1}} = 0$ com o par $(\mathbf{1}, 0)$.

Uma vantagem de obter uma expressão definida para o último vértice de um ciclo C_b é a capacidade de tratar um ciclo de forma semelhante a um caminho, com exceção do último vértice, que pode ser calculado ao final do processo. Isso pode facilitar muito o processo de diagonalização de um ciclo. Para ilustrar essa ideia e comparação, considere os valores produzidos por um caminho de 4 vértices abaixo:

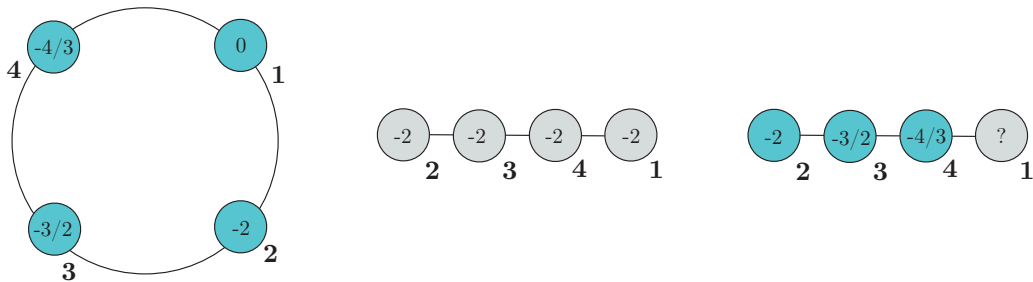


Figura 6.4: Um ciclo C_4 com os valores processados e um caminho P_4 sendo processado.

Diferentemente de um caminho, no caso de um ciclo, precisamos calcular a expressão para determinar o valor associado ao último vértice. A expressão para o último vértice de um ciclo C_b é dada por:

$$-\frac{1}{x_2} - \frac{1}{x_2^2 x_3} - \left(\frac{1}{x_2 \cdot x_3} + 1 \right)^2 \cdot \frac{1}{x_4} + m_{11} = \frac{1}{2} + \frac{1}{6} + \left(\frac{4}{3} \right)^2 \cdot -\frac{3}{4} + 2 = 0.$$

Ao final deste exemplo, concluímos que a matriz M possui um autovalor de 0 e três autovalores negativos, e como $M = A - 2I$, então C_4 tem um autovalor 2 (o índice) e os demais 3 autovalores menores do que 2.

Isso conclui o caso 1a, em que todos os vértices foram diagonalizados ao serem esquecidos e nenhum vértice se tornou do tipo I durante o processamento.

- Caso 1b: para algum $\mathbf{k} \in \{2, \dots, b\}$ ocorre $x_k = 0$.

Seja \mathbf{k} o primeiro vértice ao qual o algoritmo atribuiu $x_k = 0$ e que o vértice \mathbf{k} tornou-se do tipo I. Caso $\mathbf{k} = \mathbf{b}$, nenhuma operação é realizada no algoritmo. O próximo nó na decomposição arbórea esquece o vértice $\mathbf{1}$. Mais precisamente, ao esquecer o vértice $\mathbf{1}$, o processo de diagonalização do algoritmo produz a seguinte sequência de caixas:

$$\begin{array}{c} \mathbf{k} \quad \mathbf{1} \\ \left[\begin{array}{c|c} 0 & a \\ \hline a & b \end{array} \right] \longrightarrow \begin{array}{c} \mathbf{1} \quad \mathbf{k} \\ \left[\begin{array}{c|c} * & a \\ \hline a & b \end{array} \right] \longrightarrow \begin{array}{c} \mathbf{1} \quad \mathbf{k} \\ \left[\begin{array}{c|c} -a & 0 \\ \hline 0 & a \end{array} \right] \end{array} \end{array}$$

Os pares $(\mathbf{k}, -a)$ e $(\mathbf{1}, a)$, são obtidos, gerando um valor positivo e negativo. Note que, caso $a = 0$ estaríamos no caso 1a apresentado anteriormente. O algoritmo seguiria um procedimento semelhante se estivéssemos processando um caminho, onde ao final do processo os dois últimos vértices produziriam um valor negativo e positivo. Situações como essas podem surgir quando aplicamos o algoritmo para matrizes de adjacência para ciclos de tamanho múltiplos de 4.

Se o vértice $\mathbf{k} \neq \mathbf{b}$, antes de esquecer o próximo vértice $\mathbf{k} + \mathbf{1}$, devemos adicionar o próximo vértice $\mathbf{k} + \mathbf{2}$, se existir. Abaixo é representada a sequência de

acontecimentos nas caixas que introduzem o vértice $\mathbf{k}+2$ e o esquecimento do vértice $\mathbf{k}+1$ seguido de sua diagonalização.

$$\begin{array}{c} \mathbf{k} \\ \mathbf{1} \\ \mathbf{k}+1 \end{array} \begin{array}{c} \mathbf{k} \quad \mathbf{1} \quad \mathbf{k}+1 \\ \left[\begin{array}{ccc|ccc} 0 & a & 1 & & & \\ \hline a & b & 0 & & & \\ \hline 1 & 0 & 0 & & & \end{array} \right] \end{array} \longrightarrow \begin{array}{c} \mathbf{k}+1 \\ \mathbf{k} \\ \mathbf{1} \\ \mathbf{k}+2 \end{array} \begin{array}{c} \mathbf{k}+1 \quad \mathbf{k} \quad \mathbf{1} \quad \mathbf{k}+2 \\ \left[\begin{array}{cccc|ccc} c & 1 & 0 & 1 & & & \\ \hline 1 & 0 & a & 0 & & & \\ \hline 0 & a & b & 0 & & & \\ \hline 1 & 0 & 0 & 0 & & & \end{array} \right] \end{array} \longrightarrow \begin{array}{c} \mathbf{k}+1 \\ \mathbf{k} \\ \mathbf{1} \\ \mathbf{k}+2 \end{array} \begin{array}{c} \mathbf{k}+1 \quad \mathbf{k} \quad \mathbf{1} \quad \mathbf{k}+2 \\ \left[\begin{array}{ccc|ccc} -1 & 0 & 0 & 0 & & \\ \hline 0 & 1 & 0 & 0 & & \\ \hline 0 & 0 & b+a^2c & -a & & \\ \hline 0 & 0 & -a & 0 & & \end{array} \right] \end{array}$$

Após a diagonalização dos pares $(\mathbf{k}+1, -1)$ e $(\mathbf{k}, 1)$, a caixa produzida possui algumas alterações em suas entradas.

$$\begin{array}{c} \mathbf{1} \\ \mathbf{k}+2 \end{array} \begin{array}{c} \mathbf{1} \quad \mathbf{k}+2 \\ \left[\begin{array}{cc|cc} b+a^2c & -a & & \\ \hline -a & 0 & & \end{array} \right] \end{array}$$

Como a entrada do vértice $\mathbf{k}+2$ não foi modificada, o processamento dos demais vértices do ciclo ocorre de modo análogo ao de um caminho começando em $\mathbf{k}+2$. A entrada do vértice $\mathbf{1}$ neste momento é dada por $b+a^2c$. Caso $\mathbf{k}+2$ seja o último vértice do ciclo, o valor da entrada do vértice $\mathbf{1}$ se torna:

$$b+a^2c - \frac{(a-1)^2}{\mathbf{x}_{\mathbf{k}+2}}.$$

Caso contrário, a soma dos valores relacionados ao vértice $\mathbf{1}$ se torna:

$$b+a^2c - \frac{a^2}{x_{\mathbf{k}+2}} - \dots - \frac{a^2}{x_{\mathbf{k}+2}^2 x_{\mathbf{k}+3}^2 \dots x_{\mathbf{b}-1}} - \left(-\frac{a}{x_{\mathbf{k}+2} \cdot x_{\mathbf{k}+3} \dots x_{\mathbf{b}-1}} + 1 \right)^2 \cdot \frac{1}{x_{\mathbf{b}}} + m_{11}$$

No trabalho de Braga [4], é abordado um algoritmo para localização de autovalores em grafos unicíclicos. Durante o desenvolvimento do algoritmo, também é realizada a atualização da entrada relacionada ao vértice inicial do ciclo. Um dos objetivos do *Algoritmo Diagonal Arbórea* para o caso 1a e 1b seria obter uma expressão geral que permitisse calcular os valores associados a todos os vértices do ciclo, de forma semelhante à como é feito em uma árvore, e assim recuperar, no

ponto de junção do ciclo, o valor do vértice **1**. No entanto, como abordado anteriormente, em alguns pontos do processo de diagonalização podem ocorrer anulações, o que torna difícil obter uma expressão fechada para a entrada do vértice. A seguir, apresentaremos um exemplo que ilustra esse problema em um ciclo de 5 vértices.

Exemplo 6.2. Considere o grafo C_5 com a decomposição agradável \mathcal{T} e M a matriz definida por

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & 1/2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 1 & -1/2 \end{bmatrix}.$$

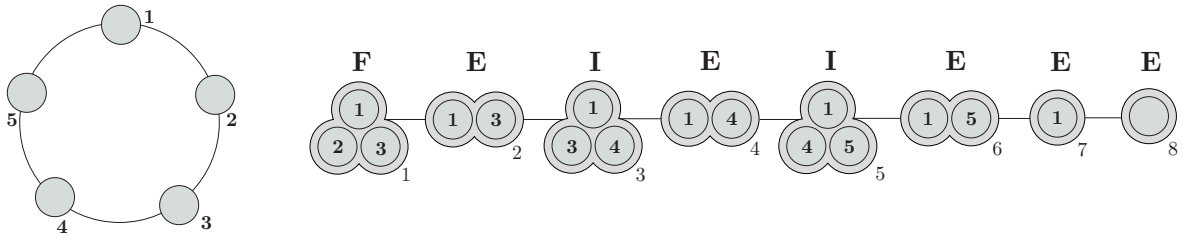


Figura 6.5: Ciclo C_5 e uma decomposição agradável \mathcal{T} .

Quando $i = 1$ o nó é do tipo folha e uma caixa $\mathbf{0}^{3 \times 3}$ com linhas e colunas rotuladas pelos vértices da bolsa é criada. O nó $i = 2$ esquece o vértice **2** produzindo a sequência de caixas:

$$N_2^* = \begin{array}{c} \mathbf{2} \\ \mathbf{1} \\ \mathbf{3} \end{array} \begin{array}{c|cc} & \mathbf{2} & \mathbf{1} & \mathbf{3} \\ \hline & -2 & 1 & 1 \\ \hline & 1 & 0 & 0 \\ \hline & 1 & 0 & 0 \end{array} \longrightarrow \begin{array}{c} \mathbf{2} \\ \mathbf{1} \\ \mathbf{3} \end{array} \begin{array}{c|cc} & \mathbf{2} & \mathbf{1} & \mathbf{3} \\ \hline & \mathbf{x}_2 = -2 & 0 & 0 \\ \hline & 0 & 1/2 & 1/2 \\ \hline & 0 & 1/2 & 1/2 \end{array} \longrightarrow \begin{array}{c} \mathbf{1} \\ \mathbf{3} \end{array} \begin{array}{c|cc} & \mathbf{1} & \mathbf{3} \\ \hline & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array},$$

O par $(\mathbf{2}, -2)$ é diagonalizado. No nó $i = 3$, o vértice $v = \mathbf{4}$ é introduzido, adicionando uma linha de zeros em ordem crescente às linhas do tipo II presentes na

caixa. Já no nó $i = 4$, o vértice **3** é esquecido e a caixa resultante é:

$$N_4^* = \begin{array}{c} \mathbf{3} \\ \mathbf{1} \\ \mathbf{4} \end{array} \left[\begin{array}{c|cc} & \mathbf{3} & \mathbf{1} & \mathbf{4} \\ \hline & 1/2 - 1/2 = 0 & 1/2 & 1 \\ \hline & 1/2 & 1/2 & 0 \\ \hline & 1 & 0 & 0 \end{array} \right]$$

A caixa $N_4^* = N_4$ está no formato padrão pela ordem definida nos nós de \mathcal{T} devemos adicionar o vértice **5** e, em seguida, esquecer o vértice **4**. A caixa intermediária N_5^* é mostrada abaixo, seguida das operações do Caso 1b que o *Algoritmo Diagonal Arbórea* realiza para diagonalização dos pares $(\mathbf{4}, -1)$ e $(\mathbf{3}, 1)$ que são retirados da caixa para obtenção de N_5 .

$$N_5^* = \begin{array}{c} \mathbf{4} \\ \mathbf{3} \\ \mathbf{1} \\ \mathbf{5} \end{array} \left[\begin{array}{c|ccc} & \mathbf{4} & \mathbf{3} & \mathbf{1} & \mathbf{5} \\ \hline & 2 & 1 & 0 & 1 \\ \hline & 1 & 0 & 1/2 & 0 \\ \hline & 0 & 1/2 & 1/2 & 0 \\ \hline & 1 & 0 & 0 & 0 \end{array} \right] \longrightarrow \begin{array}{c} \mathbf{4} \\ \mathbf{3} \\ \mathbf{1} \\ \mathbf{5} \end{array} \left[\begin{array}{c|ccc} & \mathbf{4} & \mathbf{3} & \mathbf{1} & \mathbf{5} \\ \hline & -1 & 0 & 0 & 0 \\ \hline & 0 & 1 & 0 & 0 \\ \hline & 0 & 0 & 1 & -1/2 \\ \hline & 0 & 0 & -1/2 & 0 \end{array} \right] \longrightarrow \begin{array}{c} \mathbf{1} \\ \mathbf{5} \end{array} \left[\begin{array}{c|c} & \mathbf{1} & \mathbf{5} \\ \hline & 1 & -1/2 \\ \hline & -1/2 & 0 \end{array} \right].$$

Na bolsa $i = 7$, ocorre o esquecimento do vértice **5** e, como resultado, obtemos as caixas a seguir:

$$N_7^* = \begin{array}{c} \mathbf{5} \\ \mathbf{1} \end{array} \left[\begin{array}{c|cc} & \mathbf{5} & \mathbf{1} \\ \hline & -1/2 & 1 - 1/2 \\ \hline & 1 - 1/2 & 1 \end{array} \right] \longrightarrow \begin{array}{c} \mathbf{5} \\ \mathbf{1} \end{array} \left[\begin{array}{c|c} & \mathbf{5} & \mathbf{1} \\ \hline & -1/2 & 0 \\ \hline & 0 & 3/2 \end{array} \right]$$

A diagonalização do vértice **5** é realizada com o par $(\mathbf{5}, -1/2)$. Por fim, o vértice $i = 8$ esquece o vértice **1** do ciclo, e seu valor é processado como $3/2 + m_{11} = 3/2$.

Como mencionado anteriormente, podemos tratar o ciclo como se fosse um caminho iniciando do vértice **2**, como no primeiro caso.

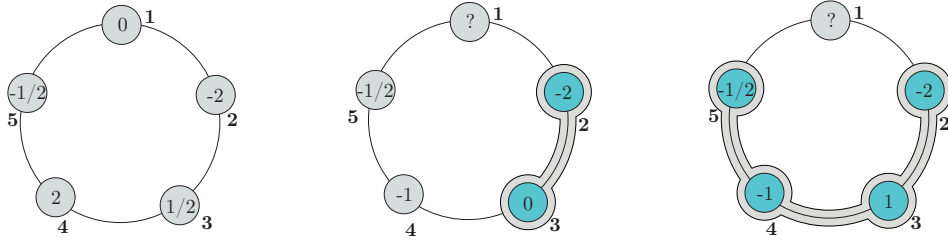


Figura 6.6: Operações em ciclos.

Observamos que, no primeiro caso, podemos tratar o ciclo como se fosse um caminho iniciando no vértice **2**. Como o cancelamento ocorre no vértice $\mathbf{k}=\mathbf{3}$ e o vértice $\mathbf{k}+\mathbf{2}$ é o último vértice do ciclo, podemos utilizar a expressão para obter o cálculo do valor do vértice **1**:

$$b + a^2c - \frac{(a-1)^2}{\mathbf{x}_{\mathbf{k}+\mathbf{2}}} = -\frac{1}{\mathbf{x}_2} + \left(\frac{1}{\mathbf{x}_2}\right)^2 \cdot 2 - \frac{1^2}{2} \cdot \frac{1}{-1/2} + m_{11} = 3/2.$$

Sabemos que a entrada a é $-1/\mathbf{x}_{\mathbf{k}-1}$. Após a diagonalização dos vértices **2** e **3**, a expressão a^2c se torna $1/2^2 \cdot 2$, visto que $a = -1/\mathbf{x}_2$ e $c = m_{44}$. Como não houve vértices além de $\mathbf{k}+\mathbf{2}$, não foi necessário realizar a soma das parcelas obtidas da diagonalização de outros vértices, e a expressão se torna somente $\frac{(a-1)^2}{x_5}$. Isso conclui o caso 1b. Agora que já verificamos todos os eventos para ciclos isolados, vamos agora considerar grafos cactos mais gerais.

Até agora, examinamos os casos em que há ciclos isolados e seus possíveis desdobramentos. Agora, vamos olhar para situações em que podemos ter árvores ou ciclos anexados a um ciclo central. Para facilitar a compreensão desses casos, utilizaremos a decomposição em blocos mencionada no capítulo 2. A seguir, apresentamos um cenário no qual um vértice de corte i está presente nos blocos **A**, **B** e **C**. Esses blocos podem ser compostos de ciclos ou arestas. Na Figura 6.7, apresentamos um grafo que possui três estruturas, em que o vértice i é um vértice de corte. Junto à primeira ilustração, apresentamos a árvore de blocos e uma decomposição arbórea (que não é considerada agradável) antes da conexão com o ciclo central, que inclui o vértice **i**.

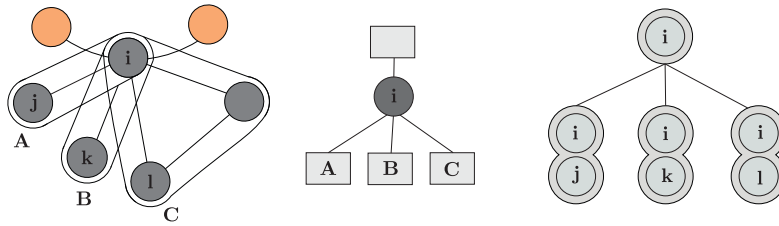


Figura 6.7: Subgrafos anexos ao vértice de corte \mathbf{i} , bem como a sua representação em blocos, seguida de uma decomposição arbórea.

Para cada bloco ligado ao vértice \mathbf{i} , podemos ter estruturas que precisam ser incorporadas no ciclo central. Suponha que o vértice \mathbf{j} seja o último vértice esquecido no bloco A , e que \mathbf{k} e \mathbf{l} sejam os últimos vértices esquecidos nos blocos B e C , respectivamente. Note que podemos ter a diagonalização desses vértices quando eles foram esquecidos ou não. Caso a primeira situação ocorra, as caixas no nó que contém somente \mathbf{i} possuem um único vértice, o vértice \mathbf{i} .

$$\mathbf{v} \begin{bmatrix} \mathbf{v} \\ *A \end{bmatrix} \quad \mathbf{v} \begin{bmatrix} \mathbf{v} \\ *B \end{bmatrix} \quad \mathbf{v} \begin{bmatrix} \mathbf{v} \\ *C \end{bmatrix}.$$

Por outro lado, é possível que algum vértice que está ligado ao vértice \mathbf{i} não tenha sido diagonalizado quando foi esquecido, resultando na transmissão de uma caixa que contém um vértice de tipo I para os próximos nó da árvore. Suponha que isso ocorreu no bloco A , quando o último vértice a fazer conexão com o vértice de corte \mathbf{i} , o vértice \mathbf{j} , foi esquecido.

$$\begin{matrix} & \mathbf{j} & \mathbf{i} \\ \mathbf{j} & \begin{bmatrix} 0 & \Delta \end{bmatrix} \\ \mathbf{i} & \begin{bmatrix} \Delta & * \end{bmatrix} \end{matrix}$$

Antes de incorporar as informações das estruturas anexas ao vértice \mathbf{i} , é necessário simplificá-las para facilitar o procedimento no ciclo central. As caixas utilizadas para realizar a junção podem ter um dos dois tipos abordados anterior-

mente. Se todas as caixas possuírem somente o vértice \mathbf{v} , então o procedimento de junção soma as entradas de uma linha ou coluna rotulada por \mathbf{v} de cada caixa.

$$\mathbf{v} \left[\begin{array}{c} \mathbf{v} \\ *A + *B + \dots \end{array} \right]$$

Podemos ter o caso em que há mistura dos dois tipos de caixas. Suponha que pelo menos duas caixas possuem dois vértices que se tornaram do tipo I, sendo eles os vértices \mathbf{j} e \mathbf{k} que vieram dos blocos A e B , respectivamente. As caixas possuem os seguintes formatos:

$$\mathbf{j} \left[\begin{array}{c|c} \mathbf{j} & \mathbf{i} \\ \hline 0 & *A \\ \hline *A & \Delta_A \end{array} \right] \quad \mathbf{k} \left[\begin{array}{c|c} \mathbf{k} & \mathbf{i} \\ \hline 0 & *B \\ \hline *B & \Delta_B \end{array} \right]$$

Podemos considerar que os blocos A e B tanto podem ser uma aresta quanto um ciclo. De qualquer forma, as caixas teriam o mesmo formato caso o vértice \mathbf{j} ou \mathbf{k} não fosse diagonalizado quando esquecido. O procedimento de junção dos nós que contêm o vértice \mathbf{i} consiste em empilhar as linhas do tipo I e somar as entradas rotuladas pelo vértice \mathbf{i} . Ao realizar a junção das caixas que possuem \mathbf{j} e \mathbf{k} como linha do tipo I, a forma escalonada na submatriz superior não é obtida, sendo necessário realizar operações nas linhas e colunas para manter a caixa no formato padrão. A sequência de caixas a seguir ilustra essa necessidade de operações:

$$\begin{array}{c} \mathbf{j} \quad \mathbf{k} \quad \mathbf{i} \\ \mathbf{j} \left[\begin{array}{c|c|c} 0 & 0 & *A \\ \hline 0 & 0 & *B \\ \hline *A & *B & \Delta_A + \Delta_B \end{array} \right] \longrightarrow \mathbf{k} \left[\begin{array}{c|c|c} \mathbf{j} & \mathbf{k} & \mathbf{i} \\ \hline 0 & 0 & *A \\ \hline 0 & 0 & 0 \\ \hline *A & 0 & \Delta_A + \Delta_B \end{array} \right] \longrightarrow \mathbf{j} \left[\begin{array}{c|c} \mathbf{j} & \mathbf{i} \\ \hline 0 & *A \\ \hline *A & \Delta_A + \Delta_B \end{array} \right] \end{array}$$

Após a diagonalização do vértice \mathbf{k} com o par $(\mathbf{k}, 0)$, a caixa resultante mantém o mesmo formato das caixas inicializadas no processo. Caso haja mais caixas com linhas do tipo I, esse processo pode ser repetido até que reste no máximo uma caixa com este formato. Ao final do processo, teremos uma caixa com uma linha do

tipo I e o vértice \mathbf{i} . As caixas que possuíam somente o vértice \mathbf{i} são agrupadas pelo caso anterior e incorporadas na entrada diagonal do mesmo vértice.

Neste ponto, as estruturas anexas ao vértice \mathbf{i} do ciclo foram reduzidas a apenas um tipo de caixa que carrega todas as informações antes de incorporá-las no ciclo. Esta etapa pode ser repetida para todos os vértices presentes no ciclo.

Antes de ser esquecido, cada vértice do ciclo resgatará a caixa produzida pelas estruturas anexas a ele. Essa caixa pode conter apenas o próprio vértice ou, ainda, o vértice e mais um vértice do tipo I. No caso em que todas as caixas produzidas pelas estruturas anexas ao vértice do ciclo central não contiverem linhas do tipo I, é possível processar o ciclo como o caso 1a ou 1b, incorporando o valor das estruturas no esquecimento do vértice.

Para fins de análise, dividiremos nossos casos em dois: caso 2a e caso 2b. No caso 2a, analisaremos a primeira ocorrência de uma caixa com vértices do tipo I, originária de uma estrutura anexa a um vértice. No caso 2b, examinaremos a situação em que um vértice do ciclo não foi diagonalizado e, no vértice subsequente, temos uma caixa originária da estrutura com vértices do tipo I.

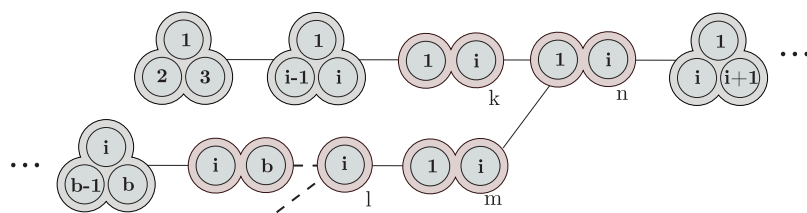


Figura 6.8: Junção de estrutura anexa a um vértice \mathbf{i} do ciclo.

No primeiro caso, o vértice $\mathbf{i-1}$ foi diagonalizado e repassou uma caixa N_k com dois vértices: $\mathbf{1}$ e \mathbf{i} . Ao juntar essa caixa com N_m , que contém o vértice \mathbf{i} e um vértice do tipo I (digamos, \mathbf{b}), obtemos a seguinte caixa de junção pelo algoritmo:

$$N_k = \begin{array}{c} \mathbf{1} \\ \mathbf{i} \end{array} \begin{array}{c|c} \mathbf{1} & \mathbf{i} \\ \hline * & *_{1i} \\ \hline *_{1i} & *_{\mathbf{i}}^{\mathbf{1}} \end{array} \quad
N_m = \begin{array}{c} \mathbf{b} \\ \mathbf{i} \\ \mathbf{1} \end{array} \begin{array}{c|c|c} \mathbf{b} & \mathbf{i} & \mathbf{1} \\ \hline 0 & *_{bi} & 0 \\ \hline *_{bi} & *_{\mathbf{b}}^{\mathbf{i}} & 0 \\ \hline 0 & 0 & 0 \end{array} \quad
N_n = \begin{array}{c} \mathbf{b} \\ \mathbf{i} \\ \mathbf{1} \end{array} \begin{array}{c|c|c} \mathbf{b} & \mathbf{i} & \mathbf{1} \\ \hline 0 & *_{bi} & 0 \\ \hline *_{bi} & *_{\mathbf{b}}^{\mathbf{i}} & *_{1b} \\ \hline 0 & *_{1b} & * \end{array}$$

A caixa N_k é gerada durante o processamento do ciclo, visto que o vértice $\mathbf{i}-\mathbf{1}$ foi diagonalizado. Já a caixa N_m é proveniente da estrutura anexa ao vértice \mathbf{i} , que possui um vértice do tipo I. Por fim, a caixa N_n é resultante da junção de ambas as caixas. A caixa N_n encontra-se no formato requerido pelo algoritmo e, portanto, nenhuma operação adicional é realizada, deixando a caixa N_n como herança para o próximo nó em \mathcal{T} . Nos próximos passos, adicionamos o próximo vértice do ciclo C e esquecemos o vértice \mathbf{i} . Vale ressaltar que, caso não exista o vértice $\mathbf{i}+\mathbf{1}$, o vértice \mathbf{i} é esquecido. Todas as estruturas anexas ao vértice $\mathbf{i}+\mathbf{1}$, se houver, são incorporadas após o esquecimento do vértice \mathbf{i} na árvore \mathcal{T} . Portanto, ao esquecer o vértice \mathbf{i} , obtemos a transição das caixas representadas:

$$\begin{array}{c} \mathbf{b} \\ \mathbf{i} \\ \mathbf{1} \\ \mathbf{i}+\mathbf{1} \end{array} \begin{array}{c|c|c|c} \mathbf{b} & \mathbf{i} & \mathbf{1} & \mathbf{i}+\mathbf{1} \\ \hline 0 & *_{bi} & 0 & 0 \\ \hline *_{bi} & *_{\mathbf{b}}^{\mathbf{i}} & *_{1b} & 0 \\ \hline 0 & *_{1b} & * & 0 \\ \hline 0 & 0 & 0 & 0 \end{array} \longrightarrow \begin{array}{c} \mathbf{i} \\ \mathbf{b} \\ \mathbf{1} \\ \mathbf{i}+\mathbf{1} \end{array} \begin{array}{c|c|c|c} \mathbf{i} & \mathbf{b} & \mathbf{1} & \mathbf{i}+\mathbf{1} \\ \hline \mathbf{X}_{\mathbf{i}} & *_{\mathbf{b}} & *_{1b} & \mathbf{1} \\ \hline *_{bi} & 0 & 0 & 0 \\ \hline *_{1b} & 0 & * & 0 \\ \hline 1 & 0 & 0 & 0 \end{array} \longrightarrow \begin{array}{c} \mathbf{1} \\ \mathbf{i}+\mathbf{1} \end{array} \begin{array}{c|c} \mathbf{1} & \mathbf{i}+\mathbf{1} \\ \hline * & 0 \\ \hline 0 & 0 \end{array}$$

Caso o vértice \mathbf{i} seja o último vértice do ciclo, o esquecimento dele geraria o mesmo processo de diagonalização, restando somente a entrada relacionada ao vértice $\mathbf{1}$. As operações do *Algoritmo Diagonal Arbórea* diagonalizam os vértices \mathbf{b} e \mathbf{i} com os pares positivo e negativo, respectivamente, e não afetam as entradas da submatriz rotulada por $\mathbf{1}$ e $\mathbf{i}+\mathbf{1}$. Qualquer operação realizada após a introdução do próximo vértice não é capaz de afetar a entrada $\mathbf{1}$. Portanto, o valor da entrada $\mathbf{1}$ é fixado até o final do processo, quando ele é esquecido. A seguir, apresentamos uma representação da estrutura no grafo após a diagonalização dos vértices \mathbf{i} e \mathbf{b} .

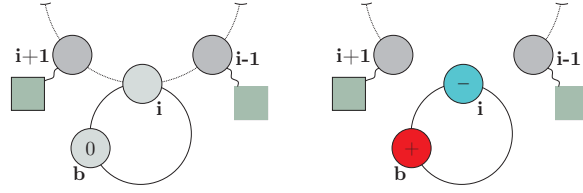


Figura 6.9: Diagonalização com algum filho nulo para o vértice \mathbf{i} do ciclo C .

No último caso que abordaremos, ocorre quando o nó k esquece o vértice $\mathbf{i-1}$ e não o diagonaliza. As caixas N_k e N_m nesse ponto possuem linhas do tipo I.

$$N_k = \begin{array}{c} \mathbf{i-1} \\ \mathbf{1} \\ \mathbf{i} \end{array} \begin{array}{c|cc} \mathbf{i-1} & \mathbf{1} & \mathbf{i} \\ \hline 0 & *_{1\mathbf{i-1}} & 1 \\ \hline *_{1\mathbf{i-1}} & * & 0 \\ \hline 1 & 0 & 0 \end{array} \quad N_m = \begin{array}{c} \mathbf{b} \\ \mathbf{i} \\ \mathbf{1} \end{array} \begin{array}{c|cc} \mathbf{b} & \mathbf{i} & \mathbf{1} \\ \hline 0 & *_{b\mathbf{i}} & 0 \\ \hline *_{b\mathbf{i}} & *_{b\mathbf{i}}^{\mathbf{i}} & 0 \\ \hline 0 & 0 & 0 \end{array}$$

A caixa N_n faz a fusão das caixas N_m e N_k obtendo uma nova caixa no formato padrão do *Algoritmo Diagonal Arbórea* e nenhuma operação é realizada.

$$\begin{array}{c} \mathbf{i-1} \\ \mathbf{b} \\ \mathbf{1} \\ \mathbf{i} \end{array} \begin{array}{c|cc|cc} \mathbf{i-1} & \mathbf{b} & \mathbf{1} & \mathbf{i} \\ \hline 0 & 0 & *_{1\mathbf{i-1}} & 1 \\ \hline 0 & 0 & 0 & *_{b\mathbf{i}} \\ \hline *_{1\mathbf{i-1}} & 0 & * & 0 \\ \hline 1 & *_{b\mathbf{i}} & 0 & *_{b\mathbf{i}}^{\mathbf{i}} \end{array}$$

Os vértices $\mathbf{i-1}$ e \mathbf{b} só poderão ser processados após o esquecimento de outro vértice. Se \mathbf{i} for o último vértice do ciclo C que se conecta com $\mathbf{1}$, ao esquecê-lo, obtemos a diagonalização dos vértices \mathbf{i} e \mathbf{b} , restando apenas a submatriz rotulada por $\mathbf{i-1}$ e $\mathbf{1}$.

$$\begin{array}{c} \mathbf{i-1} \\ \mathbf{1} \end{array} \begin{array}{c|c} \mathbf{i-1} & \mathbf{1} \\ \hline 0 & *_{1\mathbf{i-1}} \\ \hline *_{1\mathbf{i-1}} & *_{b\mathbf{i}}^{\mathbf{i}} \end{array}$$

Note que as operações realizadas não afetam as entradas da submatriz restante. É importante mencionar que, antes de realizar o esquecimento, devemos

adicionar a caixa resultante das estruturas que estavam penduradas no nó **1**, se houver. Caso seja somente o vértice **1** presente na caixa, sua entrada é incorporada na matriz. Se houver uma caixa com um vértice do tipo I, esse vértice é diagonalizado com valor nulo, seguindo um processo semelhante ao de simplificação das caixas provenientes dos blocos. Em seguida, ao esquecer o vértice **1**, obtemos a diagonalização dos vértices **1** e **i-1**, com valor positivo e negativo respectivamente.

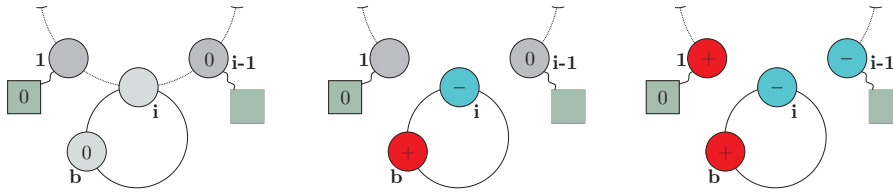


Figura 6.10: Processamento com filho e vizinho de **i** nulos.

Por outro lado, como o vértice **i** não se conecta com o vértice inicial do ciclo, é necessário adicionar o vértice **i+1** antes de realizar o esquecimento do vértice **i**. O processo de introdução do nó **i+1** e o esquecimento do vértice **i** são apresentados na primeira caixa abaixo. A caixa da direita exibe o resultado da diagonalização dos vértices **i** e **b** no processo.

$$\begin{array}{c}
 \begin{array}{c}
 \mathbf{i} \\
 \mathbf{i-1} \\
 \mathbf{b} \\
 \mathbf{1} \\
 \mathbf{i+1}
 \end{array}
 \begin{array}{c}
 \mathbf{i} \quad \mathbf{i-1} \quad \mathbf{b} \quad \mathbf{1} \quad \mathbf{i+1} \\
 \left[\begin{array}{c|c|c|c|c}
 ? & 1 & *_{bi} & 1 & 1 \\
 1 & 0 & 0 & *_{1i-1} & 0 \\
 *_{bi} & 0 & 0 & 0 & 0 \\
 1 & *_{1i-1} & 0 & * & 0 \\
 1 & 0 & 0 & 0 & 0
 \end{array} \right]
 \end{array}
 \longrightarrow
 \begin{array}{c}
 \mathbf{i-1} \\
 \mathbf{1} \\
 \mathbf{i+1}
 \end{array}
 \begin{array}{c}
 \mathbf{i-1} \quad \mathbf{1} \quad \mathbf{i+1} \\
 \left[\begin{array}{c|c|c}
 0 & *_{1i-1} & 0 \\
 *_{1i-1} & * & 0 \\
 0 & 0 & 0
 \end{array} \right]
 \end{array}
 \end{array}$$

O algoritmo diagonaliza os vértices **i** e **b** com valores positivo e negativo, respectivamente, e mantém na caixa os vértices **1**, **i+1** e **i-1**. Nesse ponto, não há conexões entre os vértices **i+1** e **1**, portanto, a entrada não é atualizada e podemos processar o ciclo como um caminho sem reatualizar a entrada **1**. Além disso, se houver uma caixa com vértice do tipo I anexa ao vértice **1**, este vértice é

diagonalizado com o valor zero do mesmo modo que a situação anterior. Ao final do processo, o vértice $i-1$ e 1 são processados com valor positivo e negativo. Na estrutura abaixo podemos observar como ocorre essa situação no grafo:

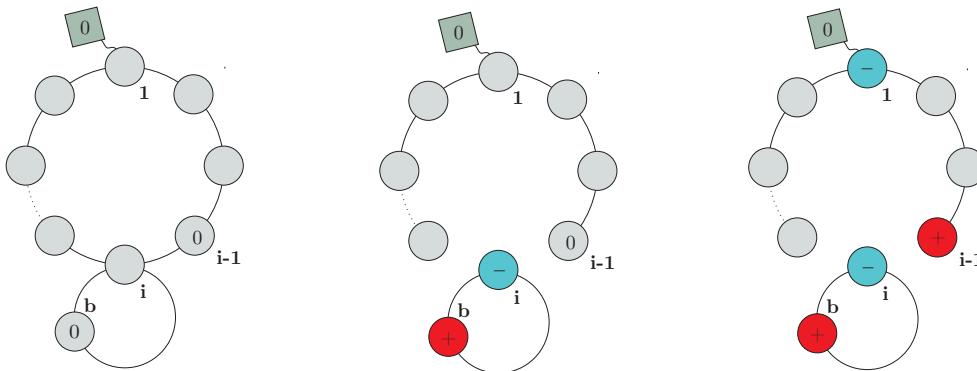


Figura 6.11: Processamento com filho e vizinho de i nulos com $i \neq b$.

Um fato interessante é que em qualquer grafo cactos em que haja uma estrutura anexada a um vértice principal, a entrada relacionada ao início do ciclo não precisa ser resgatada ou calculada, já que ela é diagonalizada em todos os casos com mais algum vértice do tipo I. Isso significa que todos os casos para grafos cactos foram abordados e sabemos como realizar o processamento desta classe de grafos.

Agora, podemos explorar uma aplicação prática para esta classe de grafos, onde o *Algoritmo Diagonal Arbórea* é uma ferramenta capaz de realizar a diagonalização de grafos cactos com facilidade.

6.2 Grafos Cactos Reflexivos

A literatura tem estudado estimativas superiores e inferiores para o segundo maior autovalor da matriz de adjacência de um grafo G sob várias restrições. Diversos resultados fornecem cotas baseadas no número de vértices e arestas de G . Em 1984, Brigham e Dutton [5] estabeleceram a desigualdade $\lambda_2(G) \leq \sqrt{\frac{m(n-2)}{n}}$ para $\lambda_2(G)$. Powers [27] demonstrou em 1986 que $-1 \leq \lambda_2(G) \leq \frac{n}{2} - 1$. Cotas

mais precisas têm sido investigadas em detalhes para diferentes classes de grafos. Cao e Yuan [6] determinaram em 1993 todos os grafos conexos G com $\lambda_2(G) \leq \frac{1}{3}$. Cvetković e Simic [9] determinaram em 1995 os grafos com $\lambda_2(G) \leq \sqrt{2} - 1$. Grafos que satisfazem $\lambda_2 \leq r$ para $r \in \{\frac{\sqrt{5}-1}{2}, \frac{\sqrt{5}+1}{2}, \sqrt{2}, \sqrt{3}, 2\}$ foram investigados em diversos artigos. Neste trabalho, nosso objeto de estudo são os grafos que satisfazem a propriedade $\lambda_2 \leq 2$, também conhecidos como *reflexivos*.

Nos últimos anos, a pesquisa sobre grafos que respeitam a desigualdade $\lambda_2 \leq 2$ tem recebido considerável destaque, abrangendo diversas classes específicas de grafos, entre elas os grafos cacto. Um cacto G é considerado reflexivo se e somente se $\lambda_2(G) \leq 2$, onde λ_2 representa o segundo maior autovalor do grafo. O foco desta classe foi a caracterização dos cactos que não formam um *pacote*. Em termos formais, um *pacote* é um cacto em que todos os seus ciclos compartilham um vértice em comum. O objetivo é caracterizar a classe dos grafos cactos reflexivos maximais em relação à propriedade $\lambda_2 \leq 2$ ser *hereditária*. Em termos mais simples, os grafos cactos reflexivos maximais são aqueles que não podem ser ampliados pela adição de mais arestas ou vértices sem violar a propriedade $\lambda_2 \leq 2$.

Em 1982, Neumaier determinou todas as árvores reflexivas em [25]. No caso de grafos cactos, em 1996, foi realizada a caracterização de todos os grafos cactos bicíclicos [30], seguida das caracterizações de grafos cactos reflexivos com três ciclos [24], quatro ciclos [28], e para cactos com mais de quatro ciclos [29] que não formam um pacote. No entanto, os grafos cactos reflexivos que formam um pacote ainda não foram caracterizados. A seguir, apresentaremos a demonstração de um dos principais teoremas que é conhecido para grafos cactos que formam um pacote, utilizando o *Algoritmo Diagonal Arbórea*. Para isso, definiremos um grupo de grafos conhecido como grafos Smith.

Definição 6.2. (Smith) Dizemos que um grafo G é um grafo Smith se ele satisfaz a propriedade $\lambda_1(G) = 2$, onde $\lambda_1(G)$ representa o maior autovalor da matriz de adjacência do grafo.

Teorema 6.1. *A condição necessária e suficiente para que um grafo G satisfaça $\lambda_1(G) \leq 2$ é que cada componente de G seja um subgrafo de um dos grafos apresentados na Figura 6.12, os quais têm $\lambda_1(G) = 2$.*

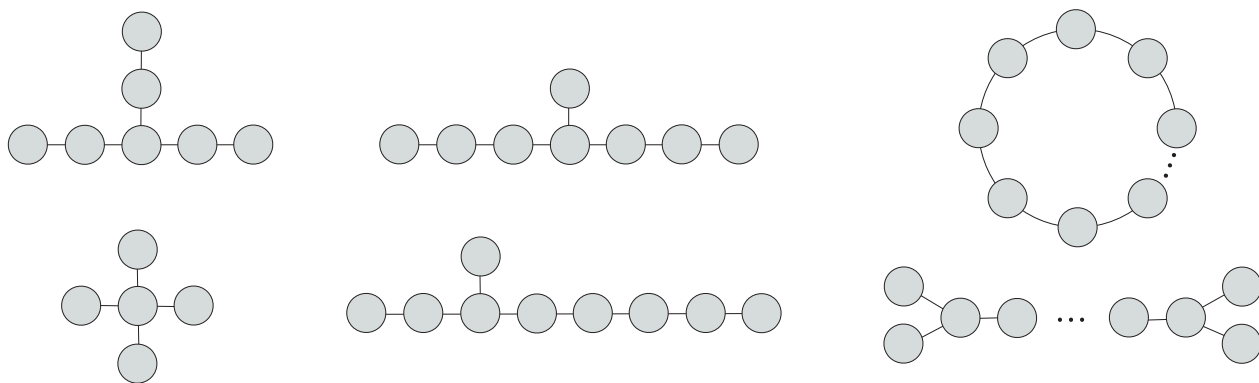


Figura 6.12: Os grafos Smith.

A Figura 6.13 ilustra a estrutura de um grafo cacto que forma um pacote, onde os ciclos são conectados em um vértice central (denotado por \mathbf{u}) e podem haver árvores anexadas aos vértices dos ciclos, inclusive ao vértice central.

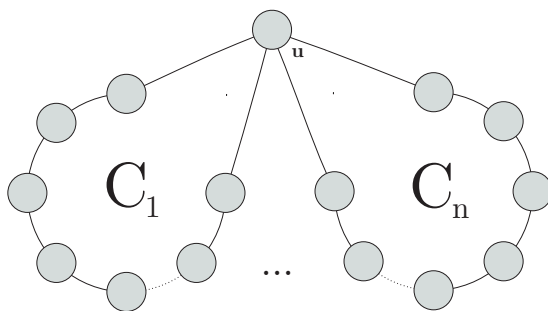


Figura 6.13: Grafo cactos que forma um pacote.

O Teorema 6.2 foi inicialmente apresentado por [30] para caracterizar grafos bicíclicos que são reflexivos. Mais tarde, o uso desse teorema foi uma peça importante para caracterizar famílias mais gerais de cactos. A prova envolveu mecanismos algébricos e, neste trabalho, apresentaremos uma demonstração usando o *Algoritmo Diagonal Arbórea*.

Teorema 6.2. *Seja G um grafo cacto que forma um pacote conforme a Figura 6.13, onde u é um vértice de corte.*

1. *Se pelo menos duas componentes de $G - u$ são supergrafos de grafos Smith, e pelo menos um deles é supergrafo próprio, então $\lambda_2(G) > 2$.*
2. *Se pelo menos duas componentes de $G - u$ são grafos Smith, e as demais são subgrafos de grafos Smith, então $\lambda_2(G) = 2$.*
3. *Se no máximo uma componente de $G - u$ é um grafo Smith e as demais são subgrafos próprios de grafos Smith, então $\lambda_2(G) < 2$.*

Demonstração. Para ilustrar cada item do enunciado, será utilizado o diagrama abaixo, que representa uma decomposição em blocos de um grafo cacto G formando um pacote. O vértice u é um vértice de corte que está presente em todos os blocos adjacentes. Para demonstrar os casos do teorema, será considerada a subárvore de cada bloco como uma componente.

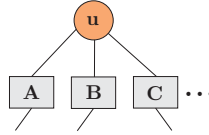


Figura 6.14: Representação por blocos de um grafo cacto que forma um pacote.

Para provar o primeiro item, suponhamos que existam pelo menos dois blocos adjacentes ao vértice de corte u , cujas subárvores com raiz neles formam dois supergrafos próprios de um grafo Smith. Esses blocos são denotados como A e B . Em seguida, aplicamos o *Algoritmo Diagonal Arbórea* com a matriz de entrada $M = A(G) - 2I$ durante o processamento. Como as componentes possuem um número de vértices maior do que um grafo Smith, pelo Teorema 2.3 temos a garantia que essa operação resultará em um valor positivo e outro negativo para pelo menos dois vértices em cada bloco. Essa condição é suficiente para garantir que $\lambda_2(G) > 2$, mesmo sem ter processado o vértice u e os demais blocos, se existirem.

Agora, suponhamos que a subárvore formada com raiz no bloco A seja um supergrafo próprio de Smith. Ao processarmos o bloco A com o *Algoritmo Diagonal Arbórea*, podemos garantir pelo menos um valor positivo. É possível que a componente do bloco A seja grande o suficiente para obtermos mais de um valor positivo. No entanto, ao processar as outras subárvores enraizadas nos blocos de \mathbf{u} , é possível que vértices do tipo I apareçam em pelo menos uma estrutura que é um supergrafo de Smith. Como pode haver mais de uma estrutura desse tipo, cada vértice que se tornar do tipo I é diagonalizado com valor nulo até que reste apenas uma caixa no processo de junção da decomposição. Caso contrário, as caixas rotuladas por \mathbf{u} são incorporadas antes de seu esquecimento. Esse processo é realizado da mesma maneira que foi apresentado na simplificação das estruturas anexas a um vértice do ciclo. No final do processo, teremos uma única caixa da forma:

$$\begin{array}{c} \mathbf{v} \quad \mathbf{u} \\ \mathbf{v} \left[\begin{array}{c|c} 0 & * \\ \hline * & \Delta \end{array} \right]. \\ \mathbf{u} \end{array}$$

O vértice \mathbf{v} é proveniente de algum vértice que tornou-se do tipo I. Ao esquecer o vértice \mathbf{u} ocorre a diagonalização do vértice \mathbf{v} e \mathbf{u} onde ocorre a aparição do segundo valor positivo, na execução do Algoritmo, e portanto $\lambda_2(G) > 2$.

Na situação em que temos pelo menos duas subárvores de dois blocos do tipo Grafo Smith, a junção dos blocos anexas ao vértice \mathbf{u} que são subgrafos Smith antes de processarmos \mathbf{u} segue o processo de simplificação unindo as caixas rotuladas por \mathbf{u} . Sendo assim, todas as caixas que possuem vértices do tipo I são juntadas até restar uma única caixa com um vértice do tipo I e o vértice \mathbf{u} . As demais componentes dos outros blocos contribuirão com um valor na entrada de \mathbf{u} , e os blocos que possuíam linha do tipo I foram diagonalizados com um valor nulo. Neste ponto, todas as entradas foram diagonalizadas com valor negativo ou nulo, restando somente os vértices \mathbf{u} e um vértice do tipo I, digamos \mathbf{v} , obtendo a caixa

no formato:

$$\begin{array}{c} \mathbf{v} \quad \mathbf{u} \\ \mathbf{v} \left[\begin{array}{c|c} 0 & * \\ \hline * & \Delta \end{array} \right] \\ \mathbf{u} \end{array}.$$

Ao esquecer o vértice \mathbf{u} , diagonalizamos os vértices \mathbf{u} e \mathbf{v} , obtendo um valor positivo e um valor negativo. Como resultado, temos valores negativos ou nulos em cada subárvore, e apenas no final obtemos um valor positivo. Esses fatos garantem que o autovalor $\lambda_2(G)$ do grafo é igual a 2.

Por fim, se todas as subárvores enraizadas nos blocos filhos de \mathbf{u} forem subgrafos próprios de Smith, realizamos a junção das caixas incorporando na entrada de \mathbf{u} . A caixa obtida a partir do bloco que possui subgrafo de Smith apresenta o seguinte formato:

$$\begin{array}{c} \mathbf{v} \quad \mathbf{u} \\ \mathbf{v} \left[\begin{array}{c|c} 0 & * \\ \hline * & \Delta \end{array} \right] \\ \mathbf{u} \end{array}.$$

Após incorporar as subárvores dos blocos filhos de \mathbf{u} que são subgrafos próprios na entrada \mathbf{u} seguindo o processo de simplificação, todos os valores de cada bloco foram diagonalizados com valores negativos. Em seguida, o algoritmo arbóreo diagonaliza a caixa final obtendo um valor positivo e negativo. Dessa forma, obtemos apenas um valor positivo, enquanto os demais são negativos, o que implica que $\lambda_2(G) < 2$.

□

Assim, o Teorema 6.2 resolve uma ampla classe de grafos cactos reflexivos, mas não é capaz de dar respostas precisas quando tivermos um bloco com um supergrafo próprio Smith e o restante das estruturas são subgrafos Smith. Além disso, o problema de encontrar todos os grafos cactos reflexivos maximais não está completamente resolvido. Embora existam resultados para grafos cactos que não

formam um pacote e para aqueles em que o teorema não pode ser aplicado, o caso em que todos os ciclos têm um vértice comum é especialmente difícil, uma vez que o número de ciclos anexados a \mathbf{u} não é limitado. Além disso, é possível ter árvores penduradas em alguns vértices do ciclo e em \mathbf{u} .

Esperávamos obter resultados utilizando o *Algoritmo Diagonal Arbórea* em casos não cobertos pelo teorema. No entanto, encontramos dificuldades principalmente quando ocorre uma mudança de sinal no bloco que possui um supergrafo próprio de Smith. Essa situação pode ocorrer devido às possíveis estruturas anexadas ao vértice \mathbf{u} e às próprias estruturas anexas no vértice do ciclo, onde a entrada u pode receber contribuições positivas ou negativas.

O Teorema 2.3 garante que, se encontrarmos duas estruturas Smith desconexas em um bloco, já obtemos a violação da propriedade de um grafo cacto reflexivo. Portanto, esses casos são descartados naturalmente. No entanto, a busca por caracterizar grafos cactos reflexivos que formam um pacote é um problema difícil, pois geralmente não depende apenas da componente que é um supergrafo Smith, mas também da relação estreita com as estruturas anexas ao vértice \mathbf{u} . O *Algoritmo Diagonal Arbórea* é uma ferramenta valiosa para identificar grafos reflexivos em grafos cactos, pois é capaz de processar o grafo de forma linear devido à largura limitada do grafo cactos. Além disso, a utilização do algoritmo permite determinar propriedades e características específicas de cada vértice e das estruturas anexas aos vértices. Isso possibilita a identificação de padrões que podem ajudar a entender a estrutura dos grafos cactos reflexivos. No entanto, a caracterização da subclasse de grafos cactos reflexivos que formam um pacote ainda é um problema em aberto. Portanto, embora o algoritmo seja capaz de dar respostas precisas sobre a reflexividade de um grafo cactos quando executado, não é capaz de caracterizar completamente essa subclasse faltante.

7 CONSIDERAÇÕES FINAIS

Durante esta dissertação de mestrado, diversos tópicos relacionados à teoria espectral de grafos foram abordados. Todos esses tópicos tiveram como base a análise do comportamento do *Algoritmo Diagonal Arbórea* em determinadas classes de grafos, bem como a compreensão detalhada de seu funcionamento. Uma das principais motivações foi apresentar de maneira clara e intuitiva o procedimento executado pelo *Algoritmo Diagonal Arbórea*, demonstrando suas operações em detalhes a cada passo. Para sua execução, tornou-se necessário a definição de conceitos fundamentais como decomposição arbórea e largura arbórea, que são requisitos e parâmetros para a execução do algoritmo.

No Capítulo 3, foram abordadas propriedades e teoremas clássicos relacionados à decomposição arbórea, bem como cotas para a largura arbórea de algumas classes de grafos. Também foi apresentado o conceito de decomposição agradável de um grafo e demonstrado como é possível transformar uma decomposição arbórea em uma decomposição agradável sem aumentar a largura da decomposição.

Um dos objetivos deste trabalho foi analisar e comparar o desempenho do *Algoritmo Diagonal Arbórea* com o *Algoritmo de Árvore* desenvolvido em [19]. Enquanto o *Algoritmo Diagonal Arbórea* é uma técnica geral utilizada para resolver uma ampla classe de grafos, o *Algoritmo de Árvore* é projetado para solucionar especificamente o problema de grafos que são árvores. Buscou-se identificar semelhanças e diferenças entre essas duas abordagens, bem como suas respectivas eficiências em diferentes cenários. Como resultado dessa análise, foi definida uma sequência para a construção da decomposição arbórea \mathcal{T} de modo que os vértices que eram esquecidos no *Algoritmo Diagonal Arbórea* fossem processados na mesma ordem do *Algoritmo de Árvore*. Observou-se que ambos os algoritmos produzem os mesmos valores nos nós dos vértices, exceto por cancelamentos acidentais onde os valores são diferentes (-1 e 1 no *Algoritmo Diagonal Arbórea* e $-1/2$ e 2 no *Algoritmo de Árvore*). Esta di-

ferença não afeta a conclusão da saída do algoritmo, uma vez que ambos algoritmos são baseados na inércia da matriz.

No Capítulo 6, abordamos como o *Algoritmo Diagonal Arbórea* diagonaliza uma matriz M de um grafo que é um cacto. Uma das motivações para estudar essa classe foi a possibilidade de resolver um problema em aberto sobre grafos cactos reflexivos, no qual uma subclasse ainda permanecia não caracterizada. Os grafos cactos são definidos como grafos em que quaisquer dois ciclos têm no máximo um vértice em comum. Portanto, naturalmente, realizamos a análise dos casos em que o grafo cactos é somente um ciclo e examinamos os possíveis cenários que surgem. Em seguida, examinamos uma situação mais geral de um cacto, onde os ciclos poderiam ter estruturas penduradas em seus vértices, como diversos ciclos e árvores. Para isso, mostramos como o *Algoritmo Diagonal Arbórea* se desenvolvia para um ciclo central. Utilizamos a decomposição em blocos para um grafo cactos arbitrário, simplificando as operações necessárias para iniciar o processamento do ciclo. Em seguida, iniciamos o processo de diagonalização do ciclo, considerando que cada vértice do ciclo poderia ter no máximo uma caixa que incorporava as estruturas que estavam anexadas àquele vértice. Para isso, analisamos todas as possibilidades de diagonalização do ciclo, considerando diferentes casos para as caixas dos vértices.

Utilizamos o estudo da performance do *Algoritmo Diagonal Arbórea* em grafos cactos para analisar a classe de grafos cactos reflexivos. A classe de cactos reflexivos já foi caracterizada para todos os grafos cactos que não formam um pacote, e que o Teorema 6.2 não pode ser aplicado. O Teorema 6.2 foi uma das ferramentas essenciais para determinar os grafos cactos reflexivos que não formavam um pacote. Nos dedicamos a demonstrá-lo em uma nova versão utilizando o *Algoritmo Diagonal Arbórea* para sua validação.

O desenvolvimento do *Algoritmo Diagonal Arbórea* representou uma importante generalização de algoritmos desenvolvidos para classes específicas de grafos. Dessa forma, pretendemos continuar explorando o funcionamento deste al-

goritmo em outras classes específicas, a fim de identificar problemas que possam ser solucionados por meio desta ferramenta. Além disso, continuamos a investigar o problema dos grafos cactos reflexivos, buscando determinar subclasses e soluções para esta questão que ainda permanece em aberto.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] ARNBORG, S., PROSKUROWSKI, A., AND CORNEIL, D. G. Forbidden minors characterization of partial 3-trees. *Discrete Mathematics* 80, 1 (1990), 1–19.
- [2] BECKER, C. O., PEQUITO, S., PAPPAS, G. J., AND PRECIADO, V. M. Network design for controllability metrics. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)* (2017), pp. 4193–4198.
- [3] BODLAENDER, H. L., AND KLOKS, T. A linear-time algorithm for finding tree-decompositions of small width. *Journal of Algorithms* 18, 3 (1995), 432–451.
- [4] BRAGA, R. O., RODRIGUES, V. M., AND TREVISAN, V. Locating eigenvalues of unicyclic graphs. *Applicable Analysis and Discrete Mathematics* 11, 2 (2017), 273–298.
- [5] BRIGHAM, R., AND DUTTON, R. Bounds on graph spectra. *Journal of Combinatorial Theory, Series B* 37, 3 (1984), 228–234.
- [6] CAO, D., AND YUAN, H. Graphs characterized by the second eigenvalue. *Journal of Graph Theory* 17, 3 (1993), 325–331.
- [7] COURCELLE, B., AND OLARIU, S. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics* 101, 1 (2000), 77–114.
- [8] CVETKOVIĆ, D., ROWLINSON, P., AND SIMIĆ, S. *An Introduction to the Theory of Graph Spectra*. London Mathematical Society Student Texts. Cambridge University Press, 2009.
- [9] CVETKOVIĆ, D., AND SIMIĆ, S. On graphs whose second largest eigenvalue does not exceed $(51)2$. *Discrete Mathematics* 138, 1 (1995), 213–227. 14th British Combinatorial Conference.

- [10] DIESTEL, R. *Graph Theory*, 5 ed. Springer Berlin, 2017.
- [11] FRITSCHER, E., HOPPEN, C., ROCHA, I., AND TREVISAN, V. On the sum of the laplacian eigenvalues of a tree. *Linear Algebra and its Applications* 435, 2 (2011), 371–399.
- [12] FÜRER, M., HOPPEN, C., AND TREVISAN, V. Efficient diagonalization of symmetric matrices associated with graphs of small treewidth. arXiv: 2109.02515.
- [13] FÜRER, M., HOPPEN, C., JACOBS, D. P., AND TREVISAN, V. Eigenvalue location in graphs of small clique-width. *Linear Algebra and its Applications* 560 (2019), 56–85.
- [14] HARARY, F. *Graph Theory*. Addison-Wesley, Reading, MA, 1994.
- [15] HARARY, F., AND UHLENBECK, G. E. On the number of husimi trees. *Proceedings of the National Academy of Sciences* 39, 4 (1953), 315–322.
- [16] HORN, R. A., AND JOHNSON, C. R. *Matrix Analysis*. Cambridge University Press, 1985.
- [17] HUSIMI, K. Note on Mayers’ Theory of Cluster Integrals. *The Journal of Chemical Physics* 18, 5 (12 2004), 682–684.
- [18] JACOBS, D. P., OLIVEIRA, E. R., AND TREVISAN, V. Most laplacian eigenvalues of a tree are small. *Journal of Combinatorial Theory, Series B* 146 (2021), 1–33.
- [19] JACOBS, D. P., AND TREVISAN, V. Locating the eigenvalues of trees. *Linear Algebra and its Applications* 434, 1 (2011), 81–88.
- [20] JACOBS, D. P., TREVISAN, V., AND TURA, F. Eigenvalue location in threshold graphs. *Linear Algebra and its Applications* 439, 10 (2013), 2762–2773.
- [21] JACOBS, D. P., TREVISAN, V., AND TURA, F. C. Eigenvalue location in cographs. *Discrete Applied Mathematics* 245 (2018), 220–235.

- [22] KLOKS, T. *Treewidth, Computations and Approximations*, vol. 842 of *Lecture Notes in Computer Science*. Springer, 1994.
- [23] MEYER, C. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial Mathematics, 2000.
- [24] MIHAILOVIĆ, B., AND RADOSAVLJEVIĆ, Z. On a class of tricyclic reflexive cactuses. *Publications of the Faculty of Electrical Engineering*, 16 (2005), 55–63.
- [25] NEUMAIER, A. The second largest eigenvalue of a tree. *Linear Algebra and its Applications* 46 (1982), 9–25.
- [26] OLIVEIRA, E. R., STEVANOVIĆ, D., AND TREVISAN, V. Spectral radius ordering of starlike trees. *Linear and Multilinear Algebra* 68, 5 (2020), 991–1000.
- [27] POWERS, V. Eigenvectors and eigenvalues of graphs. *Office of Naval Research* (1986), 1–7.
- [28] RADOSAVLJEVIĆ, Z., AND RAŠAJSKI, M. A class of reflexive cactuses with four cycles. *Publications of the Faculty of Electrical Engineering*, 14 (2003), 64–85.
- [29] RADOSAVLJEVIĆ, Z., AND RAŠAJSKI, M. Multicyclic treelike reflexive graphs. *Discrete Mathematics* 296, 1 (2005), 43–57.
- [30] RADOSAVLJEVIĆ, Z., AND SIMIĆ, S. Which bicyclic graphs are reflexive ? *Publications of the Faculty of Electrical Engineering*, 7 (1996), 90–104.
- [31] ROBERTSON, N., AND SEYMOUR, P. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B* 35, 1 (1983), 39–61.
- [32] ROBERTSON, N., AND SEYMOUR, P. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms* 7, 3 (1986), 309–322.

- [33] ROBERTSON, N., AND SEYMOUR, P. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B* 92, 2 (2004), 325–357. Special Issue Dedicated to Professor W.T. Tutte.
- [34] SPERB, R. P. On an eigenvalue problem arising in chemistry. *Z. Angew. Math. Phys.* 32, 4 (jul 1981), 450–463.
- [35] SYLVESTER, J. J. A demonstration of the theorem that every homogeneous quadratic polynomial is reducible by real orthogonal substitutions to the form of a sum of positive and negative squares. *Philosophical Magazine* 4th, 23 (1852), 138–142.
- [36] WAGNER, K. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen* 114, 1 (1937), 570–590.
- [37] WEST, D. B. *Introduction to Graph Theory*, 2 ed. Prentice-Hall, Englewood Cliffs, NJ, 2000.