UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

ELISÂNGELA SILVA DIAS

Reconhecimento polinomial de álgebras cluster de tipo finito

Goiânia
2015

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE TESE EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Reconhecimento polinomial de álgebras cluster de tipo finito

Autor(a): Elisângela Silva Dias

Goiânia, 09 de Setembro de 2015.

Elisângela Silva Dias – Autor

– Orientador

ELISÂNGELA SILVA DIAS

Reconhecimento polinomial de álgebras cluster de tipo finito

Tese apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Doutor em Programa de Pós-Graduação em Ciência da Computação.

Área de concentração: Representação de Álgebras e Temas Afins.

Orientadora: Profa. Dra. Diane Castonguay

Goiânia
2015

ELISÂNGELA SILVA DIAS

Reconhecimento polinomial de álgebras cluster de tipo finito

Tese defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Doutor em Ciência da Computação, aprovada em 09 de Setembro de 2015, pela Banca Examinadora constituída pelos professores:

Profa. Dra. Diane Castonguay
Instituto de Informática – UFG
Presidente da Banca

Prof. Dr. Ralf Schiffler
Department of Mathematics – UCONN

Prof. Dr. Mitre Costa Dourado
Instituto de Matemática – DCC – UFRJ

Prof. Dr. Marcelo Henriques de Carvalho
Faculdade de Computação – UFMS

Prof. Dr. Humberto José Longo
Instituto de Informática – UFG

Elisângela Silva Dias

Graduou-se em Sistemas de Informação na Universidade Estadual de Goiás (UEG). Durante sua graduação, trabalhou com atendimento a clientes, análise de sistemas e programação em Delphi e Visual Basic. Também foi monitora no programa do Governo Itinerante intitulado “Janelas Abertas”. Durante o mestrado, cursado na Universidade Federal de Goiás (UFG), foi professora substituta no Estado no Centro de Educação Profissional Sebastião de Siqueira (CEPSS) e no Instituto de Informática – UFG. Atualmente, é professora assistente no Instituto de Informática – UFG.

Aos meus pais, Sebastiana Silva Dias e Gercino Alves Dias *(in memorian)*, pelo incentivo, apoio, paciência, por tudo que sempre fizeram por mim e por nossa família e também pelo belo exemplo de vida que me deram, o qual me fez tentar ser uma pessoa melhor a cada dia. Papai, queria muito que você estivesse aqui.

Agradecimentos

Primeiramente a Deus, por ter permitido que eu começasse o doutorado e, principalmente, por ter me dado força e persistência para terminar. Também por tudo que Ele tem me concedido. Sem Ele na minha vida, eu não teria conseguido superar todas as lutas e dificuldades enfrentadas neste período.

Aos meus pais, Sebastiana Silva Dias e Gercino Alves Dias, pelos seus bons exemplos, bondade e caráter. Mesmo possuindo condições financeiras muito limitadas, nunca pouparam esforços em incentivar eu e meus irmãos a estudar e ter uma vida melhor. Lembro-me deles me contando que saíram da roça quando eu tinha quatro anos de idade para vir para Goiânia para conceder estudos para mim, que sou a filha mais velha. Espero ter honrado esse sacrifício.

A minha orientadora e amiga professora Dra. Diane Castonguay, por me aceitar nesta longa jornada de trabalho, por querer sempre mais de mim e por acreditar em mim, às vezes até mais que eu mesma. Também agradeço pelo constante apoio, pela orientação presente, paciente e dedicada, pela amizade sincera, pelas palavras francas, pelas broncas e pelas palavras de incentivo. Eu também agradeço por estar sempre ao meu lado, mesmo discordando de mim, e por sempre me mostrar e me ensinar a ver a vida de uma maneira diferente.

Ao meu marido, Victor Ribeiro Silva, pelo apoio, compreensão, companheirismo e pelo amor que me dedicou durante todo esse tempo, me fazendo querer ser uma pessoa melhor e me ajudando nas horas mais difíceis com tanto carinho e paciência.

Aos meus familiares, pelo constante apoio, incentivo, pelo amor que sempre me deu forças para continuar e pela compreensão nas horas de desespero e nas horas de mau humor, quando algo não dava certo. Agradeço pelas palavras de conforto, pela sabedoria e pelo carinho do meu irmão *Edilson Silva Dias* e pelas poucas palavras, mas muito afeto, seriedade e sinceridade, do meu irmão *Elenilson Silva Dias*. Vocês são únicos e muito especiais! Agradeço ao meu padrasto *Lindomar Magalhães Alves*, que sempre me tratou como filha, sempre mostrou seu orgulho por mim e sempre esteve ao nosso lado. Agradeço aos meus sogros, *Waldilene Silva Ribeiro* e *Eder Ribeiro*, pela receptividade, bom humor e paciência que sempre me dedicaram. Também pela boa comida e por serem como meus segundos pais. E, por último, mas não menos importantes, agradeço as minhas cunhadas

Laísa Ribeiro Silva de Abreu, *Lorranny Nunes de Souza Dias* e *Danielle Santos Costa*, pelo jeito bem humorado de ver a vida, pelo carinho, pelo companheirismo e por serem as irmãs que nunca tive. Eu amo muito, incondicionalmente, todos vocês!

A minha amiga Dra. Márcia Rodrigues Cappelle Santana, pelas longas horas de conversas, pelas palavras de incentivo, pelas palavras de alerta, pelos sábios conselhos, pelo apoio moral, pelos abraços e colo cedidos quando eu mais precisei, pela sensatez e pela franqueza. Enfim, agradeço por sua amizade incondicional e desinteressada e por sempre estar disposta a me ouvir e me ajudar.

A minha amiga Dra. Erika Morais Martins Coelho, por sua amizade presente, por me mostrar pontos de vista diferentes, pelas conversas sérias e também pelas inúteis, pelo apoio nos momentos difíceis, pelas palavras de incentivo, pelo carinho e pelos lanches divididos.

Ao professor Dr. Humberto José Longo, pelo ótimo trabalho em equipe que desenvolvemos e também paciência em me ensinar fórmulas, algoritmos, como colocar boas figuras, tabelas, letras caligráficas etc, e tirar todas as dúvidas referentes ao Latex. Obrigada por sua disponibilidade e disposição.

Às professoras Dra. Sonia Trepode e Dra. Claudia Alicia Chaio, pela receptividade demonstrada na Universidade Nacional de Mar del Plata. Foram elas que sugeriram o estudo do artigo que deu início aos estudos dos temas abordados nesta tese. Agradeço imensamente pelo carinho demonstrado e pela atenção dispensada a mim.

Ao professor Dr. Mitre Costa Dourado, que me recebeu de braços abertos na UFRJ e me passou um pouco de sua experiência na elaboração de teoremas e proposições. Agradeço também por sua receptividade, por estar sempre disposto a me ajudar e pelo ótimo trabalho em equipe que realizamos.

Ao professor Dr. Ibrahim Assem, por sua cordialidade e receptividade na Universidade de Sherbrooke. Nunca me esquecerei que o vinho que levou ao restaurante foi o melhor que já tomei.

Aos professores Dr. Hugo Alexandre Dantas Nascimento e Dr. Leslie Richard Foulds, por me ajudarem a estruturar os artigos escritos da maneira correta, pelas correções ortográficas e semânticas e pela disponibilidade em ajudar com toda a paciência.

Ao meu colega de doutorado Walid Abdala Rfaei Jradi, por compartilhar muitos momentos difíceis, pelas noites mal dormidas implementando os algoritmos e pelas centenas de horas compartilhadas na escrita dos artigos. Também pela amizade e companheirismo. Nós trabalhamos muito, mas valeu a pena.

Aos técnico-administrativos e professores do Instituto de Informática, que sempre me trataram muito bem, foram amigos e sempre torceram pelo meu sucesso. Especialmente à Berenice, Patrícia, Danielle, Enio, Bosco, Mirian, Marlliny e Carlene, que me deram todo o suporte na secretaria e sempre estiveram dispostos a ajudar.

De um modo geral, agradeço ao Estado Brasileiro pelo financiamento da minha educação. Desde a alfabetização até o doutorado, foram 26 anos de escola pública. Em particular, agradeço à FAPEG - Fundação de Amparo à Pesquisa do Estado de Goiás, pelo suporte financeiro durante o doutorado.

“Nós só podemos ver um pouco do futuro, porém o suficiente para perceber que há muito a fazer”

Alan Turing.

Resumo

Dias, Elisângela Silva. **Reconhecimento polinomial de álgebras cluster de tipo finito**. Goiânia, 2015. 124p. Tese de Doutorado . Instituto de Informática, Universidade Federal de Goiás.

As álgebras *cluster* formam uma classe de álgebras comutativas introduzida no início do milênio por Fomin e Zelevinsky. Elas são definidas de forma construtiva a partir de um conjunto de variáveis geradoras (variáveis *cluster*) agrupadas em subconjuntos sobrepostos (*clusters*) de cardinalidade fixa. Desde a sua criação, a teoria das álgebras *cluster* encontrou aplicações em diversas áreas da matemática e afins. Nesta tese, estudamos, com foco computacional, o reconhecimento das álgebras *cluster* de tipo finito. Em 2006, Barot, Geiss e Zelevinsky mostraram que uma álgebra *cluster* é de tipo finito se o grafo associado é ciclicamente orientado, isto é, todos os ciclos sem corda do grafo são ciclicamente orientados, e se a matriz antissimetrizável associada possui uma companheira quase-Cartan positiva. Em um primeiro momento, estudamos os dois tópicos de forma independente. Em relação à primeira parte do critério, elaboramos um algoritmo que lista todos os ciclos sem corda (polinomial no tamanho destes ciclos) e outro que verifica se um grafo é ciclicamente orientado e, em caso positivo, lista todos os seus ciclos sem corda (polinomial na quantidade de vértices). Relacionado à segunda parte do critério, desenvolvemos alguns resultados teóricos e elaboramos um algoritmo polinomial que verifica se uma matriz companheira quase-Cartan é positiva. Este último algoritmo é utilizado para provar que o problema de decidir se uma matriz antissimetrizável tem uma companheira quase-Cartan positiva para grafos gerais está na classe NP. Conjecturamos que este problema pertence à classe NP-completa. Mostramos que o mesmo pertence à classe de problemas polinomiais para grafos ciclicamente orientados e, por fim, mostramos que decidir se uma álgebra *cluster* é de tipo finito também pertence a esta classe.

Palavras–chave

Álgebras *cluster*, ciclos sem corda, grafos ciclicamente orientáveis, matriz positiva.

Abstract

Dias, Elisângela Silva. **Polynomial recognition of cluster algebras of finite type**. Goiânia, 2015. 124p. PhD. Thesis . Instituto de Informática, Universidade Federal de Goiás.

Cluster algebras form a class of commutative algebra, introduced at the beginning of the millennium by Fomin and Zelevinsky. They are defined constructively from a set of generating variables (cluster variables) grouped into overlapping subsets (clusters) of fixed cardinality. Since its inception, the theory of cluster algebras found applications in many areas of science, specially in mathematics. In this thesis, we study, with computational focus, the recognition of cluster algebras of finite type. In 2006, Barot, Geiss and Zelevinsky showed that a cluster algebra is of finite type whether the associated graph is cyclically oriented, i.e., all chordless cycles of the graph are cyclically oriented, and whether the skew-symmetrizable matrix associated has a positive quasi-Cartan companion. At first, we studied the two topics independently. Related to the first part of the criteria, we developed an algorithm that lists all chordless cycles (polynomial on the length of those cycles) and another that checks whether a graph is cyclically oriented and, if so, list all their chordless cycles (polynomial on the number of vertices). Related to the second part of the criteria, we developed some theoretical results and we also developed a polynomial algorithm that checks whether a quasi-Cartan companion matrix is positive. The latter algorithm is used to prove that the problem of deciding whether a skew-symmetrizable matrix has a positive quasi-Cartan companion for general graphs is in NP class. We conjecture that this problem is in NP-complete class. We show that the same problem belongs to the class of polynomial problems for cyclically oriented graphs and, finally, we show that deciding whether a cluster algebra is of finite type also belongs to this class.

Keywords

Cluster algebras, chordless cycles, cyclically orientable graphs, positive matrix.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Símbolos

Símbolo	**Significado**
$\langle S \rangle$	Subgrafo de G induzido pelo conjunto S
$grau(v)$	Grau do vértice v
P_k	Grafo caminho com k vértices
C_k	Grafo ciclo com k vértices
W_k	Grafo roda, formado por um ciclo C_k e um vértice central
T	Árvore
T_i	Subárvore de índice i
$\pi(v)$	Predecessor do vértice v na árvore de busca
$K_{r,s}$	Grafo bipartido
K_n	Grafo completo com n vértices
G_1 x G_2	Produto Cartesiano dos grafos G_1 e G_2
$G_{m,n}$	Grafo grade, resultante do produto Cartesiano de dois caminhos P_m e P_n
$\Delta(G)$	Grau máximo entre os vértices do grafo G
$\delta(G)$	Grau mínimo entre os vértices do grafo G
$cintura(G)$	Menor ciclo sem cordas no grafo G
$lcc(G)$	Tamanho máximo de um ciclo sem cordas no grafo G
$\lambda(G)$	Conectividade de arestas de um grafo G
$\kappa(G)$	Conectividade de vértices do grafo G
$G - v$	$G \setminus \{v\}$, o grafo G obtido pela eliminação do vértice v
$V(G) - \{s\}$	$V(G) \setminus \{s\}$, o conjunto de vértices $V(G)$ obtido pela eliminação do vértice s
$T - \{i\}$	$T \setminus \{i\}$, o conjunto T obtido pela eliminação do elemento i
$T(G)$	Conjunto de triplas do grafo G
P	Pilha
$d(v)$	Tempo de descoberta do vértice v na árvore de busca
$f(v)$	Tempo de finalização do vértice v na árvore de busca

CAPÍTULO 1

Introdução

No início do milênio, Sergey Fomin e Andrei Zelevinsky [16] introduziram uma classe de álgebras comutativas, chamada álgebras *cluster*. Estas álgebras possuem uma forte estrutura combinatória e são usadas como ferramentas para estudar questões relacionadas a bases canônicas duais e positividade de grupos de Lie semissimples.

Segundo Zelevinsky [59], entre estas álgebras encontram-se aneis de coordenadas de muitas variedades algébricas que desempenham um papel de destaque na teoria da representação, na teoria de invariante, no estudo de positividade total e em outras teorias.

Uma álgebra *cluster* é definida de forma construtiva, usando uma matriz antissimetrizável ou o grafo associado a ela, com um conjunto de variáveis geradoras (variáveis *cluster*) agrupadas em subconjuntos sobrepostos (*clusters*) de cardinalidade fixa.

Uma característica básica desta classe de álgebras é que tanto os geradores quanto as relações entre eles não são dados desde o início, mas são produzidos por um processo iterativo elementar de mutação de uma semente. Este processo, aparentemente não intuitivo, parece codificar um fenômeno universal de alguma forma. Tal fato pode explicar o desenvolvimento acelerado do tema em áreas como combinatória, física e matemática (especialmente geometria), dentre outras áreas.

Desde a sua criação, a teoria das álgebras *cluster* encontrou aplicações em representações de *quivers*, álgebras pré-projetivas, álgebras e categorias Calabi-Yau, teoria Teichmüller, sistemas integráveis discretos, geometria de Poisson, entre outras aplicações. O estado atual desses desenvolvimentos, incluindo *links* para documentos, trabalhos, seminários, conferências e outros materiais, é apresentado no portal *online* das álgebras *cluster* criado e mantido por Fomin [18].

Nesta tese, estudamos as álgebras *cluster* com foco computacional e os aspectos associados ao reconhecimento das álgebras *cluster* de tipo finito, isto é, as que têm um número finito de variáveis *cluster*. Sendo assim, consideramos o seguinte problema de reconhecimento:

Problema de reconhecimento: determinar se a álgebra *cluster* associada à matrizes antissimetrizáveis é de tipo finito.
Entrada: uma matriz antissimetrizável B.
Saída: decisão se a álgebra *cluster* $\mathcal{A}(B)$ é de tipo finito ou não.

Segue de Barot, Geiss e Zelevinsky [2] que podemos decidir se uma álgebra *cluster* é de tipo finito decidindo se o grafo associado a ela é ciclicamente orientado e a matriz antissimetrizável que a define tem uma companheira quase-Cartan positiva. Abordamos estas duas condições de maneira independente.

Visto que um grafo é ciclicamente orientado se todos os seus ciclos sem corda têm orientação cíclica, surgiu o interesse em buscar um algoritmo eficiente para encontrar e listar todos os ciclos sem corda. Para alcançar este objetivo, propomos o algoritmo $CiclosSemCorda(G)$ e uma extensão deste que utiliza a estratégia de busca em largura (BFS). Estes dois algoritmos utilizam um esquema de rotulação de vértices que permite um ciclo arbitrário ser descrito de maneira única. A partir disto, geramos um conjunto inicial de triplas de vértices e usamos uma estratégia de busca em profundidade (DFS) para encontrar todos os ciclos sem corda, garantindo que cada ciclo sem corda é encontrado apenas uma vez.

Gurvich [26] e Speyer [48] apresentaram várias propriedades de grafos ciclicamente orientáveis que nos ajudaram a desenvolver o algoritmo polinomial $CiclosSemCordaGrafosCO(G)$, que verifica se um grafo G é ciclicamente orientável, ou seja, possui uma atribuição cíclica às suas arestas que o torna ciclicamente orientado. Mostramos que o algoritmo de fato encontra todos os ciclos sem corda de tais grafos. Também apresentamos uma maneira fácil de alterar este algoritmo para que ele verifique se um grafo é ciclicamente orientado.

Em relação à companheira quase-Cartan, que é uma matriz simetrizável associada a outra antissimetrizável, apresentamos algumas propriedades das matrizes simetrizáveis e antissimetrizáveis, o algoritmo $MatrizSimetrizavel(A)$ para verificar se uma matriz A é simetrizável ou não e o algoritmo $MatrizSimetrizante(A)$ para apresentar uma matriz A simetrizante, caso exista.

Mostramos algumas características das matrizes companheiras quase-Cartan positivas, estreitamos alguns limites propostos por Barot, Geiss e Zelevinsky [2] e propomos maneiras mais fáceis de encontrá-las. Provamos que o problema de decidir se existe uma companheira quase-Cartan positiva para grafos gerais pertence à classe de problemas NP e conjecturamos que ele pertence à classe NP-completa. Apresentamos o algoritmo $CompanheiraPositiva(G,B,Csc)$ o qual verifica, com complexidade de tempo polinomial na dimensão da matriz B, se B possui uma companheira quase-Cartan positiva, caso $G = G(B)$ seja ciclicamente orientado. Este algoritmo junto com o algoritmo $CiclosSemCordaGrafosCO(G)$ atende aos critérios propostos por Barot, Geiss e Zele-

vinsky [2] para decidir se uma álgebra *cluster* é de tipo finito, o que mostra que este problema pertence à classe P.

O restante da tese está organizado da seguinte maneira: o Capítulo 2 apresenta algumas definições preliminares que serão necessárias ao entendimento dos resultados; o Capítulo 3 faz uma introdução à álgebra *cluster* e apresenta o problema de decidir se uma álgebra *cluster* é de tipo finito; o Capítulo 4 apresenta algoritmos polinomiais para encontrar todos os ciclos sem corda em grafos; o Capítulo 5 apresenta dois algoritmos, um para decidir se um grafo é ciclicamente orientável e, em caso positivo, listar todos os ciclos sem corda; e outro para atribuir uma orientação cíclica para grafos ciclicamente orientáveis; o Capítulo 6 apresenta algumas propriedades das matrizes companheiras quase-Cartan positivas, mostra que o problema da existência de uma companheira quase-Cartan positiva para grafos gerais pertence à classe NP e para grafos ciclicamente orientáveis pertence à classe P. Finalmente, ele apresenta um algoritmo polinomial para decidir se uma álgebra *cluster* é de tipo finito; o Capítulo 7 faz o fechamento da tese, mostrando as principais contribuições e alguns problemas em aberto.

CAPÍTULO 2

Definições Preliminares

Para entender os conceitos apresentados nos próximos capítulos, são necessárias algumas definições preliminares. Apresentamos noções sobre grafos na Seção 2.1; um algoritmo e as principais características da busca em profundidade estão na Seção 2.2; um algoritmo e a descrição da busca em largura estão na Seção 2.3; algumas propriedades, um algoritmo para determinar os componentes biconexos de um grafo e sua corretude são apresentados na Seção 2.4; as definições e notações principais sobre matrizes, especialmente sobre matrizes simetrizáveis e antissimetrizáveis, são apresentadas na Seção 2.5; apresentamos algumas propriedades e exemplos de matrizes quase-Cartan na Seção 2.6; a Seção 2.7 introduz dois algoritmos referentes às matrizes simetrizáveis; e, por fim, a Seção 2.8 fornece algumas propriedades das matrizes positivas.

2.1 Noções sobre grafos

Ao longo do texto, a palavra grafo se refere a grafos finitos não orientados, sem laços nem arestas múltiplas. Seja G um grafo com conjunto de vértices $V(G)$ e conjunto de arestas $E(G)$. Sejam $n = |V(G)|$ e $m = |E(G)|$.

Um *caminho simples* ou *elementar*, denotado por P_k, é uma sequência finita de vértices $\langle v_1, v_2, \ldots, v_k \rangle$ tal que $(v_i, v_{i+1}) \in E(G)$ para todo $i \in \{1, \ldots, k-1\}$ e sem vértice repetido na sequência, isto é, $v_i \neq v_j$ para todo $j \in \{1, \ldots, k\}$ com $j \neq i$. A Figura 2.1, exemplifica o conceito.

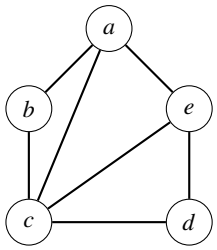
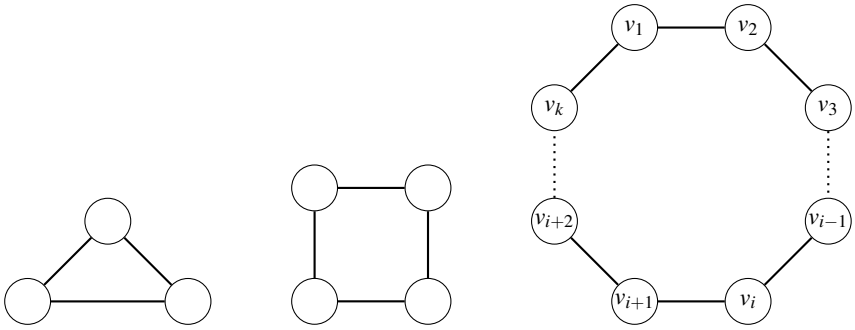


Figura 2.1: $\langle a, e, d, c \rangle$ *é um caminho simples, mas* $\langle a, e, c, a, b \rangle$ *não.*

Um *ciclo* é um caminho simples $\langle v_1, v_2, \ldots, v_k \rangle$ tal que $(v_k, v_1) \in E(G)$ e $k \geq 3$. Denotamos um grafo ciclo com k vértices por C_k. Nesse tipo de grafo, o número de arestas é igual a número de vértices e cada vértice possui grau dois (Figura 2.2).



(a) C_3. (b) C_4. (c) C_k.

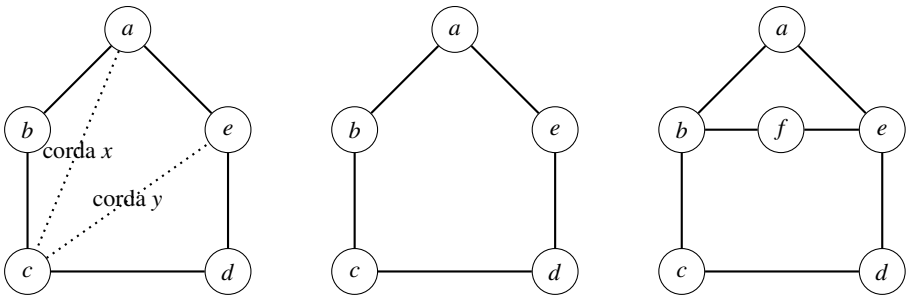
Figura 2.2: *Exemplos de grafos ciclo.*

Observe que nossa definição de ciclo não repete o primeiro vértice no final da sequência, como usualmente é feito. Nós decidimos usar esta definição (com o primeiro vértice implicitamente incluso no final), porque ela simplifica a representação de uma versão rotacionada dos ciclos. Note que se $\langle v_1, v_2, \ldots, v_k \rangle$ é um ciclo, então ele também pode ser representado por $\langle v_i, v_{i+1} \ldots, v_k, v_1, v_2, \ldots, v_{i-1} \rangle$ e $\langle v_i, v_{i-1}, \ldots, v_2, v_1, v_k, \ldots, v_{i+1} \rangle$, para todo $i = 1, \ldots, k$.

Por motivos de simplicidade, quando não há ambiguidade, os termos *caminho* e *ciclo* são utilizados para representar, respectivamente, *caminho simples* e *ciclo simples*.

Definição 2.1 *Uma **corda** de um caminho (resp. ciclo) é uma aresta entre dois vértices do caminho (ciclo) que não é parte dele.*

Podemos ver duas possíveis cordas no ciclo $\langle a, b, c, d, e \rangle$ da Figura 2.3(a).



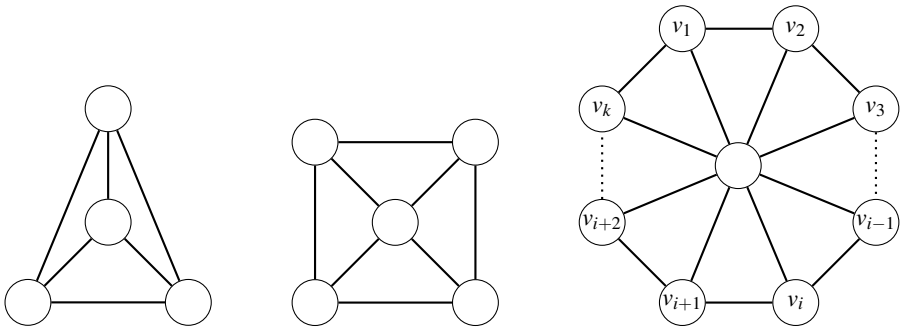
(a) (b) (c)

Figura 2.3: *Vértices $\langle a, b, c, d, e \rangle$ formando em (a) um ciclo com corda e em (b) e (c) ciclos sem corda.*

Definição 2.2 *Um caminho (ciclo) que não possui corda é chamado* **caminho sem corda (ciclo sem corda)**.

Observe que um caminho (ciclo) sem corda é um subgrafo induzido isomorfo a um P_k (resp. C_k). Para maiores informações sobre subgrafo induzido, veja o Apêndice A.

Um grafo *roda* W_k, com $k \geq 3$, é aquele obtido do grafo ciclo C_k ao se adicionar um vértice adjacente a todos os demais.



(a) W_3. (b) W_4. (c) W_k.

Figura 2.4: *Exemplos de grafos roda.*

O *grafo grade* $G_{m,n}$ é o grafo resultante do produto Cartesiano de dois caminhos P_m e P_n, como pode ser visto na Figura 2.5. Informações sobre produto Cartesiano podem ser encontradas no Apêndice A.

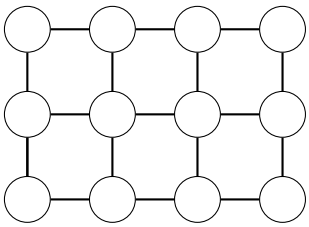
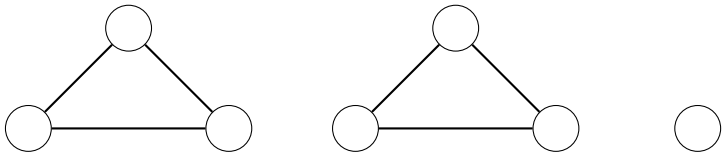


Figura 2.5: *Exemplo de uma grade, resultante de* $P_3 \times P_4$.

Um grafo G é denominado *conexo* quando existe caminho entre cada par de vértices de G, caso contrário G é *desconexo* (Figura 2.6).



(a) (b)

Figura 2.6: *(a) Grafo conexo; (b) Grafo desconexo.*

Um *componente conexo* de um grafo G é um subgrafo conexo maximal de G (Figura 2.7).

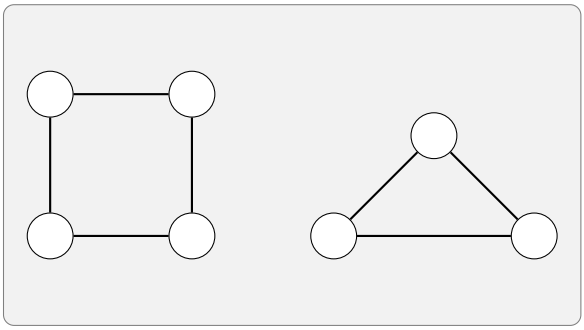


Figura 2.7: *Grafo G com dois componentes conexos.*

Uma *árvore* pode ser definida como sendo um grafo, em que qualquer par de vértices são conectados por, exatamente, um único caminho simples. Em outras palavras, qualquer grafo conexo que não contenha ciclos recebe a denominação de árvore. Uma *floresta* é definida como uma união disjunta de árvores.

Quando os ecologistas estudam relações entre animais e plantas e seu ambiente, algumas vezes eles usam um grafo orientado conhecido como *grafo food web* ou *grafo de cadeia alimentar*. Em tal grafo, os vértices correspondem às espécies investigadas e existe uma aresta da espécie A para a espécie B sempre que B é presa de A.

Como um exemplo de cadeia alimentar, considere o grafo orientado da Figura 2.8, que representa o hábito predatório de organismos em uma floresta.

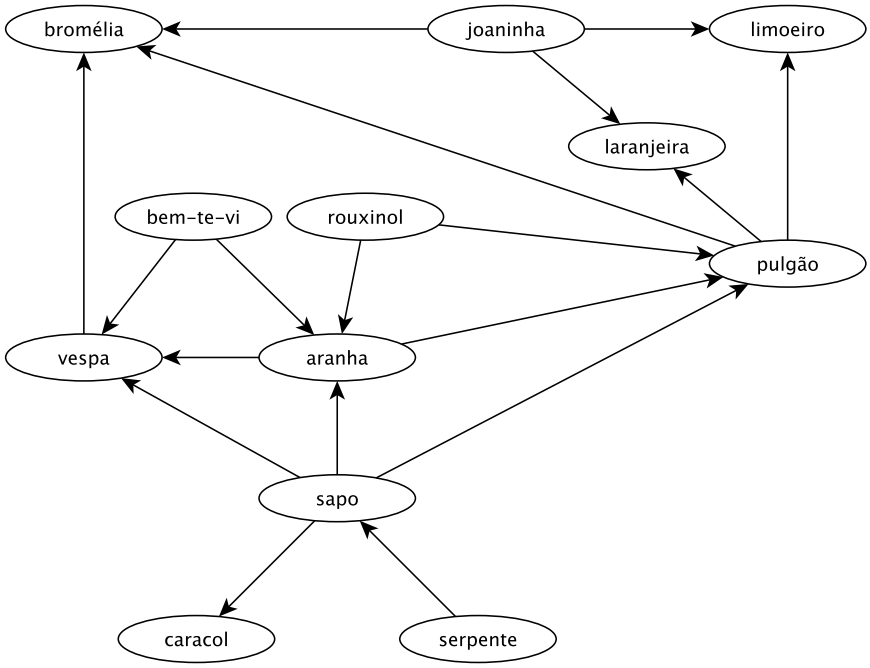


Figura 2.8: *Hábito predatório de organismos em uma floresta.*

Para entenderem melhor tais cadeias alimentares, ecologistas introduzem um grafo que mostra quais espécies competem por comida. Este grafo é conhecido como

grafo niche overlap ou *grafo de competição* e suas arestas unem pares de vértices representando espécies que dividem uma presa em comum. Por exemplo, na cadeia alimentar apresentada na Figura 2.8, o bem-te-vi e o rouxinol comem aranhas, então os vértices correspondentes são adjacentes no grafo de competição.

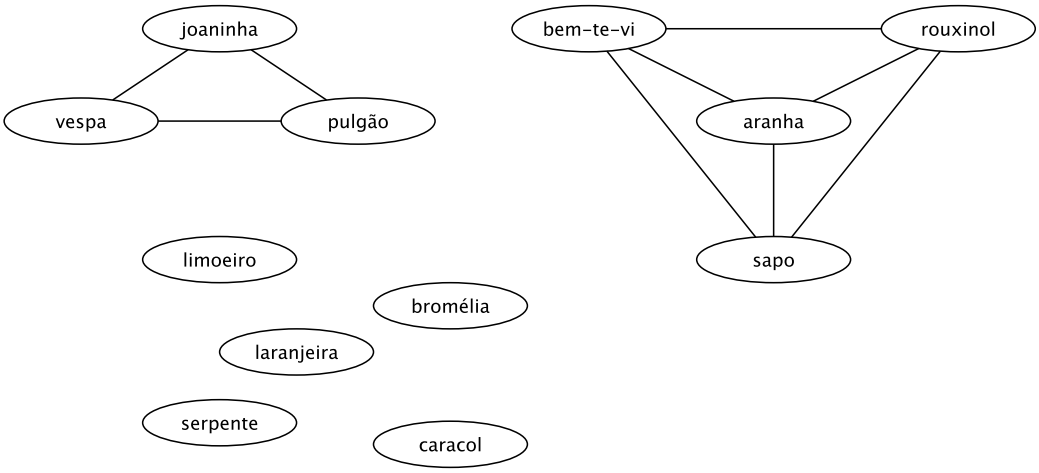


Figura 2.9: *Grafo de competição derivado do grafo da Figura 2.8.*

Tal representação tem significância ecológica em que vértices adjacentes tendem a corresponder a espécies que reagem da mesma maneira a fatores ambientais particulares, tais como temperatura, umidade e altitude. No exemplo acima, o besouros (joaninha e pulgão) e a vespa têm comportamento predatório similar, como o dos pássaros (bem-te-vi e rouxinol), a aranha e o sapo.

A maioria dos grafos de competição utilizados na prática são grafos de intervalos. Por exemplo, o grafo de competição da Figura 2.9 pode ser representado pelo seguinte conjunto de intervalos:

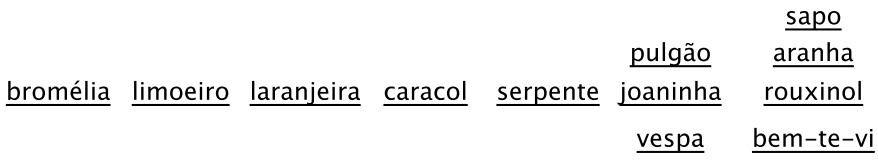


Figura 2.10: *Conjunto de intervalos derivado do grafo da Figura 2.8.*

Nas próximas seções, apresentamos alguns algoritmos básicos que são utilizados nos algoritmos elaborados no Capítulo 4. As notações e algoritmos utilizados estão de acordo com Cormen et al. [10].

2.2 Busca em profundidade (DFS)

A busca em profundidade (*Depth-First Search* – DFS) em um grafo tem a estratégia de procurar "mais fundo" no grafo sempre que possível. As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas inexploradas saindo dele. Quando todas as demais arestas de v são exploradas, a busca "regressa" para explorar as arestas que deixam o vértice a partir do qual v foi descoberto.

Esse processo continua até que todos os vértices acessíveis a partir do vértice origem inicial sejam descobertos. Se restarem quaisquer vértices não descobertos, então um deles será selecionado como uma nova origem e a pesquisa se repetirá a partir daquela origem. Esse processo inteiro será repetido até que todos os vértices sejam descobertos.

Sempre que um vértice v é descoberto durante uma varredura na lista de adjacências de um vértice já descoberto u, a busca em profundidade registra isso definindo o campo *predecessor* de v como $\pi(v) = u$.

O subgrafo predecessor produzido pela pesquisa pode ser composto por várias árvores, porque a pesquisa pode ser repetida a partir de várias origens, formando uma floresta. Arestas de árvore são aquelas descobertas quando os vértices ainda não foram visitados (coloridos). Formalmente, temos que $E_\pi = \{(\pi(v), v) : v \in V(G) \text{ e } \pi(v) \neq NIL\}$.

Os vértices são identificados por marcação de tempo, sendo que cada vértice v possui dois marcadores de tempo: o primeiro, denominado $d(v)$, registra quando v é descoberto pela primeira vez (e acinzentado) e o segundo marcador, denominado $f(v)$, registra quando a busca termina de examinar a lista de adjacência de v, colorindo-o de PRETO. Para todo vértice v, $d(v)$ é menor que $f(v)$. O vértice v é BRANCO antes do tempo $d(v)$, CINZA entre os tempos $d(v)$ e $f(v)$ e depois é marcado com PRETO.

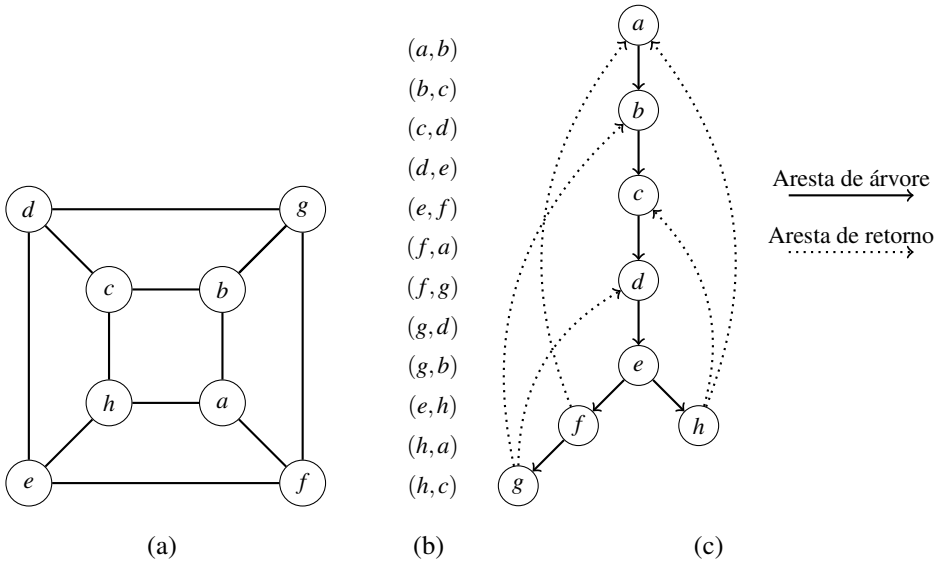


Figura 2.11: *A árvore (c) é um exemplo de DFS para o grafo (a), seguindo a ordem de exploração das arestas (b).*

O algoritmo permite que o conjunto de arestas seja classificado em:

1. uma aresta (u,v) é *aresta de árvore* (A) se v foi descoberto durante a sua exploração;
2. as *arestas de retorno* (R) conectam um vértice u com um antecessor v na árvore de busca;
3. *arestas diretas* (D) são aquelas que não são de árvore e que conectam um vértice v a um descendente u na árvore de busca; e
4. *arestas cruzadas* (C) são todas as outras.

Observe que em um grafo não orientado os conceitos de arestas de retorno e de arestas diretas são os mesmos. Em um grafo orientado, as arestas de retorno formam um ciclo orientado, enquanto que as diretas formam um ciclo não orientado.

Algoritmo 2.1: $DFS(G)$

Entrada: Grafo G.

Saída: Uma árvore de busca primeiro em profundidade.

```
1  para cada u ∈ V(G) faça
2      cor(u) ← BRANCA
3      π(u) ← NIL
4  tempo ← 0
5  para cada u ∈ V(G) faça
6      se (cor(u) = BRANCA) então
7          DFS-VISIT (u)
```

Algoritmo 2.2: $DFS-VISIT(u)$

Entrada: Um vértice $u \in V(G)$.

```
1  cor(u) ← CINZA /* Branco, o vértice u acabou de ser descoberto */
2  tempo ← tempo + 1
3  d(u) ← tempo
4  para cada v ∈ Adj(u) faça                  /* Explora a aresta (u,v) */
5      se (cor(v) = BRANCA) então
6          π(v) ← u
7          DFS-VISIT (v)
8  cor(u) ← PRETA        /* Colore u de preto; ele é finalizado */
9  f(u) ← tempo
10 tempo ← tempo + 1
```

2.3 Busca em largura (BFS)

O algoritmo de busca em largura (*Breadth-First Search* – BFS) é um algoritmo que encontra os caminhos mais curtos a partir de um vértice inicial. Essa técnica é utilizada em vários algoritmos mais complexos, como por exemplo o algoritmo de Dijkstra (para determinar o menor caminho entre vértices para grafos ponderados) e o de árvore geradora mínima.

Dado um grafo G e um vértice de origem s, a busca em largura explora sistematicamente as arestas de G até "descobrir" cada vértice acessível a partir de s. O algoritmo calcula a distância (menor número de arestas) desde s até todos os vértices acessíveis. Ele encontra o caminho mais curto de s a qualquer outro vértice.

Para controlar o andamento durante a busca em largura, cada vértice é colorido de BRANCO, CINZA ou PRETO. No início, todos os vértices são brancos e mais tarde eles podem se tornar acinzentados e depois pretos.

Algoritmo 2.3: $BFS(G,s)$

Entrada: Um grafo G e um vértice $s \in V(G)$.

Saída: Uma árvore de busca primeiro na largura.

```
1  para cada v ∈ V(G) − {s} faça
2      cor(u) ← BRANCA
3      d(u) ← ∞
4      π(u) ← NIL
5  cor(s) ← CINZA
6  d(s) ← 0
7  π(s) ← NIL
8  Q ← 0
9  ENQUEUE(Q,s)
10 enquanto (Q ≠ ∅) faça
11     u ← DEQUEUE(Q)
12     para cada v ∈ Adj(u) faça
13         se (cor(v) = BRANCa) então
14             cor(v) ← CINZA
15             d(v) ← d(v) + 1
16             π(u) ← u
17             ENQUEUE(Q,v)
18     cor(u) ← PRETA
```

A busca em largura constroi uma árvore primeiro na extensão, contendo inicialmente apenas sua raiz s. Sempre que um vértice branco v é descoberto na verificação da

lista de adjacências de um vértice u já descoberto, o vértice v e a aresta (u,v) são adicionados à árvore. Dizemos que u é predecessor ou pai de v na árvore. Se u está em um caminho na árvore a partir da raiz s até o vértice v, então u é um ancestral de v e v é um descendente de u.

As funções $cor(v)$, $\pi(v)$ e $d(u)$ têm a mesma conotação do algoritmo DFS explicado anteriormente.

2.4 Componentes biconexos

Componentes biconexos são importantes para determinar se um grafo é ciclicamente orientável ou para enumerar ciclos sem corda, pois cada ciclo pertence a um único componente. Utilizamos estes componentes para verificar se cada um deles é ciclicamente orientável e, assim, decidir se o grafo é ciclicamente orientável, veja o Capítulo 5. A enumeração dos ciclos sem corda também pode ser feita para cada componente biconexo.

As definições utilizadas nesta seção estão de acordo com Szwarcfiter [50]. Seja G um grafo. Um vértice v é denominado *articulação* ou *vértice de corte* (Figura 2.12(a)) quando a sua remoção aumenta a quantidade de componentes conexos de G. Uma aresta é denominada de *corte* ou *ponte* (Figura 2.12(b)) quando a sua remoção aumenta a quantidade de componentes conexos de G.

Lema 2.3 (Szwarcfiter [50]) *Seja G um grafo conexo com $n > 2$. Temos que:*

(a) um vértice $v \in V(G)$ é articulação de G se e somente se existirem vértices $w \neq v, u \neq v$ tais que v está contido em todo caminho entre w e u em G; e

(b) uma aresta $(p,q) \in E(G)$ é ponte se e somente se (p,q) for o único caminho simples entre p e q em G.

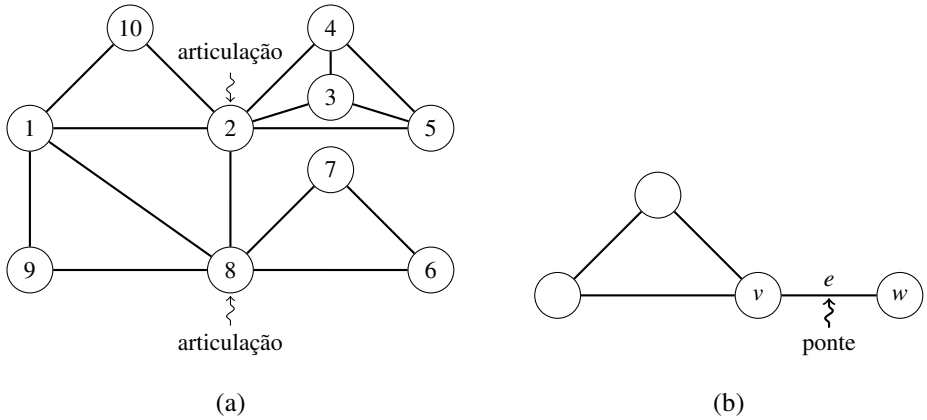


Figura 2.12: *(a) Grafo com articulações em 2 e 8; (b) Grafo com ponte entre os vértices v e w.*

Um *corte de vértices* de G é um subconjunto minimal de vértices $V'(G) \subseteq V(G)$, cuja remoção aumenta a quantidade de componentes conexos de G. Analogamente, um *corte de arestas* de G é um subconjunto minimal de arestas $E'(G) \subseteq E(G)$, cuja remoção aumenta a quantidade de componentes conexos de G.

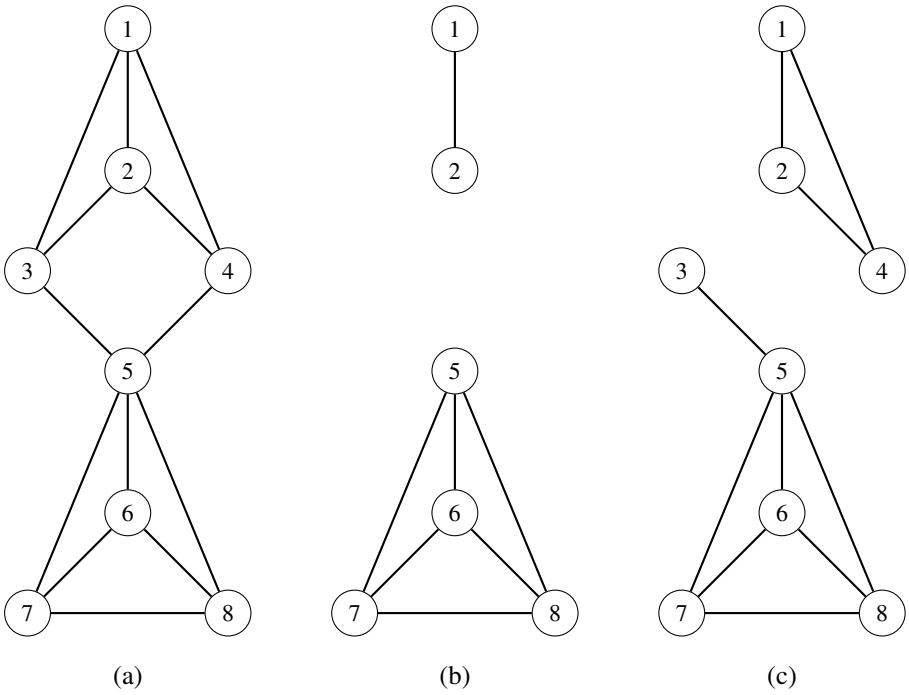


Figura 2.13: *Os grafos (b) e (c) são exemplos de cortes do grafo (a).*

Denomina-se *conectividade de vértices* $\kappa(G)$ de G como sendo a cardinalidade do menor corte de vértices de G. De maneira semelhante, a *conectividade de arestas* $\lambda(G)$ de G é igual à cardinalidade do menor corte de arestas de G. Se G é um grafo desconexo, então $\kappa(G) = \lambda(G) = 0$.

Ao analisarmos o grafo da Figura 2.13(a), como exemplo, observamos que o subconjunto $\{3,4\}$ forma um corte de vértices, pois sua remoção desconecta o grafo nos componentes ilustrados na Figura 2.13(b). Por outro lado, tomando como base o subconjunto $\{(1,3),(2,3),(4,5)\}$, temos um corte de arestas que ao ser removido de G gera um grafo desconexo (Figura 2.13(c)).

Ainda com base no grafo da Figura 2.13(a), temos, respectivamente, os subconjuntos $\{5\}$ e $\{(3,5),(4,5)\}$ como cortes de vértices e arestas de cardinalidade mínima, respectivamente. Logo, $\kappa(G) = 1$ e $\lambda(G) = 2$.

Um grafo G é *k-conexo* em vértices quando $\kappa(G) \geq k$. De forma análoga, um grafo é *k-conexo* em arestas quando $\lambda(G) \geq k$. Um grafo é *biconexo* se $\kappa(G) = 2$. Podemos ver um grafo e seus respectivos componentes biconexos na Figura 2.14.

Conforme a definição de Tamassia e Goodrich [51], um grafo conexo G é biconexo se, para quaisquer dois vértices u e v, existirem dois caminhos que não compartilham arestas ou vértices comuns, exceto u e v.

Ainda segundo [51], considerando-se G um grafo conexo com $n \geq 2$, os *componentes biconexos* são definidos como sendo subgrafos de G que satisfazem uma das seguintes condições:

- uma única aresta de G, consistindo em uma ponte e seus nós extremos (componente biconexo simples); ou
- um subgrafo de G que é biconexo maximal, ou seja, para o qual adicionar vértices ou arestas a G faria com que ele deixasse de ser biconexo.

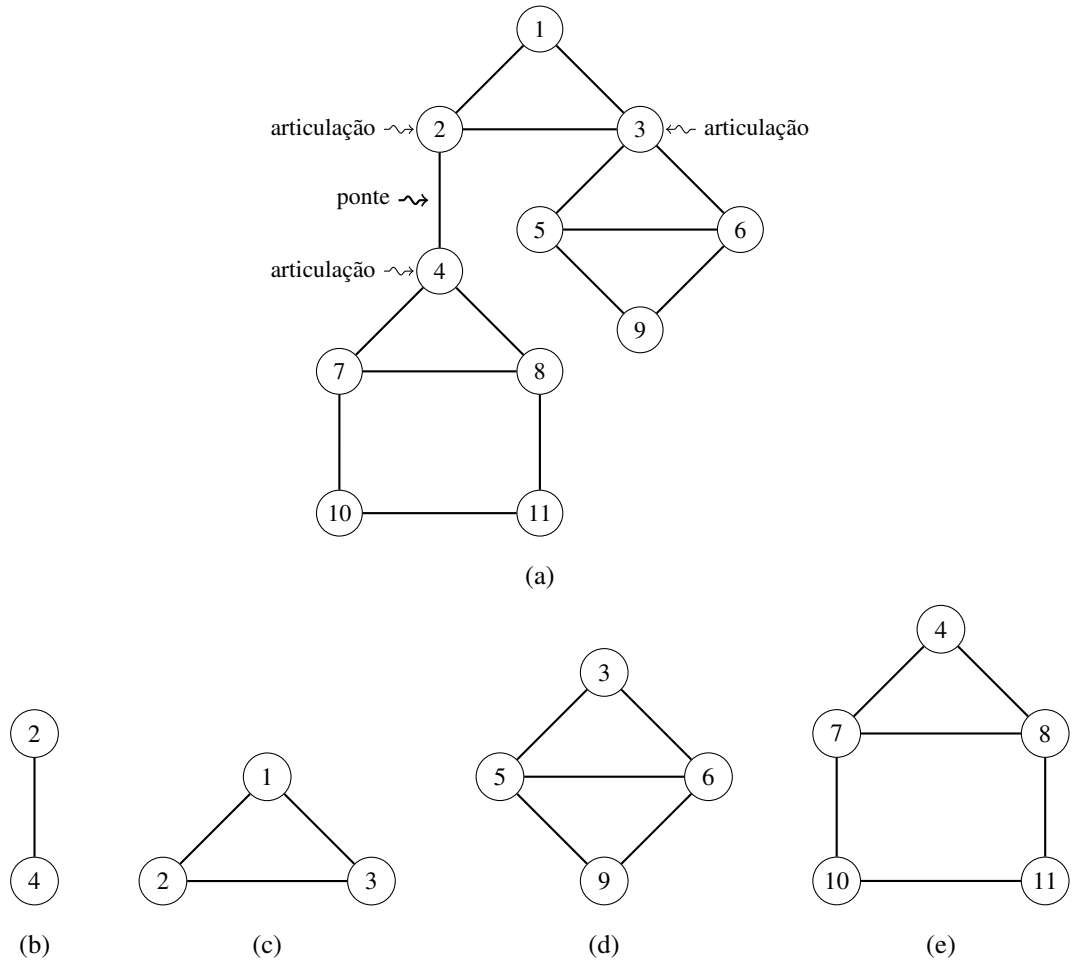


Figura 2.14: *Um grafo (a) e seus componentes biconexos (b), (c), (d) e (e).*

Para calcular os componentes biconexos de um grafo, podemos utilizar o Algoritmo 2.4, que é baseado nas ideias de Tarjan [52] e de Szwarcfiter [50]. Este algoritmo é uma aplicação da busca em profundidade (DFS) descrita anteriormente e tem complexidade de tempo $O(n^2)$. O algoritmo também determina o conjunto dos vértices de

articulação do grafo como passo intermediário. A corretude do algoritmo será mostrada posteriormente.

As arestas em E_π são chamadas arestas de árvore (A) e são as que nos interessam para calcular os componentes biconexos. Para maiores informações sobre estas arestas, veja a Seção 2.2.

Szwarcfiter [50] define a função *proximo_raiz* do modo seguinte. Seja G um grafo e T uma árvore de profundidade de G. Para cada vértice $u \in V(G)$, $proximo\_raiz(u)$ é igual ao vértice mais próximo da raiz de T que pode ser alcançado a partir de u, caminhando-se em T para baixo através de zero ou mais arestas de árvore (A) e, em seguida, para cima utilizando no máximo uma aresta de retorno (R). Associamos um valor a cada vértice, assim sendo $proximo\_raiz(u)$ guarda este valor ao invés do vértice correspondente.

A aplicação da função *proximo_raiz* para a determinação das articulações de um grafo é dada pelo lema a seguir.

Lema 2.4 (Szwarcfiter [50]) *Seja G um grafo conexo e T uma árvore de profundidade de G. Um vértice $u \in V(G)$ é uma* **articulação** *de G se e somente se*

(a) u é a raiz de T e u possui mais de um filho; ou
(b) u não é a raiz de T e u possui um filho v tal que $proximo\_raiz(v)$ é igual ao tempo de descoberta de u ($d(u)$) ou dele mesmo ($d(v)$) na árvore T.

Sejam u, v vértices de um grafo G e T uma árvore de profundidade de G. Se u é pai do vértice v em T, então v é chamado de *demarcador* de u quando $proximo\_raiz(v) = d(u)$ ou $proximo\_raiz(v) = d(v)$ ou quando u é raiz de T.

Algoritmo 2.4: $ComponentesBiconexos(G)$

Entrada: Grafo simples não orientado G.
Saída: Componentes biconexos de G.

```
1  para cada u ∈ V(G) faça
2      cor(u) ← BRANCA
3      π(u) ← NIL
4  tempo ← 0
5  P ← ∅            /* P é uma pilha */
6  i ← 0
7  para cada u ∈ V(G) faça
8      se (cor(u) = BRANCA) então
9          Visita (u)
10 retorna Os componentes biconexos G_i de G.
```

No Algoritmo 2.5, cada vértice é visitado e recebe um número que identifica o momento em que foi alcançado durante a busca, tal como descrito no procedimento DFS (Seção 2.2). Se o vértice u for ancestral de v, então $d(u) < d(v)$.

Algoritmo 2.5: $Visita(u)$

Entrada: Um vértice u.

```
1  cor(u) ← CINZA
2  tempo ← tempo + 1
3  d(u) ← tempo
4  proximo_raiz(u) ← tempo
5  para cada v ∈ Adj(u) faça
6      se (cor(v) = BRANCA) então     /* Encontra arestas de árvore. */
7          π(v) ← u
8          insira (u,v) na pilha P
9          Visita(v)
10         se ((proximo_raiz(v) ≥ d(u)) e (P ≠ ∅)) então
11             u é uma articulação se não é raiz ou possui mais de um filho
12             v é um demarcador
13             i ← i + 1
14             T_i ← os elementos da pilha P até (u,v)
15             remova T_i de P
16             G_i ← subgrafo induzido por T_i
17         proximo_raiz(u) ← min(proximo_raiz(u), proximo_raiz(v))
18     senão                          /* Observe que π(v) ≠ u. */
19         se (π(u) ≠ v) então        /* (u,v) é uma aresta de retorno. */
20             proximo_raiz(u) ← min(proximo_raiz(u), d(v))
21     cor(u) ← PRETA
```

Seguindo as ideias apresentadas no Algoritmo 2.4, mostramos um exemplo de sua execução nas Figuras 2.15 e 2.16.

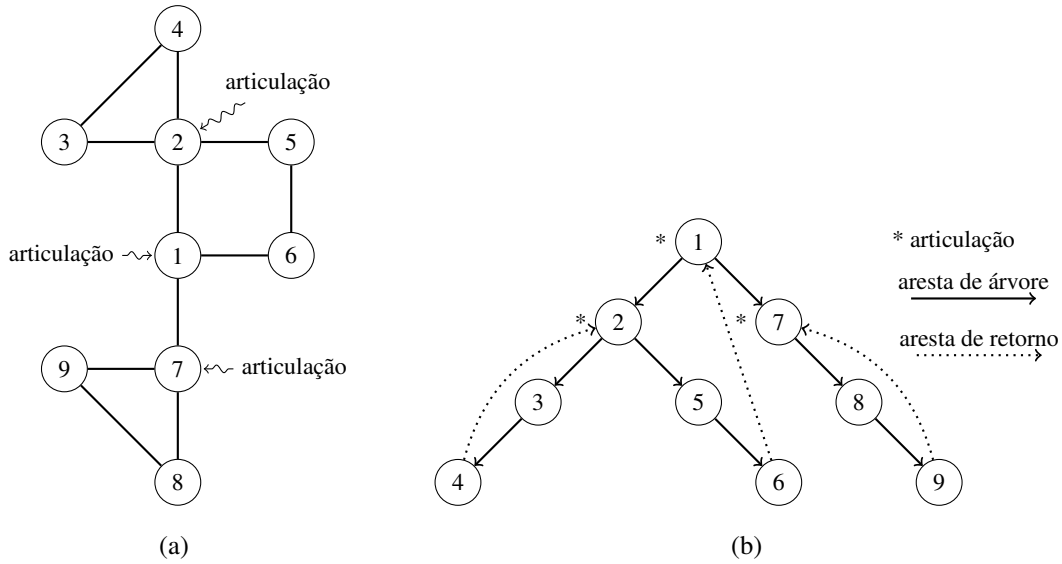


Figura 2.15: *Árvore de busca (b) gerada a partir do grafo (a).*

Tabela 2.1: *Função proximo_raiz(v) correspondente à arvore da Figura 2.15(b).*

u	1	2	3	4	5	6	7	8	9
$d(u)$	1	2	3	4	5	6	7	8	9
$proximo\_raiz(u)$	1	1	2	2	1	1	7	7	7

Na Figura 2.15, os vértices 1, 2 e 7 são articulações e os vértices 2, 3, 7 e 8 são demarcadores. Neste exemplo, a raiz da árvore de profundidade é uma articulação. No próximo exemplo, veremos uma árvore de profundidade cuja raiz não é articulação.

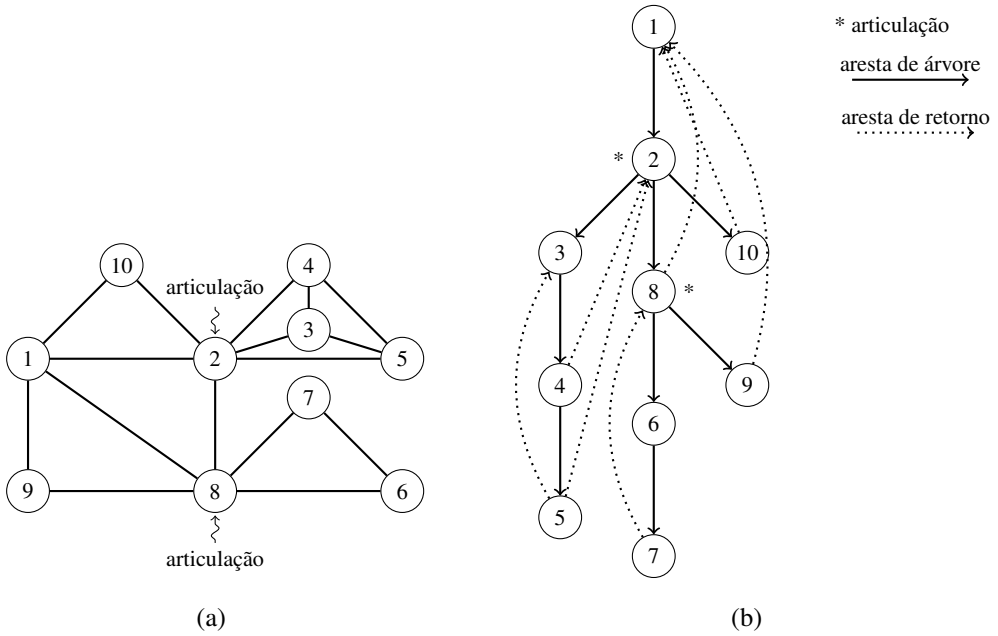


Figura 2.16: *Árvore de busca (b) gerada a partir do grafo (a).*

Tabela 2.2: *Função proximo_raiz correspondente ao grafo da Figura 2.16.*

u	1	2	3	4	5	6	7	8	9	10
$d(u)$	1	2	3	4	5	7	8	6	9	10
$proximo\_raiz(u)$	1	1	2	2	2	6	6	1	1	1

Como exemplo, considere o grafo G da Figura 2.16 (a), cujas articulações são os vértices 2 e 8. A Figura 2.16 (b) exemplifica uma árvore em profundidade T de G com as articulações destacadas. Observe que na Tabela 2.2, na linha $proximo\_raiz(u)$ o valor 6 corresponde ao vértice 8 já que $d(8) = 6$.

A subárvore $T_1 = \{(4,5),(3,4),(2,3)\}$, de raiz 2, não contém articulações em G_1 e o vértice 3 é demarcador. Logo, $\{2,3,4,5\}$ constituem os vértices de um componente biconexo. Retiramos T_1 de P. Visto que o vértice 6 é um demarcador, tal que $T_2 = \{(6,7),(8,6)\}$ não contém articulações em G_2, os vértices $\{1,2,8,9,10\}$ induzem um componente biconexo em G. Retiramos T_2 de P. Como o vértice 2 é um demarcador de $T_3 = \{(2,10),(8,9),(1,2)\}$, os vértices $\{6,7,8\}$ também induzem um componente biconexo em G. Por último, retiramos T_3 de P e o processo é finalizado, uma vez que todos os demarcadores de G foram considerados.

Para mostrar a corretude do algoritmo $ComponentesBiconexos(G)$, devemos mostrar que ele calcula o $proximo\_raiz(u)$ corretamente. Com o objetivo de simplificar a demonstração, definimos a função $pr(u)$ como sendo o valor calculado na função $proximo\_raiz(u)$ do Algoritmo 2.5 ($Visita(u)$).

Lema 2.5 *Com as notações anteriores, $proximo\_raiz(u) = pr(u)$.*

Prova. Sejam $min\_retorno(u) = \min\{d(v) \mid v \in Adj(u),\ \pi(v) \neq u \text{ e } \pi(u) \neq v\}$; $min\_arvore(u) = \min\{pr(v) \mid v \in Adj(u) \text{ e } \pi(v) = u\}$. Pelo Algoritmo 2.5, temos que $pr(u) = \min\{d(u), min\_retorno(u), min\_arvore(u)\}$.

Vamos mostrar por indução na profundidade do vértice u. Suponha que u é uma folha de T, então $pr(u) = \min\{d(u), min\_retorno(u)\}$. Observe que se $v \in Adj(u)$, temos que $\pi(v) \neq u$ e $\pi(u) \neq v$ se e somente se (u,v) é uma aresta de retorno. Logo, é fácil ver que $pr(u) = proximo\_raiz(u)$.

Por indução, temos que $min\_arvore(u) = \min\{proximo\_raiz(v) \mid v \in Adj(u) \text{ e } \pi(v) = u\}$. Em primeiro lugar, mostramos que $proximo\_raiz(u) \leq pr(u)$.

Sabemos que $proximo\_raiz(u) \leq d(u)$, por definição de $proximo\_raiz(u)$. Temos que $proximo\_raiz(u) \leq min\_retorno(u)$, pois suponha que $min\_retorno(u) = d(v)$, então temos que $u \overset{R}{\text{---}} v$.

Vamos mostrar que $proximo\_raiz(u) \leq min\_arvore(u)$. Suponha que $min\_arvore(u) = proximo\_raiz(v)$. Se $proximo\_raiz(v) = d(v)$, então temos que

$proximo\_raiz(u) \leq d(u) \leq d(v) = proximo\_raiz(v)$. Se $proximo\_raiz(v) = d(w)$, temos que $u \xrightarrow{A} v \overset{A}{\rightsquigarrow} \cdot \xrightarrow{R} w$. Assim, $proximo\_raiz(u) \leq d(w) = proximo\_raiz(v)$. Logo, $proximo\_raiz(u) \leq min\_arvore(u)$. Portanto, $proximo\_raiz(u) \leq pr(u)$.

Falta mostrar que $pr(u) \geq proximo\_raiz(u)$. Por definição, temos que $pr(u) \leq d(u)$. Suponha que existe um w tal que $proximo\_raiz(u) = d(w)$, ou seja $u \overset{A}{\rightsquigarrow} x \xrightarrow{R} w$.

Se $u \overset{A}{\rightsquigarrow} x$ é trivial, isto é, $u \overset{A}{\rightsquigarrow} x \xrightarrow{R} w = u \xrightarrow{R} w$, então $\pi(w) \neq u$ e $\pi(u) \neq w$. Logo, temos que $d(w) \geq min\_retorno(u)$. Neste caso, $pr(u) \leq d(w)$.

Se $u \overset{A}{\rightsquigarrow} x$ não é trivial, então existe $v \in Adj(u)$ tal que $u \overset{A}{\rightsquigarrow} x = u \xrightarrow{A} v \overset{A}{\rightsquigarrow} x$. Sendo assim, temos que $v \overset{A}{\rightsquigarrow} x \xrightarrow{R} w$. Logo, $proximo\_raiz(v) \leq d(w)$. Dessa forma, $min\_arvore(u) \leq d(w)$. Neste caso, $pr(u) \leq d(w)$.

Portanto, podemos concluir que $pr(u) = proximo\_raiz(u)$. □

Teorema 2.6 *O algoritmo $ComponentesBiconexos(G)$ é correto.*

Prova. Pelo Lema 2.5, mostramos que a função *proximo_raiz* é calculada corretamente e, assim, o algoritmo marca corretamente as articulações do grafo e, consequentemente, lista os componentes biconexos do grafo dado. □

2.5 Matrizes simetrizáveis e antissimetrizáveis

As matrizes podem ser utilizadas para representar diversas estruturas, entre elas os grafos e as álgebras, em particular as álgebras *cluster*. Uma álgebra *cluster* pode ser definida usando-se uma matriz antissimetrizável ou, equivalentemente, usando-se um grafo orientado $Q(B)$ sem laços ou 2-ciclos, chamado *quiver*. Para maiores informações sobre *quiver* e álgebras *cluster*, veja a Seção 3.2.

Para quaisquer matrizes arbitrárias A de dimensão $m \times n$ e B de dimensão $p \times q$, a *soma direta* de A e B, denotada por $A \oplus B$, é definida como:

$$A \oplus B = \begin{bmatrix} a_{11} & \cdots & a_{1n} & 0 & \cdots & 0 \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ a_{m1} & \cdots & a_{mn} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & b_{11} & \cdots & b_{1q} \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ 0 & \cdots & 0 & b_{p1} & \cdots & b_{pq} \end{bmatrix}, \text{ ou seja, } A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}.$$

Exemplo 2.7 *Soma direta de duas matrizes:* $\begin{bmatrix} 1 & 3 & 2 \\ 2 & 3 & 1 \end{bmatrix} \oplus \begin{bmatrix} 1 & 6 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 2 & 0 & 0 \\ 2 & 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$.

Esta operação generaliza naturalmente vetores dimensionados arbitrariamente (desde que A e B tenham a mesma quantidade de dimensões).

Uma *matriz de permutação* é a matriz binária quadrada que tem exatamente uma entrada com valor 1 em cada linha e em cada coluna e 0s nos outros casos. Cada matriz representa uma permutação específica de m elementos e, quando multiplicada por outra matriz, produz uma permutação de linhas ou colunas da outra matriz.

Dada uma permutação π de n elementos, $\pi : \{1,\ldots,n\} \to \{1,\ldots,n\}$, sua matriz de permutação é a matriz P_π de ordem n na qual as entradas são todas 0 exceto na posição $(i,\pi(i))$ que é igual a 1.

Seja P a matriz associada à permutação (i, j). A matriz PAP é obtida da matriz A trocando a linha i com a linha j e a coluna i com a coluna j.

Dizemos que A é *desconexa* se existe uma matriz de permutação P tal que PAP é a soma direta de pelo menos duas matrizes quadradas não-nulas. Se isto não ocorre, dizemos que A é *conexa*. Observe que PAP é obtida de A pela permutação de linhas e respectivas colunas. Além disso, A é conexa exatamente quando o grafo associado à matriz de adjacências de A também é.

Ao longo desta tese, consideramos somente matrizes quadradas com entradas inteiras, exceto a matriz diagonal D, que poderá ter valores racionais por motivo de simplicidade.

Considere n um inteiro positivo e $A,B,C \in \mathrm{M}_n(\mathbb{Z})$ e $D \in \mathrm{M}_n(\mathbb{Q})$. Uma matriz A é *simétrica* se $A = A^T$, onde A^T representa a transposta de A. Uma matriz C é *simetrizável* se $D \times C$ é simétrica para alguma matriz diagonal D com entradas positivas. Neste caso, a matriz $D \times C$ é chamada de *simetrização* ou *simetrizada* de C e a matriz D de *simetrizante* de C. Uma definição equivalente foi dada em [9]. Uma matriz C é *simétrica pelos sinais* se para todo $i, j \in \{1,\ldots,n\}$, com $i \neq j$, temos que $c_{ij} = c_{ji} = 0$ ou $c_{ij} \cdot c_{ji} > 0$.

Dadas as matrizes D e C de ordem 3, para que $D \times C$ seja simétrica, devemos ter

$$D = \begin{pmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{pmatrix} \text{ e } C = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}, D \times C = \begin{pmatrix} d_1 \cdot c_{11} & d_1 \cdot c_{12} & d_1 \cdot c_{13} \\ d_2 \cdot c_{21} & d_2 \cdot c_{22} & d_2 \cdot c_{23} \\ d_3 \cdot c_{31} & d_3 \cdot c_{32} & d_3 \cdot c_{33} \end{pmatrix}$$

e $\begin{cases} d_1 \cdot c_{12} = d_2 \cdot c_{21} \\ d_1 \cdot c_{13} = d_3 \cdot c_{31} \\ d_2 \cdot c_{23} = d_3 \cdot c_{32} \end{cases}$. Ou seja, se existem $d_i \in \mathbb{Z}_+$ tais que $d_i \cdot c_{ij} = d_j \cdot c_{ji}$, então C é dita simetrizável. Por questões de simplicidade, utilizamos $d_{ii} = d_i$ para todo $i \in \mathbb{Z}_+$.

Exemplo 2.8 Dada a matriz simetrizável $C = \begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix}$, a matriz $D = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$ é uma simetrizante de C. A matriz simetrizada é $D \times C = \begin{pmatrix} 6 & 2 \\ 2 & 2 \end{pmatrix}$.

Observamos que toda matriz simétrica é simetrizável e que toda matriz simetrizável é simétrica pelos sinais. Mas a recíproca não é verdadeira.

Exemplo 2.9 *Matriz simétrica pelos sinais que não é simetrizável:* $C = \begin{pmatrix} 2 & -3 & 2 \\ -1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}$.

Uma matriz A é *antissimétrica* se sua transposta coincide com sua oposta ($A^T = -A$), ou seja, $a_{ij} = -a_{ji}$, para todo i, j. Notamos que os valores da diagonal principal de uma matriz antissimétrica são nulos. Uma matriz B é *antissimetrizável* se existe uma matriz diagonal D com entradas positivas tal que $D \times B$ é uma matriz antissimétrica, ou seja, $D \times B = -(D \times B)^T$. Neste caso, a matriz $D \times B$ é chamada de *antissimetrização* ou *antissimetrizada* de B e a matriz D é chamada de *antissimetrizante* de B. Uma matriz B é *antissimétrica pelos sinais* se para todo $i, j \in \{1 \ldots n\}$ temos que $b_{ij} = b_{ji} = 0$ ou $b_{ij} \cdot b_{ji} < 0$. Observamos novamente que $b_{ii} = 0$. Toda matriz antissimétrica é antissimetrizável e toda matriz antissimetrizável é antissimétrica pelos sinais.

Para que uma matriz de ordem 3 seja antissimetrizada, devemos observar que $D = \begin{pmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{pmatrix}$ e $B = \begin{pmatrix} 0 & b_{12} & b_{13} \\ b_{21} & 0 & b_{23} \\ b_{31} & b_{32} & 0 \end{pmatrix}$, $D \times B = \begin{pmatrix} 0 & d_1 \cdot b_{12} & d_1 \cdot b_{13} \\ d_2 \cdot b_{21} & 0 & d_2 \cdot b_{23} \\ d_3 \cdot b_{31} & d_3 \cdot b_{32} & 0 \end{pmatrix}$.

e $\begin{cases} d_1 \cdot b_{12} = -d_2 \cdot b_{21} \\ d_1 \cdot b_{13} = -d_3 \cdot b_{31} \\ d_2 \cdot b_{23} = -d_3 \cdot b_{32} \end{cases}$. Ou seja, se existem $d_i \in \mathbb{Z}_+$ tais que $d_i \cdot b_{ij} = -d_j \cdot b_{ji}$, então B é antissimetrizável. Matrizes quadradas antissimetrizáveis de ordem 2 são da seguinte forma: $B = \begin{pmatrix} 0 & b \\ -c & 0 \end{pmatrix}$ ou $-B = \begin{pmatrix} 0 & -b \\ c & 0 \end{pmatrix}$ com $b, c \in \mathbb{N}$.

Assim, qualquer matriz antissimétrica é antissimétrica pelos sinais, porém a recíproca é falsa, como podemos ver no Exemplo 2.10.

Exemplo 2.10 Considere as seguintes matrizes:

$$B = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}, \quad B_1 = \begin{pmatrix} 0 & 2 & 0 \\ -3 & 0 & 5 \\ 0 & -5 & 0 \end{pmatrix} \text{ e } \quad B_2 = \begin{pmatrix} 0 & 2 & 2 \\ -3 & 0 & 5 \\ -3 & -7 & 0 \end{pmatrix}.$$

A matriz B é antissimétrica e a matriz B_1 é antissimetrizável com matriz antissimetrizante $D_1 = (3,2,2) = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$, pois $D_1 \times B_1 = \begin{pmatrix} 0 & 6 & 0 \\ -6 & 0 & 10 \\ 0 & -10 & 0 \end{pmatrix}$.

A matriz B_2 é antissimétrica pelos sinais, porém não é antissimetrizável. De fato, se existisse uma matriz diagonal antissimetrizante para a matriz B_2 dada por (a,b,c), então $D_2 \times B_2 = \begin{pmatrix} 0 & 2\cdot a & 2\cdot a \\ -3\cdot b & 0 & 5\cdot b \\ -3\cdot c & -7\cdot c & 0 \end{pmatrix}$ seria antissimétrica, ou seja, $2\cdot a = 3\cdot b$, $2\cdot a = 3\cdot c$ e $5\cdot b = 7\cdot c$. Isto levaria a dizer que $b = c$ e $5\cdot b = 7\cdot c$, uma contradição.

2.6 Companheira quase-Cartan

Nesta seção, estudamos as matrizes companheiras quase-Cartan e mostramos uma maneira simples de encontrar todas elas.

Definição 2.11 *Uma matriz simetrizável C é **quase-Cartan** se todas as entradas da diagonal principal são iguais a 2.*

Uma *matriz de Cartan generalizada* é uma matriz quase-Cartan, cujas entradas fora da diagonal principal são não positivas. Uma *matriz de Cartan* é uma matriz de Cartan generalizada cuja matriz simetrizada é positiva definida. Definiremos a positividade de matrizes na Seção 2.8.

Álgebras de Kac-Moody correspondem à matrizes de Cartan generalizadas. Uma álgebra de Kac-Moody é de dimensão finita se e somente se a correspondente matriz é de Cartan. Então, as matrizes de Cartan representam a base da classificação Cartan-Killing. Portanto, todas as tais matrizes podem ser transformadas, por permutações simultâneas de linhas e colunas, em uma matriz bloco-diagonal na qual os blocos diagonais são de tipos familiares: $\mathbb{A}_n, \mathbb{B}_n, \mathbb{C}_n, \mathbb{D}_n, \mathbb{E}_6, \mathbb{E}_7, \mathbb{E}_8, \mathbb{F}_4$ e $\mathbb{G}_2$, representados pelos diagramas de Dynkin na Figura 2.17. As álgebras *cluster* de tipo finito coincidem com a famosa classificação de Cartan-Killing das álgebras de Kac-Moody de tipo finito. Como veremos no Capítulo 3, as álgebras *cluster* correspondem às matrizes antissimetrizáveis enquanto, que as álgebras de Kac-Moody correspondem às matrizes simetrizáveis.

Exemplo 2.12 *(a)* $\begin{pmatrix} 2 & 3 \\ 2 & 2 \end{pmatrix}$ *é uma matriz quase-Cartan.*

(b) $\begin{pmatrix} 2 & -3 \\ -2 & 2 \end{pmatrix}$ *é uma matriz de Cartan generalizada.*

(c) $\begin{pmatrix} 2 & -3 \\ -1 & 2 \end{pmatrix}$ *é uma matriz de Cartan.*

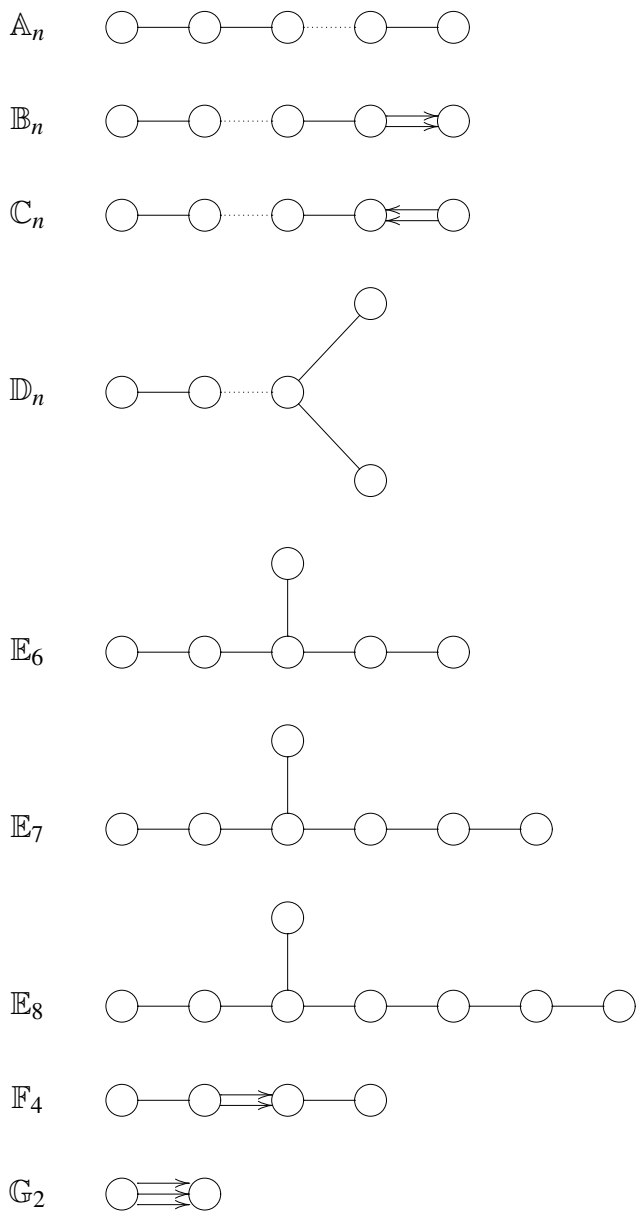


Figura 2.17: *Diagramas de Dynkin.*

Definição 2.13 *Para uma matriz antissimetrizável B, iremos nos referir à matriz quase-Cartan C, com* $|c_{ij}| = |b_{ij}|$*, para todo* $i \neq j$*, como uma* ***companheira quase-Cartan*** *de B.*

Fomin e Zelevinsky [17] mostram uma caracterização interessante para matrizes antissimetrizáveis.

Lema 2.14 (Fomin e Zelevinsky [17]) *Uma matriz B é antissimetrizável se e somente se ela é antissimétrica pelos sinais e para todo* $k \geq 3$ *e todo* $i_1, i_2, \ldots, i_k$ *ela satisfaz:*

$$b_{i_1 i_2} \cdot b_{i_2 i_3} \cdot \ldots \cdot b_{i_k i_1} = (-1)^k \cdot b_{i_2 i_1} \cdot b_{i_3 i_2} \cdot \ldots \cdot b_{i_1 i_k}. \tag{2-1}$$

Similarmente ao lema anterior, temos o resultado seguinte para matrizes simetrizáveis.

Lema 2.15 *Uma matriz C é simetrizável se e somente se ela é simétrica pelos sinais e para todo* $k \geq 3$ *e todo* $i_1, i_2, \ldots, i_k$ *ela satisfaz:*

$$c_{i_1 i_2} \cdot c_{i_2 i_3} \cdot \ldots \cdot c_{i_k i_1} = c_{i_2 i_1} \cdot c_{i_3 i_2} \cdot \ldots \cdot c_{i_1 i_k}. \tag{2-2}$$

Isso nos permite formular o Teorema 2.16, que fornece uma maneira de encontrar todas as companheiras quase-Cartan de uma matriz antissimetrizável.

Teorema 2.16 *Seja B uma matriz antissimetrizável. Considere C uma matriz tal que* $|c_{ij}| = |b_{ij}|$*, para todo* $i \neq j$*. Se C é simétrica pelos sinais, então C é simetrizável com o mesmo simetrizante de B. Além disso, se* $c_{ii} = 2$*, para todo i, então C é uma companheira quase-Cartan de B.*

Prova. Seja C uma matriz tal que $|c_{ij}| = |b_{ij}|$, para todo $i \neq j$. Pelo Lema 2.14, temos que $|b_{i_1 i_2} \cdot b_{i_2 i_3} \cdot \ldots \cdot b_{i_k i_1}| = |b_{i_2 i_1} \cdot b_{i_3 i_2} \cdot \ldots \cdot b_{i_1 i_k}|$. Logo, $|c_{i_1 i_2} \cdot c_{i_2 i_3} \cdot \ldots \cdot c_{i_k i_1}| = |c_{i_2 i_1} \cdot c_{i_3 i_2} \cdot \ldots \cdot c_{i_1 i_k}|$. Como C é simétrica pelos sinais, temos $c_{i_1 i_2} \cdot c_{i_2 i_3} \cdot \ldots \cdot c_{i_k i_1} = c_{i_2 i_1} \cdot c_{i_3 i_2} \cdot \ldots \cdot c_{i_1 i_k}$. Pelo Lema 2.15, temos que C é simetrizável. Claramente, se $c_{ii} = 2$, para todo i, então C é uma companheira quase-Cartan de B.

Uma outra demonstração nos permite ver que a matriz simetrizante das duas matrizes é a mesma. Seja $D = (d_i)$ uma matriz simetrizante de B. Lembramos que D é uma matriz diagonal com entradas positivas. Então, temos que B é uma matriz antissimetrizável, ou seja, $d_i \cdot b_{ij} = -d_j \cdot b_{ji}$. Logo, $d_i \cdot |b_{ij}| = d_j \cdot |b_{ji}|$, para todo i, j. Visto que $|c_{ij}| = |b_{ij}|$, temos que $d_i \cdot |c_{ij}| = d_i \cdot |b_{ij}| = |d_i \cdot b_{ij}| = |d_j \cdot b_{ji}| = d_j \cdot |b_{ji}| = d_j \cdot |c_{ji}|$. Como C é simétrica pelos sinais, temos que $d_i \cdot c_{ij} = d_j \cdot c_{ji}$.

Portanto, B e C têm a mesma matriz simetrizante. □

Observação 2.17 *Dada uma matriz antissimetrizável B, para construir uma companheira quase-Cartan C, devemos escolher somente os sinais dos elementos acima da diagonal principal. Há 2 sinais e* $\frac{n\cdot(n-1)}{2}$ *elementos acima da diagonal principal, onde n é a quantidade de linhas ou colunas. Então, há no máximo* $\frac{n\cdot(n-1)}{2}$ *diferentes formas de obter uma matriz tal que* $sgn(c_{ij}) = sgn(c_{ji})$ *e* $|c_{ij}| = |b_{ij}|$ *para* $i \neq j$*. De fato, B tem* 2^m *companheiras quase-Cartan, onde m é o número de arestas do grafo associado a B ou equivalentemente é a quantidade de elementos não nulos acima da diagonal principal.*

Exemplo 2.18 *Sejam* $a, b \in \mathbb{Z}_+$ *e* $B = \begin{pmatrix} 0 & a \\ -b & 0 \end{pmatrix}$*. Se B é uma matriz antissimetrizável, então,* $\begin{pmatrix} 2 & a \\ b & 2 \end{pmatrix}$ *e* $\begin{pmatrix} 2 & -a \\ -b & 2 \end{pmatrix}$ *são as possíveis companheiras quase-Cartan de B.*

Exemplo 2.19 *Dado que* $B = \begin{pmatrix} 0 & -2 & 0 \\ 1 & 0 & 1 \\ 0 & -2 & 0 \end{pmatrix}$ *é uma matriz antissimetrizável,* $\begin{pmatrix} 2 & 2 & 0 \\ 1 & 2 & 1 \\ 0 & 2 & 2 \end{pmatrix}$, $\begin{pmatrix} 2 & -2 & 0 \\ -1 & 2 & 1 \\ 0 & 2 & 2 \end{pmatrix}$, $\begin{pmatrix} 2 & 2 & 0 \\ 1 & 2 & -1 \\ 0 & -2 & 2 \end{pmatrix}$ *e* $\begin{pmatrix} 2 & -2 & 0 \\ -1 & 2 & -1 \\ 0 & -2 & 2 \end{pmatrix}$ *são as possíveis companheiras quase-Cartan de B.*

Dada uma matriz antissimetrizável B de ordem n, associamos um grafo orientado $G(B)$ com vértices $\{1,2,\ldots,n\}$ e arestas (i,j) quando $b_{ij} > 0$, com $i, j \in \{1,\ldots,n\}$. O peso de uma aresta (i,j) desse grafo é definido como $|b_{ij} \cdot b_{ji}|$.

O *grafo* $G(C)$ de uma matriz quase-Cartan C é um grafo não orientado com vértices $\{1,2,\ldots,n\}$ e arestas (i,j), para cada $i \neq j$ com $c_{ij} \neq 0$, onde a toda aresta (i,j) é atribuído um *peso* $c_{ij} \cdot c_{ji}$ e um *sinal* $\varepsilon_{ij} = -sgn(c_{ij}) = -sgn(c_{ji})$.

Exemplo 2.20 *Uma matriz simétrica* $C = \begin{pmatrix} 2 & -1 & 1 & 0 & 0 \\ -1 & 2 & -1 & 1 & 0 \\ 1 & -1 & 2 & -1 & 1 \\ 0 & 1 & -1 & 2 & -1 \\ 0 & 0 & 1 & -1 & 2 \end{pmatrix}$*, que é companheira quase-Cartan da seguinte matriz antissimétrica:*

$$B = \begin{pmatrix} 0 & -1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 & 0 \\ -1 & 1 & 0 & -1 & 1 \\ 0 & -1 & 1 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \end{pmatrix}$$

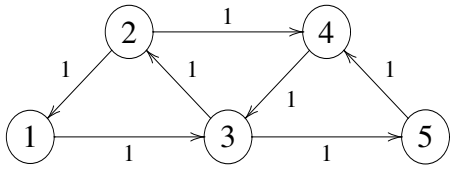


Figura 2.18: *Grafo da matriz antissimétrica B.*

Em desenhos de grafos, todos os pesos não especificados são assumidos como iguais a 1. Um grafo ponderado G na forma $G(C)$ deve satisfazer a seguinte condição: o produto de todas arestas ponderadas ao longo de todo ciclo de G é um quadrado perfeito (utilizando só o peso, não o sinal) (veja [33], Exercício 2.1).

Exemplo 2.21 Duas matrizes antissimetrizáveis distintas podem ter o mesmo grafo. Na Figura 2.19, temos um grafo que representa as matrizes A e B.

$$A = \begin{pmatrix} 0 & 4 & -2 \\ -1 & 0 & 1 \\ 1 & -2 & 0 \end{pmatrix} \text{ e } B = \begin{pmatrix} 0 & 2 & -1 \\ -2 & 0 & 1 \\ 2 & -2 & 0 \end{pmatrix}.$$

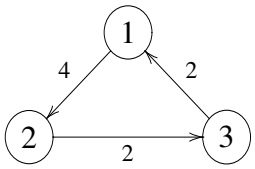


Figura 2.19: *Grafo das matrizes antissimétricas A e B.*

Um *grafo ponderado G* é chamado de *positivo* se alguma atribuição de sinais às arestas fizer com que o grafo G seja o de alguma matriz quase-Cartan positiva C. Estudaremos matrizes positivas de maneira mais aprofundada na Seção 2.8.

2.7 Algoritmos referentes à matrizes simetrizáveis

Nesta seção, elaboramos dois algoritmos: um algoritmo para decidir se uma matriz tem simetrizante e apresentá-la, caso exista, e outro para encontrar a matriz simetrizante para uma matriz simetrizável. O segundo algoritmo tem complexidade de tempo $\Theta(n^2)$ no pior caso e $\Theta(n)$ no melhor caso, e o primeiro algoritmo tem complexidade de tempo $\Theta(n^2)$ para matrizes simetrizáveis, onde n é a dimensão. Se é conhecida a informação que a matriz dada é simetrizável, é mais aconselhável usar o segundo algoritmo.

Algoritmo 2.6: $MatrizSimetrizavel(A)$

Entrada: Uma matriz A de ordem n.

Saída: Resposta se a matriz é simetrizável e, se for, uma matriz diagonal $D = (d_{ii})$ com valores positivos tal que $D \times A$ é simétrica.

```
1  inicialize d_ii ← 0 para todo i ∈ {1,...,n}
2  T ← {1,...,n} uma lista ordenada
3  enquanto (T ≠ ∅) faça
4      i ← o primeiro elemento de T
5      T ← T − {i}
6      se (d_ii = 0) então
7          d_ii ← 1
8      para cada j ∈ T faça
9          se (a_ij · a_ji = 0) então
10             se (a_ij + a_ji ≠ 0) então
11                 retorna NÃO
12         senão
13             mova j para a primeira posição de T
14             se (d_jj ≠ 0) então
15                 se (d_ii · a_ij ≠ d_jj · a_ji) então
16                     retorna NÃO
17             senão
18                 d_jj ← (d_ii · a_ij) / a_ji
19 retorna SIM, D
```

Os números de linha na Proposição 2.22 se referem às linhas do Algoritmo 2.6. Se A é conexa, então a Linha 7 é executada somente uma vez, porque a atribuição é feita para cada componente conexo do grafo $G(A)$ associado a A.

Proposição 2.22 *O Algoritmo* 2.6 *é correto.*

Prova. No início de qualquer iteração do `enquanto`, $d_{ii} \cdot a_{ij} = d_{jj} \cdot a_{ji}$ para qualquer $j \notin T$ e qualquer i. Isto claramente é verdade visto que no início não existe nenhum $j \notin T$. No final de cada iteração do `enquanto`, o elemento i removido de T satisfaz esta condição, pois senão o algoritmo retorna NÃO na Linha 16. Portanto, se o algoritmo retorna D, temos que $d_{ii} \cdot a_{ij} = d_{jj} \cdot a_{ji}$ para quaisquer $1 \leq i, j \leq n$.

Suponha que o algoritmo retorna NÃO na Linha 16, quando A é simetrizável. Portanto, existem i e j tais que $d_{ii} \cdot a_{ij} \neq d_{jj} \cdot a_{ji}$.

Seja T a lista $\langle i_1, \ldots, i_t \rangle$ no início do `enquanto`. Segue da Linha 13 que os primeiros índices $\star$ de T têm $d_{\star\star} \neq 0$ e os últimos têm $d_{\star\star} = 0$, isto é, existe $k \in \{0, \ldots, t\}$ tal que $d_{i_s i_s} \neq 0$ para $1 \leq s \leq k$ e $d_{i_s i_s} = 0$ para $k < s \leq t$. Se $k = 0$, $d_{\star\star} = 0$, para todo $\star \in T$. Visto que i é o primeiro elemento em T (Linha 4) e $d_{jj} \neq 0$ (Linha 14), temos que $d_{ii} \neq 0$ no início do `enquanto`. Portanto, d_{ii} e d_{jj} foram definidos antes e existe $k \geq 3$, $i_1 = i, i_2 = j$ e $i_3, \ldots, i_k \notin T$ tal que $a_{i_1 i_2} \cdot a_{i_2 i_3} \cdot \ldots \cdot a_{i_k i_1} \neq 0$. Uma vez que A é simetrizável, temos pelo Lema 2.15 que $a_{i_1 i_2} \cdot a_{i_2 i_3} \cdot \ldots \cdot a_{i_k i_1} = a_{i_2 i_1} \cdot a_{i_3 i_2} \cdot \ldots \cdot a_{i_1 i_k}$. Isto implica que $d_{ii} \cdot a_{ij} = d_{jj} \cdot a_{ji}$, uma contradição. □

O próximo algoritmo faz, essencialmente, o mesmo que o anterior. A diferença é que não é verificado se A é simetrizável.

Algoritmo 2.7: $MatrizSimetrizante(A)$

Entrada: Uma matriz simetrizável A.

Saída: Uma matriz diagonal $D = d_{ii}$ de valores positivos tal que $D \times A$ é simétrica.

```
1  inicialize d_ii ← 0 para todo i ∈ {1,...,n}
2  S ← {1,...,n}
3  T ← {1,...,n} é uma lista ordenada
4  enquanto (S ≠ ∅) faça
5      i ← o primeiro elemento de T
6      T ← T − {i}
7      se (d_ii = 0) então
8          d_ii ← 1
9          S ← S − {i}
10     para cada j ∈ T faça
11         se (a_ji ≠ 0) então
12             mova j para a primeira posição de T
13             se (d_jj = 0) então
14                 d_jj ← (d_ii · a_ij) / a_ji
15                 S ← S − {j}
16 retorna D
```

Proposição 2.23 *O Algoritmo* 2.7 *é correto.*

Prova. Os números de linha nesta prova se referem às linhas do Algoritmo 2.7, que é bem similar ao Algoritmo 2.6. As diferenças são que o Algoritmo 2.7 não verifica se A é simetrizável e usa uma lista adicional S na qual mantém os elementos i tal que d_{ii} não foram definidos ainda. Visto que o algoritmo assume que a matriz de entrada A é simetrizável, isto permite parar a execução quando todos os elementos de D já foram

definidos. Este controle é feito pela alteração na condição do `enquanto` e pela adição de operações para remover o elemento i de S sempre que d_{ii} é definido (Linhas 9 e 15). □

2.8 Matrizes positivas

Definimos a ij-ésima *submatriz* de uma matriz A de ordem n, para $n > 1$, como a matriz $A_{[ij]}$, de ordem $(n-1)$, obtida mediante a omissão da i-ésima linha e da j-ésima coluna da matriz A. O *ij-ésimo menor de A* é, por definição, o determinante de $A_{[ij]}$, o qual é chamado também de *menor relativo ao elemento a_{ij}*. O *determinante* de uma matriz A pode ser definido em termos dos seus menores por: $det(A) = \sum_{j=1}^{n} \left((-1)^{i+j} \cdot a_{ij} \cdot det(A_{[ij]})\right)$.

Como definimos anteriormente, uma submatriz de A é a matriz obtida de A eliminando-se algumas linhas e colunas. Uma submatriz quadrada cujos elementos da diagonal principal são elementos da diagonal principal de A é chamada de *submatriz principal*. A k-ésima *submatriz principal líder* de A, matriz A_k, consiste da interseção das k primeiras linhas e das k primeiras colunas de A. Uma submatriz principal é obtida recursivamente da matriz $A_{[ii]}$, ou seja, é igual a $((A_{[i_1 i_1]})\ldots)_{[i_t i_t]}$.

Os *menores principais* são os determinantes de todas as submatrizes principais de A. Os *menores principais líderes* são os determinantes das submatrizes principais líderes. Veja [21] para maiores informações sobre o assunto.

Uma matriz simétrica A de ordem n é *positiva definida* se $x^T \cdot A \cdot x > 0$ para todos os vetores x de tamanho n, com $x \neq 0$. Por razões de simplicidade, usaremos apenas o termo *matriz positiva* ao invés de matriz positiva definida.

Exemplo 2.24 *Um exemplo de matriz positiva é a matriz identidade, onde para qualquer vetor não nulo $x = (x_1 \cdot x_2 \cdot \ldots \cdot x_n)^T$, temos $x^T \cdot I_n \cdot x = x^T \cdot x = \sum_{i=1}^{n} x_i^2 > 0$.*

Pelo critério de Sylvester, ser positiva significa que os menores principais são todos positivos.

Teorema 2.25 (Critério de Sylvester [8]) *Seja A uma matriz simétrica. As seguintes condições são equivalentes:*

(a) A é positiva;
(b) todos os menores principais de A são positivos; e
(c) todos os menores principais líderes de A são positivos.

Exemplo 2.26 *A matriz simétrica* $A = \begin{pmatrix} 2 & -1 & 1 \\ -1 & 2 & -1 \\ 1 & -1 & 2 \end{pmatrix}$ *é um exemplo de matriz quase-Cartan positiva. Outro exemplo é a matriz do Exemplo 2.20.*

Uma matriz simetrizável é positiva se a sua matriz simetrizada também é. Para verificar se uma matriz simetrizável é positiva, podemos calcular o determinante de cada submatriz principal e verificar se cada um deles é positivo.

Proposição 2.27 *Seja C uma matriz simetrizável e D uma matriz diagonal com entradas positivas. Os menores principais de $D \times C$ são positivos se e somente se os de C também são positivos.*

Prova. Observe que $(D \times C)_{[ii]} = D_{[ii]} \times C_{[ii]}$. Logo, $det((D \times C)_{[ii]}) = det(D_{[ii]}) \times det(C_{[ii]}) = d_1 \cdot \ldots \cdot d_{i-1} \cdot d_{i+1} \ldots d_n \cdot det(C_{[ii]})$.

Como $d_i > 0$ para todo i, temos que $det(C_{[ii]}) > 0$ se e somente se $det((D \times C)_{[ii]}) > 0$. O resultado segue por indução. □

O teorema abaixo segue do critério de Sylvester e da proposição anterior.

Teorema 2.28 *Seja C uma matriz simetrizável. As seguintes condições são equivalentes:*

(a) C é positiva;
(b) todos os menores principais de C são positivos; e
(c) todos os menores principais líderes de C são positivos.

□

Os lemas seguintes mostram que as submatrizes principais preservam as propriedades de antissimetrizável e simetrizável, respectivamente.

Lema 2.29 *Se B é uma matriz antissimetrizável, então todas as submatrizes principais de B também são antissimetrizáveis.*

Prova. Seja B uma matriz antissimetrizável, ou seja, $D \times B$ é antissimétrica. Considere a matriz $(D \times B)_{[ii]}$ obtida pela eliminação da i-ésima linha e i-ésima coluna de $D \times B$. Devido a essa eliminação, temos que $(D \times B)_{[ii]} = D_{[ii]} \times B_{[ii]}$. Portanto, segue que $B_{[ii]}$ é antissimetrizável com antissimetrizante $D_{[ii]}$. O resultado segue por indução. □

Lema 2.30 *Se C é uma matriz simetrizável, então todas as submatrizes principais também são.*

Prova. Seja C uma matriz simetrizável, ou seja, $D \times C$ é simétrica. Considere a matriz $(D \times C)_{[ii]}$ obtida pela eliminação da i-ésima e i-ésima coluna de $D \times C$. Devido a essa eliminação, temos que $(D \times C)_{[ii]} = D_{[ii]} \times C_{[ii]}$. Portanto, segue que $C_{[ii]}$ é simetrizável com simetrizante $D_{[ii]}$. O resultado segue por indução. □

Relacionamos os conceitos de subgrafos e submatrizes principais.

Observação 2.31 *Um **subgrafo** de $G(A)$ é um grafo $G(A')$, onde A' é a submatriz principal de A. Logo, $G(A')$ é obtido de $G(A)$ através do subgrafo induzido de um conjunto de vértices e mantendo-se as mesmas as arestas ponderadas e sinais de $G(A)$.*

Sendo $G(A')$ um subgrafo de $G(A)$, a positividade de A implica na de A' e para termos uma álgebra *cluster* de tipo finito nenhum dos subgrafos da proposição seguinte pode aparecer considerando a correspondência dos grafos com a classe de equivalência de mutações das matrizes.

Proposição 2.32 (Barot, Geiss e Zelevinsky [2]) *Nenhum dos grafos ponderados da Figura 2.20 pode aparecer como subgrafo do grafo de uma matriz quase-Cartan positiva:*

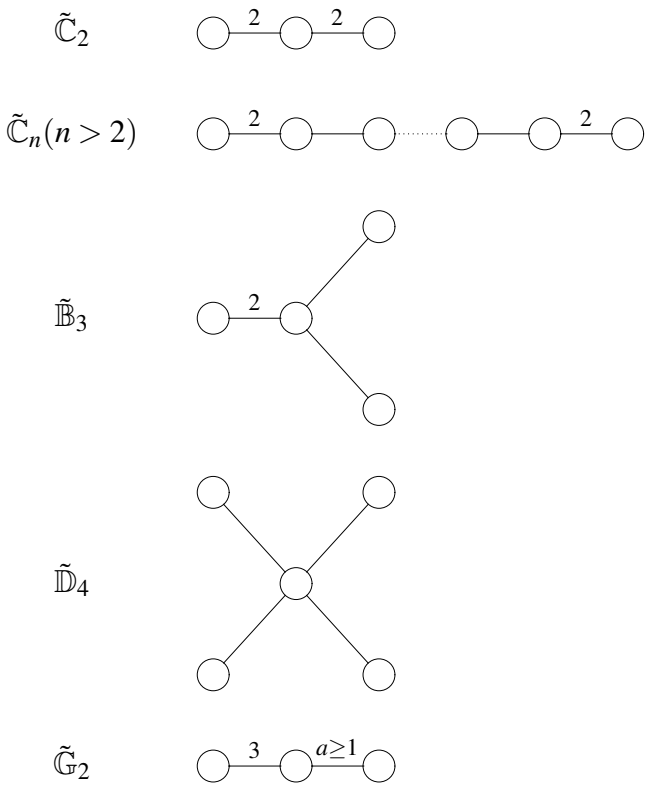
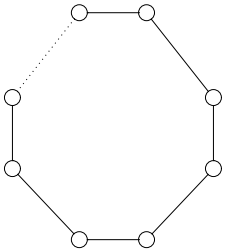
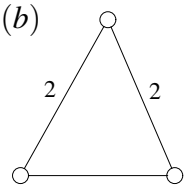


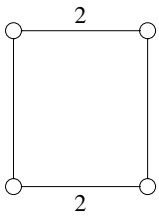
Figura 2.20: *Subgrafos proibidos:* $\tilde{\mathbb{C}}_2$, $\tilde{\mathbb{C}}_n (n > 2)$, $\tilde{\mathbb{B}}_3$, $\tilde{\mathbb{D}}_4$ *e* $\tilde{\mathbb{G}}_2$.

Para a próxima proposição, relembre que arestas sem peso convencionalmente têm peso 1.

Proposição 2.33 (Barot, Geiss e Zelevinsky [2]) *Ciclos com pesos positivos em suas arestas são precisamente aqueles que seguem dos três seguintes tipos:*

(a) 

(b) 

(c) 

Além disso, uma atribuição de sinais faz cada um destes ciclos ser o diagrama de uma matriz quase-Cartan positiva se e somente se o produto dos sinais ao longo de todas as arestas é igual a -1.

CAPÍTULO 3

Álgebras Cluster: Identificando o Problema

No início deste milênio, Sergey Fomin e Andrei Zelevinsky [16] introduziram uma classe de álgebra comutativa chamada álgebras *cluster*, que são definidas recursivamente através de álgebras comutativas com conjuntos de variáveis geradoras (*variáveis cluster*) agrupados em subconjuntos sobrepostos (*clusters*) de cardinalidade fixa. Essas álgebras têm uma forte estrutura combinatória e são ferramentas para estudar questões de bases canônicas duais e positividade de grupos de Lie semissimples.

Uma característica básica da álgebra *cluster* é que tanto os geradores quanto as relações entre eles não são dados desde o início, mas são produzidos por um processo iterativo elementar de mutação da semente. Este processo, aparentemente contraintuitivo, parece codificar um fenômeno universal. Isto pode explicar o desenvolvimento acelerado do tema álgebra *cluster* em áreas como análise combinatória, física, matemática (especialmente geometria), entre outros, como discutido em [20, 25, 39]. Essas álgebras podem ser definidas usando-se uma matriz antissimetrizável, ou de maneira equivalente um *quiver* sem laços ou 2-ciclos, como veremos na Seção 3.2.

Neste capítulo, estudaremos as álgebras *cluster*, especificamente as de tipo finito, para identificar o problema objeto desta tese.

3.1 Introdução à álgebra cluster

Definimos álgebra *cluster* a partir do conceito de matriz antissimetrizável. Mas, antes, é necessário definir alguns conceitos importantes, começando com o de semente.

Considere o corpo $\mathcal{F} = \mathbb{Q}(x_1, \ldots, x_n)$ de frações de polinômios em n variáveis sobre $\mathbb{Q}$.

Definição 3.1 *Uma* ***semente*** *em $\mathcal{F}$ é um par $(mathcalX, B)$ onde:*

- *$\mathcal{X}$ é um conjunto tal que $\{x_1, \ldots, x_n\} \subset \mathcal{F}$ é um subconjunto de n elementos algebricamente independentes que geram $\mathcal{F}$. O conjunto $\mathcal{X}$ é chamado de base de transcendência de $\mathcal{F}$; e*
- *B é uma matriz antissimétrizável pertencente a $\mathrm{M}_n(\mathbb{Z})$.*

O conjunto $\mathcal{X}$ é chamado de ***cluster*** *e os elementos de $\mathcal{X}$ são chamados de* ***variáveis cluster*** *da semente $(\mathcal{X}, B)$.*

O exemplo seguinte ilustra a definição anterior.

Exemplo 3.2 Considere $n = 2$, com $\mathcal{F} = \mathbb{Q}(x_1, x_2)$ e $\mathcal{X} = \{x_1, x_2\}$. Os pares $\left(\mathcal{X}, \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\right)$ e $\left(\mathcal{X}' = \{\frac{1+x_2}{x_1}, x_2\}, \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\right)$ são duas sementes em $\mathcal{F}$.

A matriz antissimetrizável B fornece a *regra de mudança* para passar de uma semente a outra, mas, primeiramente, é necessário entender como criar um novo elemento de $\mathcal{F}$.

Definição 3.3 *Seja $(\mathcal{X}, B)$ uma semente em $\mathcal{F}$, com $B = (b_{ij})$. Para cada $k \in \{1, \ldots, n\}$, definimos um* **novo elemento** *x'_k em $\mathcal{F}$ da seguinte maneira:*

$$x_k \cdot x'_k = \prod_{b_{ik}>0} x_i^{b_{ik}} + \prod_{b_{jk}<0} x_j^{-b_{jk}}$$

com a convenção usual que o produto vazio vale 1. Dessa forma, temos o novo conjunto de variáveis:

$$\mathcal{X}_k = (\mathcal{X} - \{x_k\}) \cup \{x'_k\}.$$

Estas relações são chamadas de **relações de mudança** *e a matriz B é chamada de* **matriz de mudança** *da semente $(\mathcal{X}, B)$.*

Antes de definir mutação de semente, é necessário definir mutação matricial.

Definição 3.4 *Uma matriz $B' = (b'_{ij}) \in \mathrm{M}_n(\mathbb{Z})$ é obtida a partir de uma matriz $B = (b_{ij}) \in \mathrm{M}_n(\mathbb{Z})$ por* ***mutação de matriz na direção*** *k, denotada por μ_k, com $k \in \{1, \ldots, n\}$, se os coeficientes de B' são dados pela seguinte fórmula:*

$$b'_{ij} = \begin{cases} -b_{ij}, & \text{se } i = k \text{ ou } j = k; \\ b_{ij} + \frac{|b_{ik}| \cdot b_{kj} + b_{ik} \cdot |b_{kj}|}{2}, & \text{caso contrário.} \end{cases}$$

Neste caso, $B' = \mu_k(B)$.

Alguns artigos utilizam a notação $b_{ij} + sgn(b_{ik}) \cdot [b_{ik} \cdot b_{kj}]_+$ no caso contrário, onde $[x]_+ = \max(x, 0)$ e $sgn(x) = \frac{x}{|x|}$. Por convenção, definimos que $sgn(0) = 0$.

Exemplo 3.5 $B = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & 2 \\ -1 & -2 & 0 \end{pmatrix}$ *é antissimetrizável e temos as mutações abaixo:* $\mu_1(B) = \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 3 \\ 1 & -3 & 0 \end{pmatrix}$, $\mu_2(B) = \begin{pmatrix} 0 & 1 & 1 \\ -1 & 0 & -2 \\ -1 & 2 & 0 \end{pmatrix}$ *e* $\mu_3(B) = \begin{pmatrix} 0 & -1 & -1 \\ 1 & 0 & -2 \\ -1 & 2 & 0 \end{pmatrix}$.

Observação 3.6 *Se B é uma matriz antissimetrizável, então $B' = \mu_k(B)$ é antissimetrizável. Além disso, as mutações formam uma relação de equivalência no conjunto das matrizes antissimetrizáveis.*

O lema seguinte fornece algumas propriedades que permitem definir a mutação de semente.

Lema 3.7 (Fomin e Zelevinsky [16]) *Seja $(\mathcal{X}, B)$ uma semente em $\mathcal{F}$ e $k \in \{1, \ldots, n\}$. As seguintes propriedades são válidas:*

(a) a mutação matricial é involutiva, ou seja, $\mu_k(\mu_k(B)) = B$; e
(b) o par $(\mathcal{X}_k, \mu_k(B))$ é uma semente em $\mathcal{F}$. Além disso, as matrizes B e $\mu_k(B)$ compartilham a mesma matriz antissimetrizante.

Definição 3.8 *Seja $(\mathcal{X}, B)$ uma semente em $\mathcal{F}$. Para cada $k \in \{1, \ldots, n\}$, definimos a* ***mutação de semente na direção*** *k $(\mathcal{X}, B)$ como a semente $\mu_k(\mathcal{X}, B) = (\mathcal{X}_k, \mu_k(B))$.*

Exemplo 3.9 Considere $n = 2$, $\mathcal{F} = \mathbb{Q}(x_1, x_2)$, $\mathcal{X} = \{x_1, x_2\}$ e a semente $S = (\mathcal{X}, B) = \left(\{x_1, x_2\}, \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \right)$.

Para $k = 1$, temos que $x_1 \cdot x_1' = 1 + x_2$. Logo, $x_1' = \frac{1+x_2}{x_1}$ e podemos concluir que $\mu_1(S) = S' = (\mathcal{X}', B') = \left(\left\{ \frac{1+x_2}{x_1}, x_2 \right\}, \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \right)$.

Para $k = 2$, temos que $x_2 \cdot x_2' = x_1' + 1 = \frac{1+x_2}{x_1} + 1$. Assim, $x_2' = \frac{1+x_1+x_2}{x_1 \cdot x_2}$ e podemos concluir que $\mu_2(S') = S'' = (\mathcal{X}'', B'') = \left(\left\{ \frac{1+x_2}{x_1}, \frac{1+x_1+x_2}{x_1 \cdot x_2} \right\}, \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \right)$.

Finalmente, podemos definir a álgebra *cluster* e seus componentes.

Definição 3.10 *Seja B uma matriz antissimetrizável em $\mathrm{M}_n(\mathbb{Z})$. Dada a semente $(\mathcal{X}, B)$ com $\mathcal{X} = \{x_1, \ldots x_n\}$, temos que:*

(a) os clusters com respeito a B são os conjuntos $\mathcal{X}'$ tais que existe uma semente $(\mathcal{X}', B')$ obtida de $(\mathcal{X}, B)$ por uma sucessão de mutações;

(b) as variáveis cluster para B são os elementos da união de todos os clusters; e

(c) a álgebra cluster $\mathcal{A}(B) = \mathcal{A}(\mathcal{X}, B)$ é a $\mathbb{Q}$-subálgebra do corpo $\mathcal{F} = \mathbb{Q}(x_1, \ldots, x_n)$ gerada pelas variáveis cluster.

Veja o conceito da definição anterior ilustrado no exemplo seguinte.

Exemplo 3.11 Considere $n = 2$, $\mathcal{F} = \mathbb{Q}(x_1, x_2)$, $\mathcal{X} = \{x_1, x_2\}$, $B = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, $B' = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ e a semente inicial $(\mathcal{X}, B)$. Temos que $\mu_k(B) = B'$ e $\mu_k(B') = B$ para $k = 1, 2$.

Fixe $\mathcal{X} = \{y_t, \text{ com } t \in \mathbb{Z}\}$ como sendo o conjunto das variáveis *cluster* para B. Os y_t satisfazem a seguinte relação de mudança: $y_t = \frac{y_{t-1}+1}{y_{t-2}}$.

Encontramos, sucessivamente, os seguintes valores

$$y_3 = \frac{1+x_2}{x_1},\ y_4 = \frac{1+x_1+x_2}{x_1 \cdot x_2},\ y_5 = \frac{1+x_1}{x_2},\ y_6 = x_1 = y_1,\ y_7 = x_2 = y_2,\ \ldots$$

Então, a álgebra *cluster* $\mathcal{A}(B)$ é dada por $\mathbb{Z}\left[x_1, x_2, \frac{1+x_2}{x_1}, \frac{1+x_1+x_2}{x_1 \cdot x_2}, \frac{1+x_1}{x_2}\right]$.

No Exemplo 3.11, notamos que as variáveis *cluster* se expressam como frações racionais nos x_i cujos denominadores são sempre monômios, ou seja, são polinômios de Laurent. De fato, as variáveis *cluster* sempre são polinômios de Laurent [45].

3.2 Quivers associados a matrizes antissimétricas

Um *quiver* Q é um grafo múltiplo orientado. Portanto, é dado por uma quádrupla formada por um conjunto Q_0 de vértices, um conjunto Q_1 de arestas e duas aplicações $s, t : Q_1 \to Q_0$ que associam a uma aresta seu início e término. Um *laço* é uma aresta cujo início corresponde ao seu término; um *2-ciclo* é um par de duas arestas distintas $\alpha \neq \beta$ tal que o início de α corresponde ao término de β e vice-versa. Um grafo orientado é finito se ambos os conjuntos de vértices e de arestas também o são.

Seja B uma matriz antissimetrizável de ordem n. Então, associamos um *quiver* $Q(B)$ com $V(Q) = \{1, \ldots, n\}$ e $E(Q) = \{(i, j) \mid b_{ij} > 0\}$ a esta matriz.

Os *quivers* sem laços ou 2-ciclos correspondem às matrizes antissimétricas e isto provê uma maneira eficiente e visual de lidar com as regras de mudanças das sementes.

Seja n um inteiro positivo e $\mathcal{F} = \mathbb{Q}(x_1, \ldots, x_n)$. Denote por $\mathrm{Skew}_n(\mathbb{Z})$ o conjunto das matrizes antissimétricas de $\mathrm{M}_n(\mathbb{Z})$. Para cada matriz antissimétrica $B \in \mathrm{Skew}_n(\mathbb{Z})$, podemos associar um *quiver* $Q(B)$ da seguinte maneira:

- os vértices de $Q(B)$ são enumerados de 1 a n; e
- se o coeficiente $b_{ij} > 0$, então $Q(B)$ admite b_{ij} arestas distintas de i até j.

O lema seguinte provê a correspondência desejada entre *quivers* sem laços nem 2-ciclos e matrizes antissimétricas.

Lema 3.12 *Considere $\mathcal{Q}_n$ a família de quivers finitos com n vértices, sem laços ou 2-ciclos. A correspondência Q define uma bijeção entre $Skew_n(\mathbb{Z})$ e $\mathcal{Q}_n$.*

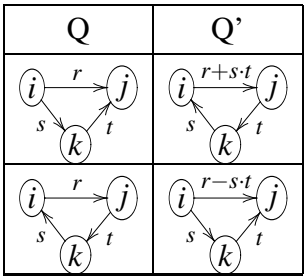
O grafo simples associado a $Q(B)$ é o grafo $G(B)$ e o peso de (i, j) em $G(B)$ corresponde à quantidade de arestas entre i e j em $Q(B)$ ao quadrado.

3.3 Sementes e mutações em quivers

Seja n um inteiro positivo. Nesta seção, será considerada uma semente o par $(\mathcal{X}, Q(B))$ no lugar de $(\mathcal{X}, B)$. Em uma semente, todas as arestas entre dois vértices fixos no *quiver* vão na mesma direção. Seja $(\mathcal{X}, Q)$ uma semente e k um vértice de Q. A *mutação* $\mu_k(\mathcal{X}, Q)$ de $(\mathcal{X}, Q)$ na direção de k é a semente $(\mathcal{X}', Q')$, onde:

(a) Q' é obtido de Q da seguinte forma:

- reverta todas as arestas que incidem em k;
- para quaisquer vértices i, j distintos de k, tais que $i \neq j$, troque a quantidade de arestas entre i e j da seguinte maneira:



onde r, s, t são inteiros não negativos, uma aresta $i \xrightarrow{\ell} j$ com $\ell \geq 0$ significa que ℓ arestas vão de i até j e uma aresta $i \xrightarrow{\ell} j$ com $\ell \leq 0$ significa que $-\ell$ arestas vão de j até i; e

(b) $\mathcal{X}'$ é obtido a partir de $\mathcal{X}$ substituindo-se o elemento x_k segundo a seguinte regra:

$$x_k \cdot x_k' = \prod_{(i,k) \in E(Q)} x_i + \prod_{(k,j) \in E(Q)} x_j.$$

É interessante observar que, na relação de mudança, se não existirem arestas com início (ou término) em k, o produto associado é o produto sobre o conjunto vazio, ou seja, é igual a 1. Não é difícil ver que $\mu_k(\mathcal{X}, Q(B)) = (\mathcal{X}_k, Q(\mu_k(B)))$ e $\mu_k(\mu_k(\mathcal{X}, Q)) = (\mathcal{X}, Q)$. De maneira análoga, a álgebra *cluster* $\mathcal{A}(Q)$ é aquela em que $Q = Q(B)$.

3.4 Álgebras cluster de tipo finito associadas às matrizes simétricas

Tanto em teoria de álgebras *cluster* como de álgebras de Kac-Moody, existe a mesma classificação de objetos de tipo finito usando a forma de Cartan-Killing. As álgebras de Kac-Moody de tipo finito são as que têm dimensão finita, isto é, são as álgebras de Lie semissimples [2]. As álgebras *cluster* de tipo finito coincidem com a famosa classificação de Cartan-Killing de álgebras de Lie semissimples.

Álgebras *cluster* de *tipo finito* são aquelas que têm um número finito de variáveis *cluster*. A combinação destas duas classificações é diferente: as álgebras de Kac-Moody correspondem às matrizes de Cartan generalizadas (simetrizáveis), enquanto que as álgebras *cluster* correspondem às matrizes antissimetrizáveis.

A álgebra *cluster* do Exemplo 3.11 tem a particularidade de ser gerada por uma quantidade finita de variáveis *cluster*, ou seja, a álgebra *cluster* é de tipo finito. Em geral, não é o caso das álgebras *cluster*.

Lembramos que a álgebra *cluster* $\mathcal{A}(Q)$ é associada a uma matriz antissimétrica B. Teremos um resultado similar para matrizes antissimetrizáveis na Seção 3.6.

Teorema 3.13 (Fomin e Zelevinsky [17]) *Seja Q um quiver com n vértices, sem laços nem 2-ciclos e $\mathcal{A}(Q)$ a sua álgebra cluster. As seguintes condições são equivalentes:*

(a) $\mathcal{A}(Q)$ é de tipo finito;
(b) o conjunto de todas as sementes obtidas iterativamente a partir de $(\mathcal{X}, Q)$ é finito;
(c) $\mathcal{A}(Q)$ tem uma quantidade finita de clusters; e
(d) existe uma semente cujo grafo orientado é do tipo Dynkin (Figura 2.17).

Observação 3.14 *Se $\mathcal{A}(Q)$ é de tipo finito, claramente a quantidade de quivers associados às sementes é finita. Porém, a recíproca nem sempre é verdadeira. De fato, podemos considerar a álgebra cluster associada a $1 \rightrightarrows 2$. Pelo teorema anterior, ela não é de tipo finito, mas claramente possui somente dois quivers associados às sementes: $1 \rightrightarrows 2$ e $1 \leftleftarrows 2$.*

Exemplo 3.15 Considere a álgebra *cluster* $\mathcal{A}(B)$ do Exemplo 3.11. Esta álgebra corresponde à álgebra *cluster* $\mathcal{A}(Q)$ com $Q : 1 \to 2$, a qual, pelo Teorema 3.13, é de tipo finito. Existem 5 variáveis *cluster*, dadas por:

$$y_1 = x_1,\ y_2 = x_2,\ y_3 = \frac{1+x_2}{x_1},\ y_4 = \frac{1+x_1+x_2}{x_1 \cdot x_2} \text{ e } y_5 = \frac{1+x_1}{x_2}.$$

Notamos que *clusters* distintos podem ser associados ao mesmo *quiver*. Por exemplo, a semente $\mu_2(\mu_1(\mathcal{X}, Q))$ e a semente inicial $(\mathcal{X}, Q)$ têm o mesmo *quiver*. De fato, $\mu_1(\mathcal{X}, Q) = (\{y_3, y_2\}, 1 \leftarrow 2)$ e $\mu_2(\mu_1(\mathcal{X}, Q)) = (\{y_3, y_4\}, Q)$.

Por outro lado, sementes diferentes podem ter o mesmo *cluster*. Por exemplo, as sementes $\mu_1(\mu_2(\mu_1(\mathcal{X},Q))) = (\{y_5,y_4\}, 1 \leftarrow 2)$ e $\mu_1(\mu_2(\mathcal{X},Q)) = (\{y_4,y_5\},Q)$ têm o mesmo *cluster*. Na verdade, só é alterada a ordem das variáveis, pois com uma reordenação das variáveis e dos vértices do *quiver* obtemos sementes "idênticas".

A álgebra *cluster* $\mathcal{A}(Q)$ do Exemplo 3.11 tem 10 sementes (salvo reordenações dos vértices e das variáveis *cluster*), 5 *clusters* e 5 variáveis *cluster*.

Como veremos na próxima proposição, ver [5], a quantidade de variáveis *cluster* associadas a um *quiver* de tipo Dynkin é relacionada com a quantidade de módulos indecomponíveis da álgebra de caminhos associados a este *quiver*. Para mais detalhes sobre estes temas, sugerimos [1].

Proposição 3.16 *Seja $\mathcal{A}(Q)$ uma álgebra cluster de tipo finito associada a um quiver Q de tipo Dynkin. A quantidade de variáveis cluster é igual à quantidade de módulos indecomponíveis em* modkQ *mais n, ou seja:*

(a) para $\mathbb{A}_n$, há $\frac{n\cdot(n+3)}{2}$ variáveis cluster;
(b) para $\mathbb{D}_n$, há n^2 variáveis cluster;
(c) para $\mathbb{E}_6$, há 42 variáveis cluster;
(d) para $\mathbb{E}_7$, há 70 variáveis cluster; e
(e) para $\mathbb{E}_8$, há 128 variáveis cluster.

3.5 O grafo das mudanças: o exemplo de $\mathbb{A}_3$

Quando a álgebra *cluster* é de tipo finito, podemos visualizar o grafo das mudanças dos *clusters*. Este grafo é da seguinte forma:

- os vértices são os *clusters*; e
- existe uma aresta entre dois *clusters* se há uma mutação entre as sementes associadas a eles.

O grafo das mudanças é um grafo n-regular, ou seja, cada vértice é incidente a n arestas.

Considere o *quiver* Q : $\textcircled{1}\to\textcircled{2}\to\textcircled{3}$. A álgebra *cluster* $\mathcal{A}(Q)$ possui 9 variáveis *cluster* enumeradas a seguir:

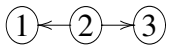
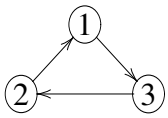
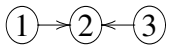
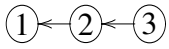
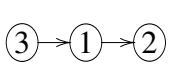
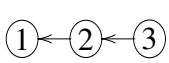
$$y_1 = x_1, \qquad y_2 = x_2, \qquad y_3 = x_3,$$

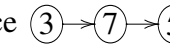
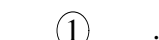
$$y_4 = \frac{1+x_2}{x_1}, \qquad y_5 = \frac{x_1+x_3}{x_2}, \qquad y_6 = \frac{1+x_2}{x_3},$$

$$y_7 = \frac{x_1+x_3+x_2\cdot x_3}{x_1\cdot x_2}, \quad y_8 = \frac{x_1+x_3+x_1\cdot x_2}{x_2\cdot x_3} \quad \text{e} \quad y_9 = \frac{x_1+x_3+x_1\cdot x_2+x_2\cdot x_3}{x_1\cdot x_2\cdot x_3}.$$

Como foi visto no último exemplo da seção anterior, podemos ter diferentes sementes associadas ao mesmo *cluster*. Este fato será ilustrado com a seguinte tabela com algumas sementes obtidas por mutações a partir da semente inicial $(\mathcal{X}, Q)$:

Tabela 3.1: *Diferentes sementes associadas ao mesmo cluster.*

semente	*quiver*	nova variável	*cluster*
$\mu_1(\mathcal{X},Q)$		$y_4 = \frac{1+x_2}{x_1}$	$\{y_4, y_2, y_3\}$
$\mu_2(\mathcal{X},Q)$		$y_5 = \frac{x_1+x_3}{x_2}$	$\{y_1, y_5, y_3\}$
$\mu_3(\mathcal{X},Q)$		$y_6 = \frac{1+x_2}{x_3}$	$\{y_1, y_2, y_6\}$
$\mu_2(\mu_1(\mathcal{X},Q))$		$y_7 = \frac{x_1+x_3+x_2\cdot x_3}{x_1\cdot x_2}$	$\{y_4, y_7, y_3\}$
$\mu_3(\mu_1(\mathcal{X},Q))$ $= \mu_1(\mu_3(X,Q))$		$y_6 = \frac{1+x_2}{x_3}$	$\{y_4, y_2, y_6\}$
$\mu_1(\mu_2(\mathcal{X},Q))$		$y_7 = \frac{x_1+x_3+x_2\cdot x_3}{x_1\cdot x_2}$	$\{y_7, y_5, y_3\}$
$\mu_1(\mu_2(\mu_1(\mathcal{X},Q)))$		$y_5 = \frac{x_1+x_3}{x_2}$	$\{y_5, y_7, y_3\}$

Porém, estas sementes são "idênticas" se reordenamos os vértices e as variáveis *cluster*. Por isso, optamos por representar os vértices do *quiver* de mudança com o *quiver* da semente usando os índices das variáveis *cluster* como vértices para explicitar as regras de mudanças usadas. Por exemplo, o vértice  representa tanto a semente $\mu_1(\mu_2(\mathcal{X},Q))$ quanto a semente $\mu_1(\mu_2(\mu_1(\mathcal{X},Q)))$. A notação  também será utilizada para designar o ciclo orientado .

A álgebra *cluster* $\mathcal{A}(Q)$ possui 14 *clusters* representados através do grafo das mudanças mostrado na Figura 3.1.

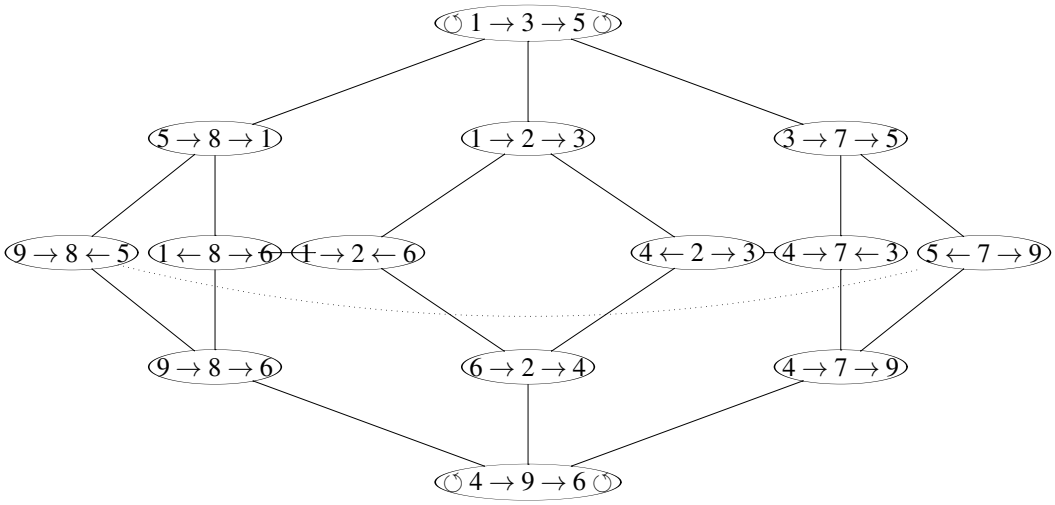


Figura 3.1: *Grafo das mudanças da álgebra $\mathcal{A}(Q)$.*

3.6 Problema de reconhecimento

O seguinte critério para decidir se uma matriz antissimetrizável corresponde a uma álgebra *cluster* de tipo finito é apresentada em [2].

Teorema 3.17 (Barot, Geiss e Zelevinsky [2]) *Seja uma álgebra cluster $\mathcal{A}(B)$ associada a uma matriz antissimetrizável B. Seja $\mathcal{S}$ a classe de equivalência das mutações. As seguintes afirmações são equivalentes:*

(a) a álgebra cluster $\mathcal{A}(B)$ é de tipo finito;
(b) $\mathcal{S}$ contém uma matriz B' tal que a companheira quase-Cartan C, com entradas fora da diagonal $c_{ij} = -|b'_{ij}|$, é positiva;
(c) para todo $B' \in \mathcal{S}$ e todo $i \neq j$, temos que $|b'_{ij} \cdot b'_{ji}| \leq 3$; e
(d) todo ciclo sem corda em $G(B)$ é ciclicamente orientado e B tem uma companheira quase-Cartan positiva.

Além disso, o tipo de Cartan-Killing da matriz C definida em (b), que é de Cartan, é unicamente determinada por $\mathcal{S}$, ou seja, B.

Geralmente, as condições (b) e (c) do Teorema 3.17 são muito difíceis de verificar, visto que pode existir uma quantidade muito grande (possivelmente infinita) de matrizes em $\mathcal{S}$. Por outro lado, a condição (d) nos leva a um algoritmo polinomial como será visto no Capítulo 6.

Sendo assim, consideramos o problema de reconhecimento seguinte.

Problema de reconhecimento: determinar se a álgebra *cluster* associada à matrizes antissimetrizáveis é de tipo finito.
Entrada: uma matriz antissimetrizável B.
Saída: decisão se a álgebra *cluster* $\mathcal{A}(B)$ é de tipo finito ou não.

Barot, Geiss e Zelevinsky [2] propuseram uma solução apresentada no critério (d) baseada na extensão do critério (b) a toda classe representativa de mutação em questão. Se o grafo $G(B)$ não é ciclicamente orientável, então a álgebra *cluster* associada à classe de equivalência de mutações de B não é de tipo finito.

Outra solução interessante foi obtida por Seven [44], em que verifica se a álgebra *cluster* é de tipo finito em termos de "menores proibidos" de B.

Tendo em vista o que foi discutido neste capítulo, para decidir se uma álgebra *cluster* é de tipo finito podemos dividir este problema em duas partes:

(a) verificar se um grafo $G(B)$ é ciclicamente orientado (veja Capítulo 5); e
(b) decidir se uma matriz antissimetrizável B tem uma companheira quase-Cartan positiva (veja Capítulo 6).

Relacionado ao item (a), temos o problema de encontrar todos os ciclos sem corda de um grafo qualquer. Estudaremos este problema no capítulo a seguir.

A proposição seguinte enfatiza os conceitos que foram vistos na Seção 3.3.

Proposição 3.18 (Barot, Geiss e Zelevinsky [2]) *Para uma matriz antissimetrizável B, o grafo $G' = G(\mu_k(B))$ é unicamente determinado pelo grafo $G = G(B)$ e um índice de matriz k. Especificamente, G' é obtido de G como segue:*

(a) A orientação de todas as arestas incidentes a k são revertidas, seu pesos continuam intactos.
(b) Para quaisquer vértices i e j que são conectados em G via um caminho com arestas duplas orientado através de k, a direção da aresta (i, j) em G' e seu peso c' são unicamente determinados pela regra:

$$\pm\sqrt{c}\pm\sqrt{c'}=\sqrt{ab}$$

onde o sinal antes de $\sqrt{c}$ (respectivamente, antes de $\sqrt{c'}$) é "+" se i, j, k formam um ciclo orientado em Q (respectivamente, em Q') e é "-" caso contrário. Aqui, ambos c e c' podem ser iguais a 0.
(c) O restante das arestas e seus pesos em Q permanecem inalterados.

Corolário 3.19 (Barot, Geiss e Zelevinsky [2]) *Suponha que uma matriz antissimetrizável B satisfaz a condição (d) no Teorema 3.17 e seja C uma companheira quase-Cartan positiva de B. Então, $\mu_k(B)$ tem uma companheira quase-Cartan positiva para qualquer índice de matriz k.*

Uma companheira quase-Cartan de B é especificada pela escolha de sinais fora das entradas da diagonal da matriz, com somente a exigência de que $sgn(c_{ij}) = sgn(c_{ji})$,

para $i \neq j$, como vimos no Teorema 2.16. Logo, o número de escolhas para C é 2^m, onde m é o número de arestas em $G(B)$, ou seja, é o número de entradas diferentes de zero acima da diagonal principal na matriz.

Exemplo 3.20 *(a) Seja* $B = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$ *uma matriz antissimetrizável.* $G(B)$ *tem um ciclo minimal que é ciclicamente orientado e* $C = \begin{pmatrix} 2 & -1 & 1 \\ -1 & 2 & -1 \\ 1 & -1 & 2 \end{pmatrix}$ *é uma companheira quase-Cartan positiva de B. Portanto, a álgebra cluster associada à classe de equivalência de B é de tipo finito.*

(b) Seja $B = \begin{pmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{pmatrix}$ *uma matriz antissimetrizável.* $G(B)$ *tem um ciclo minimal que não é ciclicamente orientado e a matriz C do item anterior é uma companheira quase-Cartan positiva de B. Portanto, a álgebra cluster associada à classe de equivalência de mutações de B não é de tipo finito.*

Observação 3.21 *A* ***forma quadrática*** *associada a uma matriz simétrica C é dada por* $q(x,y) = x \cdot C \cdot y^T$. *Se C é uma companheira de quase-Cartan de B, então o tipo de Cartan-Killing da álgebra cluster associada a B é o mesmo tipo de Cartan-Killing da forma quadrática representada pela matriz simetrizada* $D \times C$.

CAPÍTULO 4

Ciclos sem Corda

Do estudo do artigo de Barot, Geiss e Zelevinsky [2], surgiu o interesse pelo estudo da enumeração dos ciclos sem corda.

Como veremos no próximo capítulo, Gurvich [26] e Speyer [48] apresentam alguns resultados que proveem uma maneira mais eficiente de verificar se um grafo é ciclicamente orientável sem aparentemente ter que listar todos os ciclos sem corda, embora o faça para grafos ciclicamente orientáveis, como veremos no Teorema 5.12. Relembre que um grafo é ciclicamente orientável se todos os ciclos sem corda são ciclicamente orientados, ou seja, possuem uma orientação cíclica.

Um *buraco* é um ciclo sem corda com cinco ou mais vértices. O *grafo antiburaco* é o complemento do grafo buraco. Buracos e antiburacos têm sido extensivamente estudados em diferentes contextos algorítmicos da Teoria dos Grafos. Os exemplos mais notáveis são de grafos fracamente cordais, também conhecidos como grafos fracamente triangulados [23, 29], que são aqueles que não contêm buracos nem antiburacos.

Nikolopoulos e Palios [40] apresentam 2 algoritmos, um para detecção de buracos e outro para detecção de antiburacos em grafos arbitrários. Ambos algoritmos têm complexidade de tempo $O(n+m^2)$ e de espaço $O(n \cdot m)$. A existência de tais ciclos é verificada por um tipo de busca em profundidade traversal, que eles chamam de P_4-DFS. Spinrad [49] formula uma solução melhorada com tempo $O(n^{k-3} \cdot M)$, onde $M \equiv n^{2.376}$ é o tempo requerido para a multiplicação de duas matrizes de ordem n e k é um inteiro positivo.

Uma solução de complexidade de tempo $O(n^k)$ para o problema de determinar se um grafo contém ou não um ciclo sem corda com $k \geq 4$ vértices, para algum valor fixo de k é proposta por Hayward [30]. Golumbic [23] propõe um algoritmo de complexidade $O(n+m)$ para reconhecer grafos cordais, isto é, grafos que não possuem ciclos sem cordas. O caso em que $k \geq 5$ vértices é resolvido por Nikolopoulos e Palios [40]. Entretanto, encontrar qualquer ciclo sem corda com um tamanho k dado é obviamente mais fácil do que encontrar todos os ciclos sem corda em um grafo G.

Enumeração é uma tarefa fundamental em Ciência da Computação e alguns algoritmos são propostos para enumerar estruturas tais como ciclos [15, 3, 37, 36, 42,

43, 57], circuitos [4, 53], caminhos [27, 42], árvores [34, 42] e cliques [38, 55]. Devido à quantidade de ciclos - que pode ser exponencialmente grande - estes tipos de tarefas são normalmente difíceis de controlar, pois mesmo pequenos grafos podem conter um número grande de ciclos. Entretanto, a enumeração é necessária em muitos problemas práticos. Pfaltz [41] mostra que ciclos sem corda efetivamente caracterizam estruturas de conectividade de redes como um todo. Por exemplo, a enumeração de ciclos é usada para a análise da WWW (*World Wide Web*) e redes sociais, onde ciclos podem ser úteis para identificar padrões de conectividade na rede.

Ciclos sem corda são usados para o melhor entendimento de estruturas de redes ecológicas, tais como cadeias alimentares (*food webs*), onde o objetivo é descobrir quais predadores competem pela mesma presa [47]. Para alcançar este objetivo, um grafo direcionado de uma cadeia alimentar é transformado em um grafo de competição (*niche overlap*) para realçar a competição entre as espécies. A falta de ciclos sem corda no último grafo significa que as espécies podem ser rearranjadas com uma hierarquia simples. Para maiores detalhes, veja a Seção 2.1. Outra aplicação é na natureza de relações estrutura-propriedade em alguns componentes químicos que são relacionados à presença de ciclos sem corda [22].

Wild [57] propõe um algoritmo para listar todos os subjconjuntos de ciclos de cardinalidade maior ou igual a cinco, usando o princípio da exclusão. O algoritmo pode ser facilmente modificado para listar somente os ciclos sem corda. Infelizmente, o autor não apresenta a análise de complexidade do algoritmo mas é fácil ver que ele tem uma complexidade assintótica grande.

Um algoritmo para enumerar todos os ciclos sem corda, com complexidade de tempo $O(n+m)$ no tamanho da saída, é proposto por Uno e Satoh [56], mas cada ciclo sem corda aparece mais de uma vez na saída. Na verdade, cada ciclo aparece a quantidade de vezes igual ao seu número de arestas. Então, o algoritmo tem complexidade de tempo $O(n \cdot (n+m) \cdot tam(Csc))$, onde $tam(Csc)$ é a soma de comprimentos de todos os ciclos sem corda no grafo.

Nós elaboramos um algoritmo, e uma extensão do mesmo com BFS, para enumerar todos os ciclos sem corda de um grafo dado G, com a vantagem de encontrar cada ciclo sem corda apenas uma vez. A extensão com BFS tem complexidade de tempo $O((n+m) \cdot tam(Csc))$, ou seja, $O(n+m)$ no tamanho da saída. A extensão com BFS garante que cada busca encontra pelo menos um ciclo sem corda. O algoritmo sem a extensão com BFS tem complexidade de tempo $O((n+m) \cdot tam(Psc))$, onde $tam(Psc)$ é a soma de comprimentos de todos os caminhos sem corda no grafo. De fato, nesta soma são considerados apenas alguns caminhos sem corda específicos. Porém, como veremos na Seção 4.4, o tempo de execução é melhor do que o algoritmo com extensão.

A ideia principal dos nossos algoritmos é usar um esquema de rotulação de

vértices, com o qual qualquer ciclo arbitrário pode ser descrito de uma única maneira. A partir disto, nós geramos um conjunto inicial de triplas de vértices e usamos uma estratégia de DFS para encontrar todos os ciclos sem corda. Nossa abordagem garante que cada ciclo sem corda é encontrado apenas uma vez. Nós gostaríamos de esclarecer que nosso método, apesar de baseado em ideias similares às apresentadas por Sokhn et al. [47], foi desenvolvido independentemente e tem um resultado significantemente mais rápido. A abordagem de Sokhn et al. [47] tem a desvantagem de encontrar e listar cada ciclo duas vezes.

O capítulo está dividido da seguinte forma: na Seção 4.1 apresentamos mais algumas notações e resultados importantes para o entendimento dos algoritmos; na Seção 4.2 apresentamos o algoritmo principal para listar todos os ciclos sem corda de um grafo e uma variante utilizando o BFS, que garante uma busca bem sucedida; na Seção 4.3 mostramos a corretude e a complexidade algorítmica do algoritmo e sua extensão e também propomos melhorias futuras; e, por fim, apresentamos os resultados experimentais na Seção 4.4.

4.1 Ciclos vistos de forma única

Seja um grafo G, com o conjunto de arestas $E(G)$ e o conjunto de vértices $V(G)$. Denotamos por $Adj(x)$ a *vizinhança aberta*, que é o conjunto de vizinhos do vértice $x \in V(G)$, ou seja, $Adj(x) = \{y \in V(G) \mid (x,y) \in E(G)\}$ e por $Adj[x] = \{x\} \cup Adj(x)$ a *vizinhança fechada* do vértice x.

O *grau mínimo* entre todos os vértices de G é denotado por $\delta(G)$; o *grau máximo* é denotado por $\Delta(G)$; e $grau_G(v)$ denota o grau de um vértice particular $v \in V(G)$. Denotamos por $G - X$ $(G - u)$ o subgrafo induzido pelo subconjunto $V(G) - X$, para $X \subseteq V(G)$ $(V(G) - \{u\}$, para $u \in V(G))$. Similar a Chandrasekharan, Laskshmanan e Medidi [7], damos uma caracterização de grafos com ciclos sem corda de tamanho $k \geq 4$.

Lema 4.1 *Dado $t \geq 3$, um grafo G tem um ciclo sem corda C_k, com $k \geq t$, se e somente se existe um caminho sem corda $\langle u_1, u_2, \ldots, u_t \rangle$ tal que u_1 e u_t estão no mesmo componente conexo de*

$$G' = G - \left(\bigcup_{i=2}^{t-1} Adj[u_i] - \{u_1, u_t\} \right). \tag{4-1}$$

Prova. ($\Rightarrow$) Suponha que existe um ciclo sem corda $C_k = \langle u_1, u_2, \ldots, u_k \rangle$ em G tal que $k \geq t$. Consideramos o caminho sem corda $p = \langle u_1, u_2, \ldots, u_t \rangle$. Sendo $G' = G - \left(\bigcup_{i=2}^{t-1} Adj[u_i] - \{u_1, u_t\} \right)$, temos que $\langle u_t, u_{t+1}, \ldots, u_k, u_1 \rangle$ é um caminho de u_t a u_1 em G'. Logo, u_1 e u_t estão no mesmo componente conexo de G'.

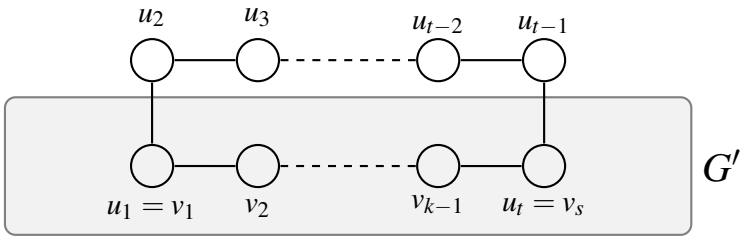


Figura 4.1: *Condição de suficiência para a existência de um ciclo sem corda.*

($\Leftarrow$) Suponha que existe um caminho sem corda $p = \langle u_1, u_2, \ldots, u_t \rangle$ tal que u_1 e u_t estão no mesmo componente conexo de G'. Seja $q = \langle v_1, v_2, \ldots, v_s \rangle$ um caminho mais curto em G' tal que $v_1 = u_1$ e $v_s = u_t$. Esta situação é mostrada na Figura 4.1.

Suponha, por contradição, que $p \cup q = \langle v_1 = u_1, u_2, \ldots, u_t = v_k, v_{k-1}, \ldots, v_2 \rangle$ não é um ciclo sem corda. Então, existem i, j satisfazendo um dos três seguintes casos:

(a) $1 \leq i < j \leq t$, com $j \neq i+1$ e $(u_i, u_j) \in E(G)$. Isto contradiz o fato que p é um caminho sem corda.

(b) $1 \leq i < j \leq s$, com $j \neq i+1$ e $(v_i, v_j) \in E(G)$. Logo, $\langle v_1, \ldots, v_i, v_j, \ldots, v_s \rangle$ é um caminho de v_1 a v_s em G'. Como q é um caminho mais curto de v_1 a v_s em G', temos a contradição desejada.

(c) $1 < i < t$ e $1 < j < s$, com $(u_i, v_j) \in E(G)$, é uma contradição, pois $v_j \in G'$ e logo $v_j \notin Adj(u_i)$.

Portanto, $p \cup q$ é um ciclo sem corda de tamanho $t + s - 2$. Visto que $s \geq 2$, temos que $k = t + s - 2 \geq t$. □

Uma ordenação de vértices de G pode ser definida por uma bijeção $\ell : V(G) \to \{1, 2, \ldots, n\}$. Nós chamamos tal bijeção de *rotulação de vértices*. O lema abaixo mostra que qualquer rotulação de vértices possibilita um ciclo ser definido de maneira única.

Lema 4.2 *Sejam G um grafo não orientado e $\ell : V(G) \to \{1, 2, \ldots, n\}$ uma rotulação de vértices. Se G contém um ciclo simples c de tamanho k, então ℓ define este ciclo de uma maneira única da seguinte forma: $c = \langle v_1, v_2, \ldots, v_k \rangle$, com $\ell(v_2) = \min\{\ell(v_i) \mid i = 1, \ldots, k\}$ e $\ell(v_1) < \ell(v_3)$.*

Prova. Qualquer ciclo $\langle v_1, v_2, \ldots, v_k \rangle$ pode ser descrito de $2 \cdot k$ formas possíveis, sendo $\langle v_{i-1}, v_i, v_{i+1}, \ldots, v_k, v_1, v_2, \ldots, v_{i-2} \rangle$ ou $\langle v_{i+1}, v_i, v_{i-1}, \ldots, v_2, v_1, v_k, \ldots, v_{i+2} \rangle$, para $i = 1, \ldots, k$. Seja i o índice de vértice tal que $\ell(v_i) = \min\{\ell(v_j) \mid j = 1, \ldots, k\}$. Existem somente duas possibilidades para o vértice v_i ser o segundo do ciclo: $\langle v_{i-1}, v_i, v_{i+1}, \ldots, v_k, v_1, v_2, \ldots, v_{i-2} \rangle$ ou $\langle v_{i+1}, v_i, v_{i-1}, \ldots, v_2, v_1, v_k, \ldots, v_{i+2} \rangle$. Visto que

os vizinhos de v_i no ciclo são v_{i-1} e v_{i+1}, exatamente uma destas possibilidades satisfaz a condição $\ell(v_{i-1}) < \ell(v_{i+1})$, como ilustrado na Figura 4.2. □

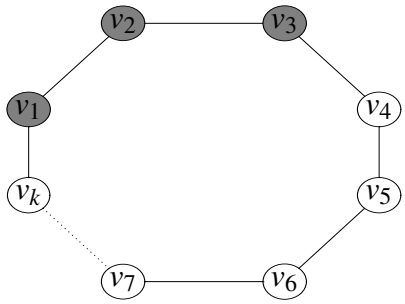


Figura 4.2: *Uma única maneira de ver o ciclo sem corda.*

Definição 4.3 *Uma* **tripla** *é uma sequência de vértices* $\langle x,u,y\rangle$ *tal que* $x,y \in Adj(u)$, $\ell(u) < \ell(x) < \ell(y)$ *e* $(x,y) \notin E(G)$.

As triplas são as sequências de três vértices que podem iniciar um possível ciclo sem corda de tamanho maior que três. Denotamos por $T(G)$ o conjunto das triplas de G.

Observação 4.4 $C_3 = \{\langle x,u,y\rangle \mid x,u,y \in V(G)$ *com* $x,y \in Adj(u)$, $\ell(u) < \ell(x) < \ell(y)$ *e* $(x,y) \in E(G)\}$ *é o conjunto de todos os ciclos de tamanho 3.*

O Algoritmo 4.1 gera todas as triplas válidas, na forma $\langle x,u,y\rangle$, de acordo com a Definição 4.3. Aquelas que formam um ciclo de tamanho 3 são armazenadas no conjunto C_3 e as demais, triplas expansíveis, são guardadas em $T(G)$. Este algoritmo tem complexidade de tempo $O(n \cdot \Delta^2)$, ou seja, $O(n^3)$.

Algoritmo 4.1: $Triplas(G,\ell)$

Entrada: Um grafo simples não orientado G e uma rotulação de vértices ℓ.
Saída: O conjunto $T(G)$ de triplas e o conjunto C_3 de ciclos de tamanho 3.

```
1 T(G) ← ∅
2 C3 ← ∅
3 para cada u ∈ V(G) faça /* Gera todas triplas na forma ⟨x,u,y⟩. */
4     para cada x,y ∈ Adj(u) tal que ℓ(u) < ℓ(x) < ℓ(y) faça
5         se ((x,y) ∉ E(G)) então
6             T(G) ← T(G) ∪ {⟨x,u,y⟩}
7         senão
8             C3 ← C3 ∪ {⟨x,u,y⟩}
9 retorna T(G), C3
```

A rotulação de vértices que usamos, chamada *rotulação de grau*, é construída sobre uma sequência de subgrafos de G. Começamos com $G_1 = G$. Para $i \geq 1$, o $(i+1)$-ésimo subgrafo é definido como $G_{i+1} = G_i - u_i$, para um dado $u_i \in V(G_i)$ tal que $grau_{G_i}(u_i) = \delta(G_i)$. Dada tal sequência, definimos a rotulação de grau como $\ell(u_i) = i$ para cada i. Se G é uma árvore, então não existe nenhuma tripla possível, isto é, $T(G) = \varnothing$ e se G é um ciclo, então $|T(G)| = 1$. Sendo assim, triplas desnecessárias são descartadas neste casos, não importa qual rotulação de grau é utilizada.

O Algoritmo 4.2 calcula os graus dos vértices e atribui uma rotulação de grau ao grafo G. Ele tem complexidade de tempo $\Theta(n^2)$.

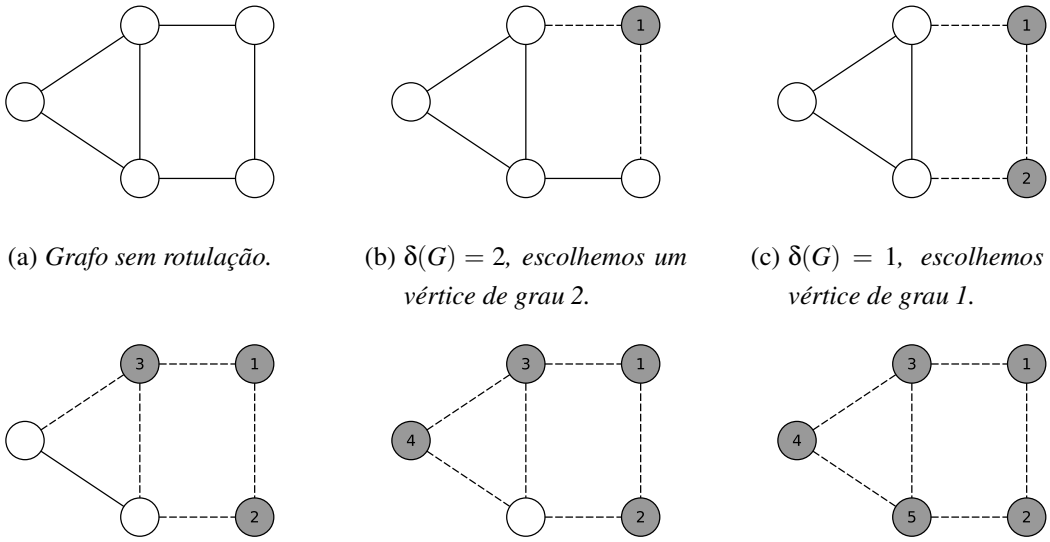
Algoritmo 4.2: $RotulacaoGrau(G)$

Entrada: Um grafo simples não orientado G.

Saída: Uma rotulação de grau de G.

```
1  para cada v ∈ V(G) faça
2      grau(v) ← 0
3      cor(v) ← BRANCA
4      para cada u ∈ Adj(v) faça
5          grau(v) ← grau(v) + 1
6  para i = 1 até n faça
7      min_grau ← n
8      para cada x ∈ V(G) faça
9          se ((cor(x) = BRANCA) e (grau(x) < min_grau)) então
10             v ← x
11             min_grau ← grau(x)
12     ℓ(v) ← i
13     cor(v) ← PRETA
14     para cada u ∈ Adj(v) faça
15         se (cor(u) = BRANCA) então
16             grau(u) ← grau(u) − 1
17 retorna ℓ
```

A Figura 4.3 mostra a rotulação de grau passo a passo em um grafo com 5 vértices.



(a) *Grafo sem rotulação.*

(b) $\delta(G) = 2$, *escolhemos um vértice de grau 2.*

(c) $\delta(G) = 1$, *escolhemos o vértice de grau 1.*

(d) $\delta(G) = 2$, *escolhemos um vértice de grau 2.*

(e) $\delta(G) = 1$, *escolhemos o vértice de grau 1.*

(f) *Escolhemos o vértice restante.*

Figura 4.3: *Exemplo de rotulação de grau passo a passo.*

Exemplo 4.5 *No grafo da Figura 4.4,* $T(G) = \{\langle 7,1,10\rangle, \langle 7,2,13\rangle, \langle 10,3,13\rangle\}$ *e temos um exemplo de uma rotulação em que existem buscas infrutíferas (nas duas últimas triplas). Como veremos, o uso do BFS garante que cada busca encontre um ciclo sem corda.*

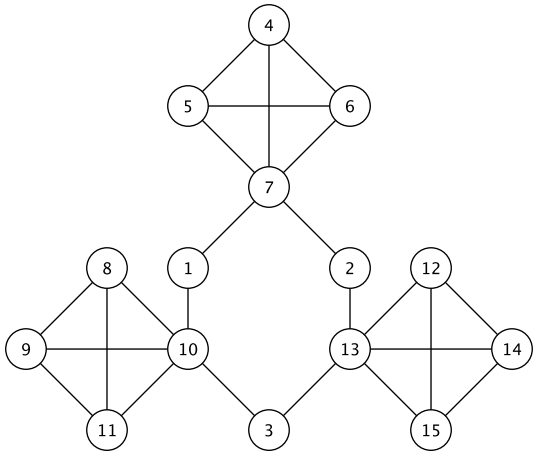


Figura 4.4: *Exemplo de rotulação de grau.*

O Lema 4.6 usa a rotulação de grau anteriormente definida para estabelecer um limite superior para a cardinalidade do conjunto $T(G)$.

Lema 4.6 *Seja G um grafo e* ℓ *uma rotulação de grau. Então,*

$$|T(G)| \leq \frac{(\Delta(G)-1)\cdot m}{2}. \tag{4-2}$$

Prova. Provaremos o resultado por indução em n. Claramente, se $n = 1$ o resultado é verdadeiro, pois $\delta(G) = \Delta(G) = 0$, $m = 0$ e $T(G) = 0$. Suponha que $n > 1$. Sejam $u_1 \in V(G)$, com $grau_G(u_1) = \delta(G)$, e $G_2 = G - u_1$. Lembramos que $m_2 = |E(G_2)|$ e $m_1 = m$. Não é difícil ver que $T(G) = T(G_2) \cup \{\langle x, u_1, y\rangle$ tal que $x, y \in Adj(u_1)$ com $\ell(x) < \ell(y)$ e $(x,y) \notin E(G)\}$.

Por indução, temos que $|T(G_2)| \leq \frac{(\Delta(G_2)-1)\cdot m_2}{2}$. Visto que $\Delta(G_2) \leq \Delta(G)$ e $m_2 = m - \delta(G)$, temos que

$$|T(G_2)| \leq \frac{(\Delta(G)-1)\cdot(m-\delta(G))}{2}$$

e

$$\begin{aligned} |T(G)| &\leq \frac{(\Delta(G)-1)\cdot(m-\delta(G))}{2} + \frac{\delta(G)\cdot(\delta(G)-1)}{2} \\ &\leq \frac{(\Delta(G)-1)\cdot(m-\delta(G))}{2} + \frac{\delta(G)\cdot(\Delta(G)-1)}{2} \\ &\leq \frac{(\Delta(G)-1)\cdot(m-\delta(G)+\delta(G))}{2} = \frac{(\Delta(G)-1)\cdot m}{2}. \end{aligned}$$

Portanto, $|T(G)| \leq \frac{(\Delta(G)-1)\cdot m}{2}$ e a prova está completa. □

Este limite é justo para árvores, mas para ciclos C_k a diferença é grande.

O lema abaixo estabelece as possíveis propriedades de um vizinho do último vértice de um caminho sem corda.

Lema 4.7 *Seja $p = \langle u_1, u_2, \ldots, u_t\rangle$ um caminho sem corda e $v \in Adj(u_t)$, $v \neq u_{t-1}$. Exatamente um dos seguintes ocorre:*

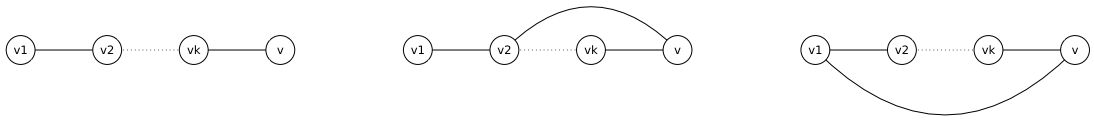
(a) $\langle p, v\rangle = \langle u_1, u_2, \ldots, u_t, v\rangle$ é um caminho sem corda;
(b) existe $i \in \{1, \ldots, t-1\}$ tal que $\langle u_i, u_{i+1}, \ldots, u_t, v\rangle$ é um ciclo sem corda.

Prova. Visto que $v \in Adj(u_t)$, $v \neq u_{t-1}$ e p é um caminho sem corda, então $v \neq u_i$ para todo $i = 1, \ldots, t$, ou seja, $\langle p, v\rangle$ é um caminho simples.

(a) Se $(u_i, v) \notin E(G)$ para todo $i \in \{1, \ldots, t-1\}$, então $\langle p, v\rangle$ é um caminho sem corda.

(b) Suponha que existe um índice $i \in \{1, \ldots, t-1\}$ tal que $(v, u_i) \in E(G)$. Escolhendo o maior índice i com esta propriedade, temos o ciclo sem corda desejado. □

No caso (a), ilustrado na Figura 4.5(a), o caminho $\langle p, v\rangle$ pode ser parte de um ciclo sem corda. O caso (b) (Figura 4.5(b)), com $i \neq 1$, estabelece que o caminho $\langle p, v\rangle$ tem uma corda e, logo, não pode ser estendido a um ciclo sem corda. No caso (b), com $i = 1$, temos que $\langle p, v\rangle$ é um ciclo sem corda, conforme ilustrado na Figura 4.5(c).



(a) *Um exemplo de caminho sem corda.*

(b) *Um exemplo de caminho com corda.*

(c) *Um exemplo de um ciclo sem corda.*

Figura 4.5: *Condições do Lema 4.7.*

4.2 Algoritmos para encontrar todos os ciclos sem corda

O princípio geral dos algoritmos propostos é limitar o espaço de busca através da criação de um conjunto inicial de triplas de vértices $T(G)$ e usar uma estratégia de DFS para identificar caminhos sem corda a partir de cada tripla em $T(G)$. Em cada passo nossa estratégia emprega algumas técnicas de otimização para reduzir o número de testes requeridos antes de decidir o que fazer com o caminho expandido atual. Entre estas técnicas, as mais relevantes são o bloqueio e a rotulação. A primeira é uma adaptação de uma técnica originalmente apresentada por Tiernan [54] e subsequentemente usada por Tarjan [52], Johnson [31] e Read e Tarjan [42]. Os algoritmos propostos são apresentados a seguir.

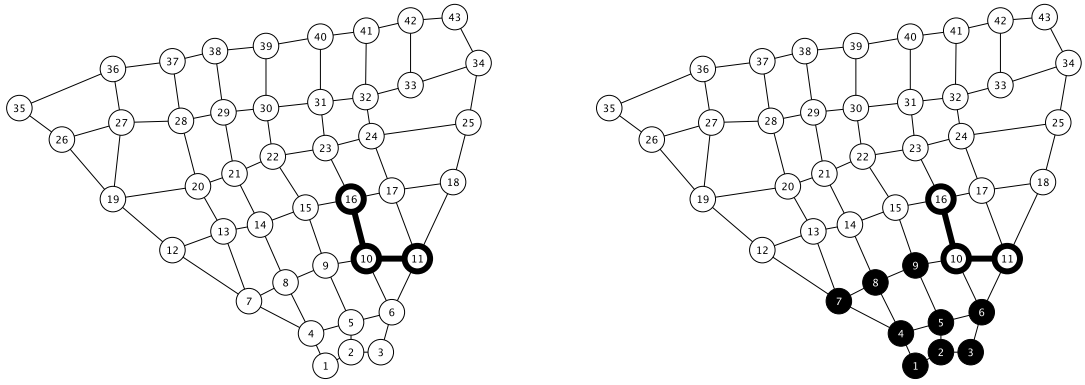
Para simplificar a explicação, o Algoritmo 4.3 ($CiclosSemCorda(G)$) é dividido em cinco partes. Primeiramente, ele chama o Algoritmo 4.2 ($RotulacaoGrau(G)$) para gerar a rotulação de grau de G, que permite que seja minimizada a cardinalidade do conjunto de triplas $T(G)$, computada pelo Algoritmo 4.1 ($Triplas(G,\ell)$). Este último algoritmo também calcula os ciclos sem corda de tamanho 3, que são armazenados no conjunto *Csc*. Depois, o contador *nr_bloqueios*, que será explicado a seguir, é inicializada com 0 para todos os vértices do grafo. Em seguida, o Algoritmo 4.6 (*CC_Visit*) processa cada tripla em busca de ciclos sem corda.

Algoritmo 4.3: *CiclosSemCorda(G)*

Entrada: Um grafo simples não orientado G.

Saída: O conjunto Csc de todos os ciclos sem corda de G.

```
1  ℓ ← RotulacaoGrau(G)
2  (T(G), C3) ← Triplas(G, ℓ)
3  Csc ← C3
4  para cada u ∈ V(G) faça
5      nr_bloqueios(u) ← 0
6  para cada p = ⟨x, u, y⟩ ∈ T(G) faça
7      BloqueiaVizinhos(u)
8      Csc ← CC-Visit(p, Csc, ℓ(u), bloqueado)
9      DesbloqueiaVizinhos(u)
10 retorna Csc
```



(a) *Exemplo de uma tripla inicial.*

(b) *Os vértices v coloridos são descartados, pois* $\ell(v) < \ell(u)$.

Figura 4.6: *Exemplos de vértices descartados.*

No Algoritmo *CC_Visit*, na expansão de caminhos, os vértices v com $\ell(v) > \ell(u)$ não são considerados durante a expansão, conforme o Lema 4.2. Um exemplo pode ser visto na Figura 4.6.

Usamos o conceito de bloqueio de vértices para facilitar a localização de um vértice v adjacente a um caminho sem corda que o estenda ou gere um ciclo sem corda, conforme o Lema 4.7. A estratégia original de Tiernan [54], que consiste no simples bloqueio e desbloqueio de vértices, não é suficiente para satisfazer as necessidades dos algoritmos propostos, porque cada vértice no caminho sem corda pode ser vizinho de vários outros. Assim, expandimos o conceito e usamos um contador que indica o número de vezes que um vértice é encontrado como vizinho de outro em um caminho sem corda. A ideia é que o vértice atual pode ser considerado desbloqueado apenas quando todos os seus vizinhos já tenham sido processados.

Para cada $v \in V(G)$, $nr\_bloqueios(v)$ denota o número de vizinhos de v em um caminho sem corda (sem o primeiro e o último vértice). Um vértice v é dito ser *desbloqueado* se $nr\_bloqueios(v) = 0$ e *bloqueado*, caso contrário. Os vértices cinzas na Figura 4.7 indicam os vértices bloqueados, enquanto que os pretos são os descartados. Observe que $nr\_bloqueios(17) = 2$. No início do processamento de uma tripla, com exceção do seu primeiro vértice, os vizinhos dos outros dois vértices são marcados como bloqueados e já não podem ser usados no caminho. O objetivo dessa abordagem é bloquear os vizinhos que poderiam formar uma corda com vértices no caminho sem corda. Esta estratégia permite a extensão do caminho de forma mais rápida. Após a conclusão do processamento de uma tripla, todos estes vizinhos são marcados como desbloqueados. As operações de bloqueio e desbloqueio são detalhadas nos Algoritmos 4.4 (*BloqueiaVizinhos(v)*) e 4.5 (*DesbloqueiaVizinhos(v)*).

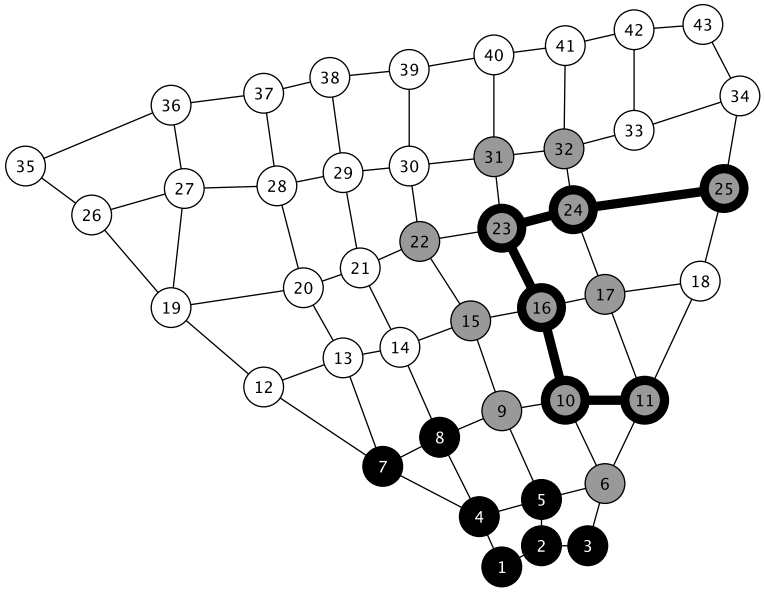


Figura 4.7: *Exemplo de bloqueio do caminho* $\langle 11, 10, 16, 23, 24 \rangle$.

Algoritmo 4.4: $BloqueiaVizinhos(v, nr\_bloqueios)$

Entrada: Um vértice $v \in V(G)$ e um vetor global $nr\_bloqueios$.

1 **para cada** $u \in Adj(v)$ **faça**
2 $\quad nr\_bloqueios(u) \leftarrow nr\_bloqueios(u) + 1$.

Algoritmo 4.5: *DesbloqueiaVizinhos*$(v, nr\_bloqueios)$

Entrada: Um vértice $v \in V(G)$ e o vetor global *nr_bloqueios*.

```
1 para cada u ∈ Adj(v) faça
2     se (nr_bloqueios(u) > 0) então
3         nr_bloqueios(u) ← nr_bloqueios(u) − 1.
```

Algoritmo 4.6: *CC_Visit*$(p, Csc, chave, nr\_bloqueios)$

Entrada: O caminho sem corda $p = \langle u_1, u_2, \ldots, u_t \rangle$; o conjunto *Csc* de ciclos sem corda; $chave = \ell(u_2)$, que é o menor elemento do caminho sem corda; e o vetor global *nr_bloqueios*.

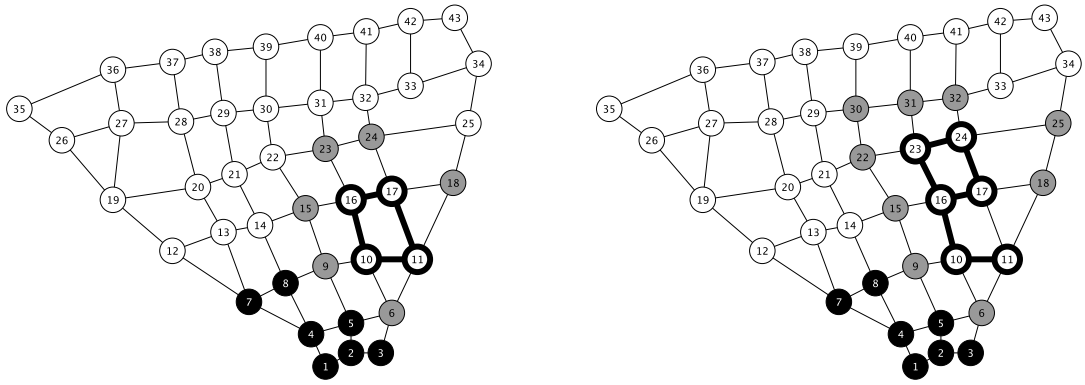
Saída: O conjunto *Csc* de caminhos sem corda atualizado.

```
1  BloqueiaVizinhos(u_t)
2  para cada v ∈ Adj(u_t) faça
3      se ((ℓ(v) > chave) e (nr_bloqueios(v) = 1)) então
4          p' ← ⟨p, v⟩
5          se ((v, u_1) ∈ E(G)) então
6              Csc ← Csc ∪ {p'}
7          senão
8              Csc ← CC-Visit(p', Csc, chave)
9  DesbloqueiaVizinhos(u_t)
10 retorna Csc
```

O Algoritmo 4.6 (*CC_Visit*) estende um caminho sem corda p a partir do último vértice usando uma estratégia de DFS. Em cada passo da recursão, ele verifica qual será o resultado da adição de cada vizinho v do último vértice no caminho atual. Existem três opções para $\langle p, v \rangle$:

(a) tem uma corda;
(b) é um ciclo sem corda; ou
(c) é um caminho sem corda.

Se $\langle p, v \rangle$ forma uma corda, $nr\_bloqueios(v) \geq 2$ e $\langle p, v \rangle$ não satisfaz a condição da Linha 3 do Algoritmo 4.6, como explicado na Figura 4.8(b) para $v = 17$. Se a adição de v a p resulta em um ciclo sem corda, $\langle p, v \rangle$ é adicionado ao conjunto *Csc* de ciclos sem corda e a recursividade termina, como pode ser visto na Figura 4.8(a). Se $\langle p, v \rangle$ é um caminho sem corda, o Algoritmo *CC_Visit* é chamado novamente.



(a) *Exemplo de um ciclo sem corda* $\langle 11,10,16,17\rangle$.

(b) *Exemplo de uma corda* $\langle 16,17\rangle$ *no caminho.*

Figura 4.8: *Exemplo de ciclo sem corda e sem corda.*

Lembramos que, pelo Lema 4.1, se existe um caminho sem corda $p = \langle u_1, u_2, \ldots, u_t\rangle$ e um caminho mais curto $q = \langle v_1, v_2, \ldots, v_k\rangle$, no subgrafo induzido $G' = G - \left(\bigcup_{i=2}^{t-1} Adj[u_i] - \{u_1, u_t\}\right)$, entre u_1 e u_t, então $\langle p, q\rangle$ forma um ciclo sem corda.

Para garantir que cada expansão realizada a partir de um caminho sem corda encontre um ciclo sem corda, as Linhas 1–3 do Algoritmo 4.6 são modificados a fim de utilizar uma busca em largura (BFS) no subgrafo induzido pela remoção de vértices e dos seus vizinhos do caminho atual. Todos os vértices $v \in V(G)$ tal que $\ell(v) < \ell(u_2)$ também são descartados.

A seguir, o trecho do algoritmo que substitui as Linhas 1–3 do Algoritmo 4.6 mostra essa modificação, onde $\pi(v)$ denota o antecessor do vértice v no caminho da árvore de busca gerada pela execução da BFS.

```
BFS(u_1, G − ({v | nr_bloqueios(v) ≠ 0 ou ℓ(v) < ℓ(u_2)} − {u_1, u_t}))
BloqueiaVizinhos(u_t)
se (∃ π(u_t)) então
    para cada v ∈ Adj(u_t) faça
        se (∃ π(v)) então
```

No início do Algoritmo 4.6 $G' = G - (\{v \mid nr\_bloqueios(v) \neq 0\} - \{u_1, u_t\})$ Assim, dado o caminho $\langle u_1, u_2, \ldots, u_t\rangle$, a expansão será realizada somente se u_t é um descendente de u_1 na árvore de busca (isto é caracterizado pela existência de $\pi(u_t)$). Neste caso, um vértice v adjacente a u_t será considerado para a expansão deste caminho somente se v é um descendente de u_1 na árvore de busca e $\ell(v) > \ell(u_2)$, o que garante a existência de ciclos sem corda pelo Lema 4.1.

Usando-se a estratégia de BFS, é possível verificar em tempo $O(n+m)$ se dois vértices u e v pertencem ao mesmo componente conexo. Neste caso, qualquer caminho

sem corda pode ser estendido a um ciclo sem corda. De fato se temos um caminho de u_1 a u_t em G', então a condição da Linha 3 do Algoritmo 4.6 é satisfeita por pelo menos um vértice.

4.3 Análise dos algoritmos

A corretude dos algoritmos é devida ao fato que nenhum vértice é mantido bloqueado no final da sua execução, o que é garantido pelo lema abaixo.

Lema 4.8 *Seja $p = \langle u_1, u_2, \ldots, u_t \rangle$ um caminho sem corda. No início e no final da execução do Algoritmo CC_Visit(), $nr\_bloqueios(v) = k$ se e somente se v é um vizinho de k vértices em $\{u_2, \ldots, u_{t-1}\}$, para qualquer vértice $v \in V(G)$.*

Prova. Como na prova do Lema 4.7, não é difícil ver que no início de qualquer execução do Algoritmo $CC\_Visit()$ o contador $nr\_bloqueios(v)$ é incrementado em 1 para todos os vizinhos de v em $\{u_2, \ldots, u_{t-1}\}$. No final do processo, são decrementados cada um destes valores, o que é garante que $nr\_bloqueios(v) = k$. □

Seja $\langle u_1, \ldots, u_t \rangle$ um caminho sem corda e $v \in Adj(u_t)$. Dos Lemas 4.7 e 4.8 é fácil ver que, depois da chamada do Algoritmo *BloqueiaVizinhos(u_t)* para bloquear os vizinhos de u_t para $v \in Adj(u_t)$, $nr\_bloqueios(v) = 1$ se e somente se $\langle p, v \rangle$ é um caminho sem corda ou um ciclo sem corda.

Usando os Lemas 4.2, 4.7 e 4.8 agora nós demonstramos a corretude dos algoritmos propostos, como estabelecido pelo teorema seguinte.

Teorema 4.9 Corretude do Algoritmo *CiclosSemCorda(G)*.
O Algoritmo CiclosSemCorda(G) enumera todos os ciclos sem corda do grafo G.

Prova. Seja $C_k = \langle u_1, u_2, u_3, \ldots, u_k \rangle$ um ciclo sem corda de G. Se $k = 3$, então o ciclo é encontrado na Linha 2. Pelo Lema 4.2 podemos assumir que $\ell(u_2) = \min\{\ell(u_i) \mid i = 1, \ldots, k\}$ e $\ell(u_1) < \ell(u_3)$. Como $k > 3$, temos que $(u_1, u_3) \notin E(G)$. Portanto, a tripla $\langle u_1, u_2, u_3 \rangle$ é gerada pelo Algoritmo 4.1 ($Triplas(G, \ell)$. Então, o Algoritmo *CiclosSemCorda(G)* executa as Linhas 6–9 com $p = \langle u_1, u_2, u_3 \rangle$.

Agora, sejam $\langle u_1, \ldots, u_t \rangle$ um caminho sem corda e $v \in Adj(u_t)$. Combinando-se os Lemas 4.7 e 4.8, depois da chamada do Algoritmo *BloqueiaVizinhos(u_t)* em *CC_Visit* para bloquear os vizinhos de u_t, $nr\_bloqueios(v) = 1$ se e somente se $\langle p, v \rangle$ é um caminho sem corda ou um ciclo sem corda. Neste caso, o Algoritmo *CC_Visit()* será chamado novamente até, eventualmente, $t = k$ e encontrar o ciclo C_k; no segundo caso já temos o ciclo sem corda desejado e ele é adicionado ao conjunto Csc. □

Observamos que a profundidade da árvore de busca resultante da chamada ao Algoritmo *CC_Visit*() é no máximo o tamanho do maior caminho sem corda. No caso do algoritmo com BFS, o tamanho do maior ciclo sem corda. Além disso, o número da chamadas ao BFS é limitado pelo tamanho da saída. Visto que o BFS é executado em complexidade de tempo $O(n+m)$, nosso algoritmo tem complexidade de tempo $O((n+m)\cdot tam(Csc))$, isto é, $O(n+m)$ no tamanho da saída. Analisamos o algoritmo em relação ao tamanho da saída, porque mesmo em grafos pequenos a quantidade de ciclos sem corda pode ser exponencial. Na verdade, o BFS opera em um subgrafo que diminui a cada iteração. O melhor algoritmo do qual temos conhecimento para encontrar todos os ciclos sem corda em um grafo G [56] tem complexidade de tempo $O(n\cdot(n+m)\cdot tam(Csc))$, visto que ele encontra o mesmo ciclo sem corda mais de uma vez. O algoritmo sem BFS tem tempo de execução melhor, como podemos ver na Seção 4.4.

Com o objetivo de reduzir o tempo de execução do algoritmo, estratégias para identificação de componentes biconexos apresentadas por Tarjan [52] e Szwarcfiter [50], que têm complexidade de tempo $O(n^2)$, podem ser usadas. Estas estratégias descartam todos os vértices com $\delta(G)<2$ e alguns caminhos que não podem levar a um ciclo sem corda.

Para melhorar o desempenho do Algoritmo 4.3, é possível fazer um pré-processamento de cada tripla $\langle x,u,y\rangle \in T(G)$ para tentar estender o caminho sem corda através da extremidade x. Já que um caminho sem corda é estendido somente através da extremidade y. Este pré-processamento é detalhado no Algoritmo 4.7 (*PrimeExtends*()). Esta extensão pode ser realizada somente quando existe apenas um vértice próximo à extremidade do caminho que pode ser adicionado a ele, ou seja, um vizinho desbloqueado. Assim, o caminho sem corda poderia ser estendido de uma maneira única e não mudaria o número de caminhos sem corda examinados. Além disso, esta extensão pode reduzir o trabalho do Algoritmo 4.3, visto que bloqueia os vizinhos de vértices visitados que não necessitam ser examinados novamente.

Algoritmo 4.7: $PrimeExtends(p,q,Csc,chave,bloqueado)$

Entrada: Caminhos $p = \langle x,u,y\rangle$ e $q = \langle v_1,v_2,\ldots,v_{s-1}\rangle$, tal que o caminho $\langle q,x,u,y\rangle$ é um caminho sem corda; o conjunto Csc de ciclos sem cordas; $chave = \ell(u)$; e um vetor global *nr_bloqueios*.

Saída: Um novo caminho $q' = \langle z,v_1,v_2,\ldots,v_{s-1}\rangle$, tal que $\langle q',x,u,y\rangle$ é um caminho sem corda e o novo conjunto Csc de ciclos sem corda.

```
1  contador ← 0;
2  para cada v ∈ Adj(v_1) faça
3      se (ℓ(v) > chave) então
4          se (nr_bloqueios(v) = 0)) então
5              contador ← contador + 1
6              z ← v
7          se (nr_bloqueios(v) = 1) então
8              se ((v,y) ∈ E(G)) então
9                  p' ← ⟨p,v,q⟩
10                 Csc ← Csc ∪ {⟨p'⟩}
11 se (contador = 1) então
12     se ((z,y) ∈ E(G)) então
13         p' ← ⟨p,v,q⟩
14         Csc ← Csc ∪ {⟨p'⟩}
15     senão
16         BloqueiaVizinhos(v_1)
17         q' ← ⟨z,q⟩
18         q' ← PrimeExtends(p,q',Csc,chave)
19 retorna q',Csc
```

Algoritmo 4.8: $DesbloqueiaVizinhosPE(q)$

Entrada: Um caminho sem corda $q = \langle v_1,v_2,\ldots,v_s\rangle$.

```
1  para i = 2 até s faça
2      DesbloqueiaVizinhos(v_i)
```

Os bloqueios feitos através da extremidade x, depois de todas as possíveis extensões, previnem que extensões feitas pela extremidade y usem qualquer vértice já inserido na extremidade oposta, sem formar um ciclo sem corda. Eles também evitam o uso de vértices que já foram bloqueados, os quais poderiam formar cordas no caminho.

No lugar de processar uma tripla, inicialmente o Algoritmo 4.6 (CC_Visit) processará a tripla junto com o caminho q' encontrado pelo Algoritmo 4.7. O parâmetro q'

será então adicionado como entrada do algoritmo e o caminho p' na Linha 4 será atualizado de forma coerente. O Algoritmo 4.8 ($DesbloqueiaVizinhosPE(q)$) deve ser executado depois da chamada inicial do Algoritmo 4.6 no algoritmo principal (Algoritmo 4.3). O lema abaixo garante que nenhum vértice bloqueado através da extremidade x continua neste estado depois do processamento de uma tripla.

Lema 4.10 *Sejam $p = \langle x,u,y\rangle$ e $q = \langle v_1,v_2,\ldots,v_{s-1}\rangle$ caminhos em G, tal que $\langle q,x,u,y\rangle$ é um caminho sem corda. No início de cada execução do passo de extensão, para qualquer vértice $v \in V(G)$, $nr\_bloqueios(v) = k$ se e somente se v é um vizinho de k vértices em $\{v_2,\ldots,v_{s-1},x,u\}$.*

Prova. Antes do início da primeira execução de um passo de extensão, o contador $nr\_bloqueios(v)$ é incrementado em 1 unidade para todos os vizinhos de u. Durante uma execução do algoritmo, o caminho q é aumentado para $\langle z,q\rangle$, para algum $z \in Adj(v_1)$, e o contador $nr\_bloqueios(v)$ é incrementado em 1 para todos os vizinhos de v_1. Logo, o resultado segue para qualquer execução do passo de extensão visto que o contador $nr\_bloqueios(v)$ é incrementado em 1 unidade para todos vizinhos de cada vértice em $\{v_2,\ldots,v_{s-1},x,u\}$. □

Sejam $\langle q,x,u,y\rangle = \langle v_1,v_2,\ldots,v_{s-1},x,u,y\rangle$ um caminho sem corda e $v \in Adj(v_1)$. Combinando-se os Lemas 4.7 e 4.10, é fácil ver que $nr\_bloqueios(v) = 0$ se e somente se $\langle v,q,x,u,y\rangle$ é um caminho sem corda e $(v,y) \notin E(G)$ ou $\langle v,q,x,u,y\rangle$ é um ciclo sem corda e $(v,y) \in E(G)$. Neste último caso, o ciclo $\langle v,q,x,u,y\rangle$ é equivalente ao ciclo $\langle x,u,y,v,q\rangle$. Além disso, o Lema 4.10 estabelece que nenhum vértice se manterá bloqueado depois da extensão através da extremidade x, enquanto que o Lema 4.8 e o uso do Algoritmo 4.8 garante que o mesmo ocorre para a extremidade y.

4.4 Resultados experimentais

Na implementação do algoritmo $CiclosSemCorda(G)$ usamos duas estruturas de dados: uma matriz de adjacências, que permite a verificação da adjacência entre dois vértices em tempo constante, e também uma representação compacta de grafos como proposto por Harish e Narayanan [28]. Este modelo de representação de grafos é composto por dois vetores V_a e E_a. Cada índice do vetor V_a corresponde a um vértice. O seu conteúdo contém o índice para o primeiro vértice de sua lista de adjacência, que está contida no vetor de adjacência E_a.

A Figura 4.9(b) ilustra uma representação compacta para o grafo da Figura 4.9(a). No exemplo, o vértice 0 é adjacente aos vértices 1 e 4. Como pode ser observado nas figuras, a mesma lógica é seguida para os demais vértices.

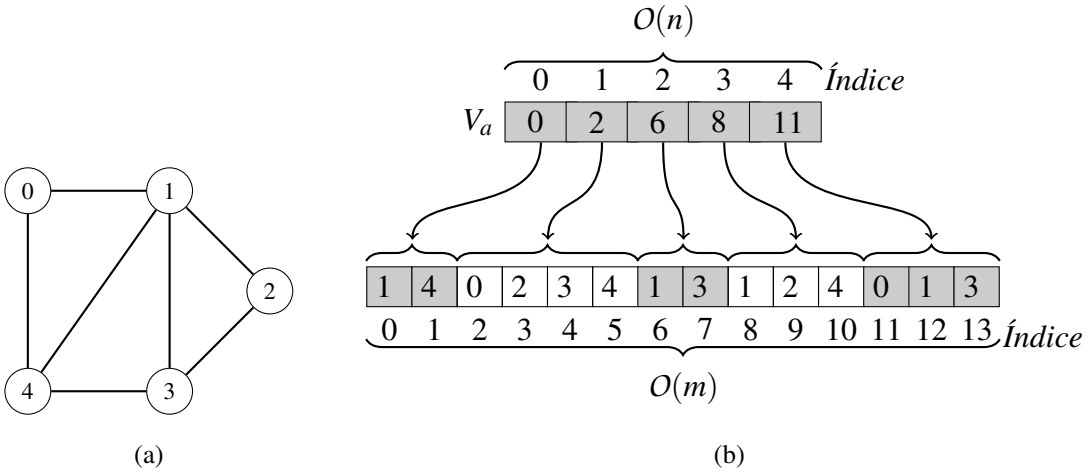


Figura 4.9: *Grafo (a) e sua representação compacta (b).*

Conforme ilustrado na Figura 4.9(b), podemos concluir que o espaço de armazenamento para esse modelo de representação é $O(n+m)$.

Essa representação possui o mesmo espaço de armazenamento que a lista de adjacências. Sendo assim, seu uso se deve a algoritmos de programação paralela, por exemplo CUDA *(Compute Unified Device Architecture)*, em que a lista de adjacências não é eficiente. Utilizamos esta representação na versão paralela do artigo (veja [32]), que foi implementada em OpenCL.

A implementação e execução do algoritmo foi realizada usando a linguagem C++ e o compilador g++ em um Sistema Operacional Linux openSUSE 12.3 “Dartmouth”, um HP Proliant DL380 G7 Xeon Quad Core E5506 2.13GHZ com 40GB de memória RAM e 1.6 TB de disco rígido.

O tempo de execução (em segundos) para os grafos apresentados em [47], representando algumas redes ecológicas, chamadas *food webs*, e também outros grafos bem conhecidos são mostrados na Tabela 4.1. A coluna denominada “Nome” contém o nome do grafo, n é a quantidade de vértices, m é a quantidade de arestas, #*csc* é a quantidade de ciclos sem corda de tamanho quatro ou mais e C_3 é a quantidade de ciclos de tamanho três no grafo. T_1, T_2, T_3 e T_4 representam, respectivamente, o tempo de execução (em segundos) do algoritmo de Sokhn et al. [47] como originalmente apresentado pelos autores e também quando executado em nosso computador, do nosso algoritmo e de sua versão modificada usando BFS. O símbolo “–” na coluna T_1 refere-se a grafos não testados por Sokhn et al. [47], portanto apenas apresentamos resultados do algoritmo deles quando executado em nosso computador.

Na Tabela 4.2, *cml*, #*vis* e #*rec* referem-se, respectivamente, ao comprimento do caminho sem corda mais longo no grafo, à quantidade de visita de vértices e à quantidade de recursões do *CC_Visit* realizadas na busca pelos ciclos sem corda.

Tabela 4.1: *Tempo de execução para enumerar todos os ciclos sem corda.*

Id.	**Nome**	n	m	**#csc**	C_3	T_1	T_2	T_3	T_4
1	CrystalD	16	86	0	293	–	0.00	0.00	0.00
2	ChesUpper	24	85	0	167	–	0.00	0.00	0.00
3	Narragan	26	168	0	586	–	0.00	0.00	0.00
4	Chesapeake	27	90	0	157	0.00	0.00	0.00	0.00
5	Michigan	29	175	0	587	–	0.00	0.00	0.00
6	Mondego	30	206	0	886	–	0.00	0.00	0.00
7	Cypwet	53	842	0	8946	6.00	0.01	0.01	0.01
8	Everglades	58	1214	710	15627	7.00	0.03	0.03	0.04
9	Mangrovedry	86	2132	27426	30659	359.00	1.10	0.31	0.78
10	Floridabay	107	3249	85976	62389	4569.00	11.57	1.03	5.37
11	Goiânia	43	75	9311	5	–	0.54	0.11	0.19
12	C_{100}	100	100	1	0	–	0.00	0.00	0.00
13	Roda 100	101	200	1	100	–	0.00	0.00	0.00
14	$K_{8,8}$	16	64	784	0	–	0.00	0.00	0.00
15	$K_{50,50}$	100	2500	1500625	0	–	1.17	1.58	2.88
16	Grade 4×10	40	66	1823	0	–	0.13	0.03	0.05
17	Grade 5×6	30	49	749	0	–	0.01	0.00	0.01
18	Grade 5×10	50	85	52620	0	–	2.20	0.60	1.13
19	Grade 6×6	36	60	3436	0	–	0.07	0.02	0.06
20	Grade 6×10	60	104	800139	0	–	37.79	9.15	16.83
21	Grade 7×10	70	123	8136453	0	–	678.09	85.23	189.86

Os primeiros dez grafos (Ids. 1–10) foram providos por base de dados conhecidas de estudos ecológicos, no qual um grafo de cadeia alimentar (orientado) é transformado em um grafo de competição (não orientado) de acordo com as definições de Wilson and Watkins [58]. Para mais informações, veja a Seção 2.1. Para obter uma melhor comparação entre os tempos de execução obtidos por Sokhn et al. [47], nós também excluímos os vértices de grau 0 nos grafos da Tabela 4.1.

A Figura 4.10 mostra um grafo que representa o centro da cidade de Goiânia, capital do estado de Goiás, Brasil. O tempo de execução para enumerar todos os ciclos sem corda do grafo é apresentado na Linha 11 da Tabela 4.1. Três dos 9316 ciclos sem corda presentes no grafo estão destacados na figura.

Os últimos dez grafos (Ids. 11–21) são bem conhecidos na área de Teoria dos Grafos. Mais informações podem ser encontradas na Seção 2.1.

Nós podemos observar na Tabela 4.1 que o tempo de execução do nosso algoritmos sem BFS é mais rápido do que a versão usando BFS, embora a quantidade de recursão do *CC_Visit* seja menor. A princípio, a busca usando BFS deveria ter um menor tempo de execução, mas isto não ocorre porque o BFS, que é usado para verificar a existência de um caminho entre dois vértices, pode potencialmente requerer complexidade de tempo $O(n)$ para cada extensão do caminho sem corda.

Tabela 4.2: *Mais informação sobre a execução dos algoritmos.*

			Algoritmo sem BFS		**Algoritmo usando BFS**	
Id.	$\|T(G)\|$	**cml**	**#vis**	**#rec**	**#vis**	**#rec**
1	16	3	1 205	16	1 293	16
2	31	3	7 119	307	2 023	31
3	59	3	9 646	215	6 062	59
4	21	3	1 057	27	1 102	21
5	92	3	14 212	257	9 563	92
6	80	3	35 582	826	8 776	80
7	909	3	244 559	2 473	193 457	909
8	1 877	6	1 008 576	10 283	755 141	2 410
9	4 095	8	23 211 495	226 078	15 984 224	42 157
10	5 837	8	108 212 550	685 492	107 989 118	273 130
11	31	28	1 298 846	128 623	1 278 623	36 785
12	1	100	587	97	10 287	97
13	1	100	980	97	15 530	97
14	140	4	5 740	140	13 132	140
15	61 250	4	15 373 750	61 250	93 467 500	61 250
16	27	28	412 944	44 846	308 928	9 648
17	20	18	34 896	3 739	75 669	2 365
18	36	32	7 161 919	729 706	7 334 301	225 960
19	25	20	178 672	18 671	362 840	10 833
20	45	36	108 866 318	10 696 603	105 704 459	3 088 973
21	54	44	1 447 348 446	140 095 162	1 128 865 130	30 945 512

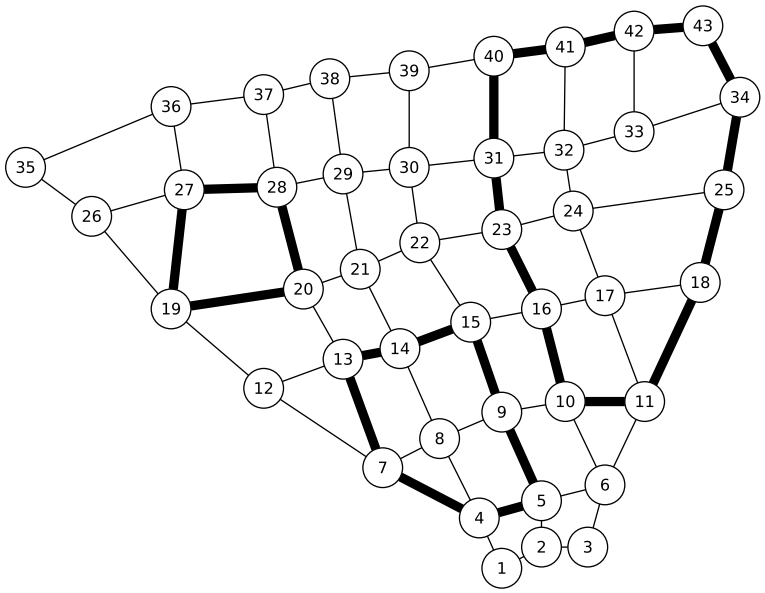


Figura 4.10: *Representação simples do centro de Goiânia, Goiás, Brasil. Os caminhos destacados são exemplos de ciclos sem corda.*

Os principais resultados deste capítulo podem ser encontrados no nosso artigo intitulado “Efficient enumeration of chordless cycles” [14]. Com base no algoritmo sequencial, desenvolvemos uma versão paralela para GPU utilizando a linguagem de programação OpenCL. Para mais detalhes, veja o nosso artigo intitulado “A GPU-based parallel algorithm for enumerating all chordless cycles in graphs” [32].

CAPÍTULO 5

Grafos Ciclicamente Orientáveis e Orientados

Neste capítulo, iremos mostrar algumas propriedades de grafos ciclicamente orientáveis e ciclicamente orientados.

A família de grafos denominada ciclicamente orientáveis (CO) foi introduzida por Barot, Geiss e Zelevinsky em [2]. Um grafo G é CO se ele admite uma orientação na qual qualquer ciclo sem corda é ciclicamente orientado. Tal orientação é chamada de cíclica. Motivados pela sua aplicação nas álgebras *cluster*, eles obtiverem boas caracterizações em relação a essa classe de grafos e as utilizaram para desenvolver um teste cujo objetivo é determinar se uma álgebra *cluster* é de tipo finito.

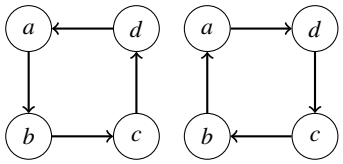
Novas caracterizações, obtidas por Gurvich [26] e Speyer [48] em trabalhos posteriores, possibilitaram a elaboração de algoritmos para reconhecer grafos ciclicamente orientáveis e para obter sua orientação cíclica em tempo linear.

Para um grafo CO G, nós mostramos que a quantidade de ciclos sem corda é polinomial no tamanho de G. Apresentamos um algoritmo que verifica se o grafo dado é ciclicamente orientável e, em caso positivo, enumera todos os ciclos sem corda em tempo polinomial. Esse algoritmo pode ser facilmente alterado para verificar se um grafo é ciclicamente orientado. Comentamos como fazer isso no final da Seção 5.2.

O capítulo está organizado como segue: algumas definições preliminares, observações e propriedades dos grafos CO são apresentadas na Seção 5.1; dois algoritmos são mostrados na Seção 5.2: um para verificar se um grafo é CO e, em caso positivo, listar todos os ciclos sem corda e outro para atribuir uma orientação cíclica a um grafo ciclicamente orientável; a análise da corretude e complexidade de tempo dos algoritmos é apresentada na Seção 5.3.

5.1 Grafos ciclicamente orientáveis

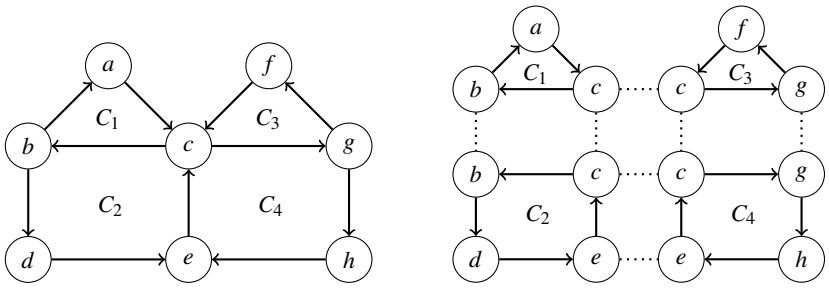
A *orientação* de um grafo G consiste na atribuição de uma ordem às extremidades de cada uma de suas arestas. A orientação de um ciclo com n vértices é chamada *cíclica* se ele recebe a orientação $\langle (1,2),\ldots,(n-1,n),(n,1)\rangle$ ou a orientação oposta.



(a) (b)

Figura 5.1: *Duas possíveis orientações cíclicas para o ciclo C_4.*

Um grafo G é chamado de ciclicamente orientado (COd) se estiver orientado de forma cíclica (Figura 5.2).



(a) (b)

Figura 5.2: *O grafo (a) está ciclicamente orientado, visto que todos os seus ciclos sem corda (b) estão orientados de forma cíclica.*

Proposição 5.1 *Os grafos roda W_k não são ciclicamente orientáveis.*

Prova. Seja W_k um grafo roda, para $n \geq 3$. Sabemos que $V(W_k) = V(C_k) \cup \{w\}$ e $E(W_k) = E(C_k) \cup \{x_i w \mid x_i \in V(C_k)\}$.

Podemos supor, sem perda de generalidade, que a orientação do ciclo $\langle x_1, x_2, w\rangle$ é $\langle x_1 \to x_2 \to w \to x_1\rangle$. Logo, o ciclo $\langle x_2, x_3, w\rangle$ deve ter a seguinte orientação: $\langle x_2 \to w \to x_3 \to x_2\rangle$. Portanto, o ciclo $\langle x_1, x_2, \ldots, x_n\rangle$ não é ciclicamente orientável, pois $\langle x_1 \to x_2 \leftarrow x_3\rangle$. Isto completa a prova. □

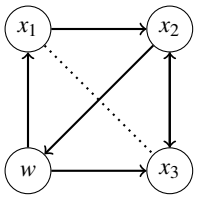


Figura 5.3: *Grafo W_k não é ciclicamente orientável.*

Visto que o grafo K_4 é isomorfo ao grafo W_3, temos o corolário seguinte.

Corolário 5.2 *O grafo K_4 não é ciclicamente orientável.*

Prova. Segue diretamente do fato que o grafo K_4 é isomorfo a W_3. □

Outro grafo que é fácil ver que não é ciclicamente orientável é apresentado no exemplo a seguir.

Exemplo 5.3 *Sem perda de generalidade, orientamos o ciclo sem corda $\langle a,c,e,d\rangle$ no sentido anti-horário. Logo, o ciclo $\langle a,c,e,b\rangle$ deve ser orientado no sentido horário. Sendo assim, o ciclo $\langle a,b,e,d\rangle$ não é orientado de forma cíclica.*

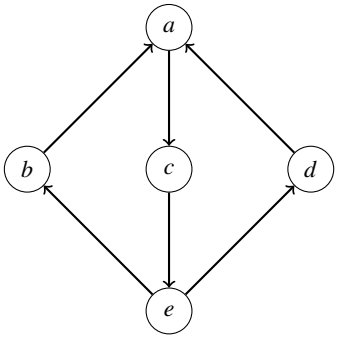


Figura 5.4: *Construção de grafo não ciclicamente orientável.*

O resultando seguinte abrange os dois casos apresentados acima.

Teorema 5.4 (Gurvich [26]) *Um grafo G é ciclicamente orientável se e somente se satisfaz as propriedades:*

(a) G não contém K_4.
(b) Dados dois vértices $a,b \in V(G)$, se existirem três caminhos entre a e b, então existe um vértice distinto de a e b pertencente a pelo menos dois dos três caminhos ou $(a,b) \in E(G)$.

Dado um grafo G, denotamos $con(G)$ seu conjunto de componentes conexos, Csc o conjunto de seus ciclos sem corda. e $\#csc(G)$ a cardinalidade de Csc.

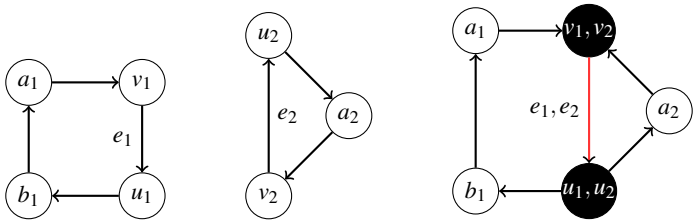
Proposição 5.5 (Barot, Geiss e Zelevinsky [2]) *A seguinte propriedade funciona para um grafo finito G arbitrário: $\#csc(G) \geq m - n + |con(G)|$.*

Teorema 5.6 (Barot, Geiss e Zelevinsky [2]) *Um grafo finito G é ciclicamente orientável se e somente se $\#csc(G) = m - n + |con(G)|$.*

Os dois parágrafos a seguir são condições preliminares para o entendimento do Lema 5.7 e do Teorema 5.8.

É possível identificar se um grafo G é ciclicamente orientável avaliando-se individualmente cada um de seus componentes biconexos. Se todos eles forem ciclicamente orientáveis, o grafo G será ciclicamente orientável. Da mesma forma para COd.

Se G_1 e G_2 são grafos ciclicamente orientáveis e e_1 e e_2 são, respectivamente, arestas de G_1 e G_2, cada uma com sua orientação, podemos escrever $G_1 \bigcup_{e_1,e_2} G_2$ como sendo o grafo gerado pela união de G_1 e G_2 ao longo das arestas e_1 e e_2, de maneira compatível com as orientações de e_1 e e_2 (Figura 5.5).



(a) G_1 (b) G_2 (c) $G_1 \bigcup_{e_1,e_2} G_2$

Figura 5.5: *A união dos grafos (a) e (b) pelas arestas e_1 e e_2 produz o grafo (c).*

Lema 5.7 (Speyer [48]) *Seja G um grafo biconexo ciclicamente orientável que não é um ciclo ou uma aresta simples. Então, $G = G_1 \bigcup_{e_1,e_2} G_2$, onde cada G_i é um grafo biconexo ciclicamente orientável que não é uma aresta simples e e_i é uma aresta com orientação. Além disso, todo grafo dessa forma é ciclicamente orientável.*

O teorema a seguir mostra claramente a construção destes grafos.

Teorema 5.8 (Speyer [48]) *Um grafo G é ciclicamente orientável se e somente se todos os seus componentes biconexos também o são. Um grafo biconexo é ciclicamente orientável se e somente se é um ciclo, uma aresta simples ou é da forma $G' \cup C_k$, onde G' é um grafo biconexo ciclicamente orientável, C_k é um ciclo e G' e C_k se conectam por uma aresta simples. Além disso, se $G = G' \cup C_k$ é qualquer decomposição de G em um ciclo e um subgrafo que se encontra em uma aresta simples, então G é ciclicamente orientável se e somente se G' também o é.*

Sejam G' e G'' grafos ciclicamente orientáveis e e_1 e e_2, respectivamente, arestas de G' e G''. É importante ressaltar que a união entre G' e G'' gerará um grafo ciclicamente orientável apenas nos casos em que for feita ao longo de uma única aresta, $G' \bigcup_{e_1,e_2} G''$, sendo e_1 e e_2 unidas de forma que a direção de orientação seja a mesma (Figura 5.6).

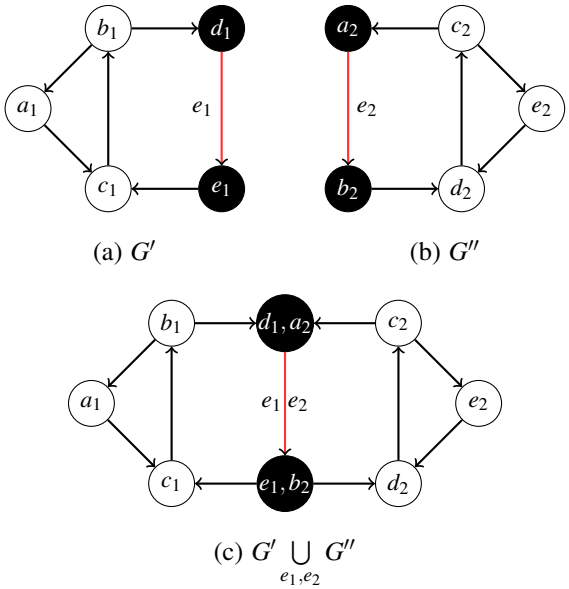


Figura 5.6: *União de grafos ciclicamente orientados.*

Como podemos observar na Figura 5.7(a), a união dos ciclos sem corda C_1 e C_2 ao longo das arestas e_{11}, e_{21} e e_{12}, e_{22}, gera como resultado o grafo, não ciclicamente orientado, da Figura 5.7(b). Este possui três ciclos sem corda, C_1, C_2 e C_3, estando este último orientado de forma não cíclica.

Portanto, para que a união entre os ciclos sem corda C_1 e C_2 da Figura 5.7(a) resulte em um grafo ciclicamente orientável, os dois ciclos devem ser unidos ao longo de uma única aresta, como podemos ver na Figura 5.7(c), em que C_1 foi unido a C_2 ao longo das arestas e_{13} e e_{23}.

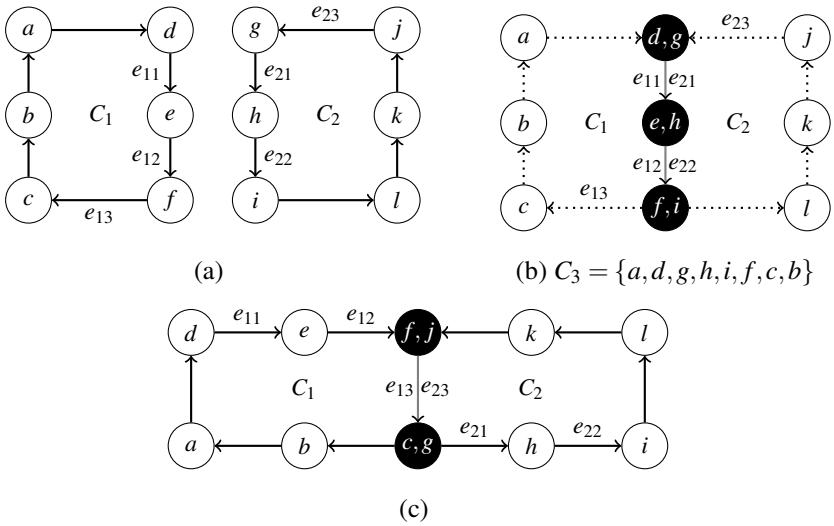


Figura 5.7: *Unindo ciclos sem corda.*

A proposição seguinte nos permite fornecer, além de uma condição necessária para um grafo ser CO, um limite na quantidade de ciclos.

Proposição 5.9 (Speyer [48]) *Se G é um grafo ciclicamente orientável com n vértices, então G tem no máximo, $2 \cdot n - 3$ arestas.*

Lema 5.10 *Seja um grafo G. Para todo $v \in V(G)$, se $grau(v) \geq 3$, então G não é CO.*

Prova. Suponha que $grau(v) \geq 3$ para todo $v \in V(G)$. Logo, temos que $m \geq 3 \cdot n$. É fácil ver que $3 \cdot n > 2 \cdot n - 3$. Portanto, pela Proposição 5.9, o grafo G não é CO. □

Corolário 5.11 *A quantidade de ciclos sem corda de um grafo conexo ciclicamente orientável é menor ou igual a $n - 2$.*

Prova. Como $m \leq 2 \cdot n - 3$, pelo Teorema 5.6, temos que $\#csc(G) = m - n + 1$. Logo, $\#csc(G) \leq n - 2$. □

5.2 Algoritmos para grafos ciclicamente orientáveis

Com base em teoremas e proposições descritas por Speyer [48], o Algoritmo 5.1 é capaz de verificar se um grafo G é ciclicamente orientável. Em caso positivo, ele retorna todos os ciclos sem corda de G, como mostrado no Teorema 5.12.

O algoritmo é baseado na análise de cada componente biconexo encontrado a partir de um grafo dado como entrada. Seguindo exatamente a ideia do Teorema 5.8, o algoritmo busca, para um componente biconexo, identificar um ciclo sem corda C_k tal que $G = G' \cup C_k$. Isso é feito visando-se reduzir o componente biconexo inicial a um único ciclo e assim mostrar que ele é ciclicamente orientável.

Inicialmente, o algoritmo verifica se o grafo dado como entrada está em acordo com a Proposição 5.9. Se não, ele retorna NÃO. Na sequência, ele encontra todos os componentes biconexos e faz a mesma verificação. Um algoritmo para encontrar todos os componentes biconexos é apresentado no Capítulo 2.

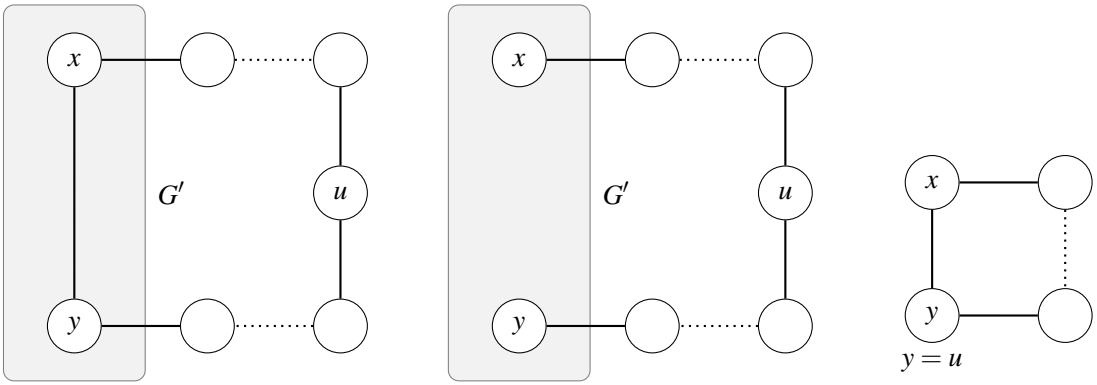
Para cada componente biconexo, o algoritmo armazena em uma fila F todos os vértices de grau 2. Se existirem tais vértices, o algoritmo retorna NÃO. Vértices são removidos e novos são adicionados a F à medida que o algoritmo é executado. Isso ocorrerá até que todos os vértices de grau dois sejam visitados. Se G é CO, então todos os vértices passarão exatamente uma vez em F.

O algoritmo tenta, a partir dos vértices na lista F, encontrar e eliminar caminhos (ciclos) de forma a reduzir o componente biconexo inicial a um ciclo e, assim, constatar que o grafo é CO. Após constatar que um componente biconexo é CO, outro componente será analisado. Ao final do processo, o grafo dado como entrada será classificado

como CO (e o algoritmo retorna SIM) se todos os componentes biconexos receberem a classificação de CO; caso contrário, será classificado como não CO. Portanto, dado um grafo biconexo G, ele determina, em complexidade de tempo $O(n^2)$, se este grafo é CO e retorna o conjunto de todos os ciclos sem corda Csc de G. O pior caso ocorre quando o grafo é CO e o melhor caso quando $m > 2 \cdot n - 3$, em que a complexidade de tempo é constante.

A função $cor(v)$ tem a mesma conotação utilizada no algoritmo DFS. A função $chave(v)$ guarda o caminho representado pelo vértice v.

Os três possíveis casos do Algoritmo 5.1 são apresentados na figura a seguir.



(a) *Caso 1 - Linha 24.* (b) *Caso 2 - Linha 31.* (c) *Caso 3 - Linha 35.*

Figura 5.8: *Casos do Algoritmo 5.1.*

O Caso 1 é transformado em G', o Caso 2 é transformado no grafo da Figura 5.9 e o Caso 3 para.

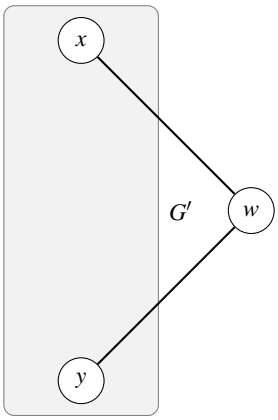


Figura 5.9: *Caso 2 do Algoritmo 5.1.*

Algoritmo 5.1: $CiclosSemCordaGrafosCO(G)$

Entrada: Um grafo simples não orientado G.

Saída: Resposta se G é CO e, se for, o conjunto Csc de ciclos sem corda de G.

```
1  se (|E(G)| > 2·|V(G)| − 3) então
2      retorna NAO
3  senão
4      para cada componente biconexo não simples G_i de G faça
5          se (|E(G_i)| > 2·|V(G_i)| − 3) então
6              retorna NAO
7      Csc ← ∅
8      para cada componente biconexo não simples G_i de G faça
9          inicialize a fila F com todos vértices de grau
10         inicialize cor(u) = BRANCA para u ∈ V(G_i)
11         inicialize chave(u) = u para u ∈ V(G_i)
12         enquanto (F ≠ ∅) faça
13             pegue o primeiro elemento u da fila F
14             se (cor(u) = BRANCA) então
15                 P ← ∅;   y ← u;   cor(u) ← CINZA
16                 x ← a, tal que a ∈ Adj(u) e cor(a) = BRANCA
17                 enquanto ((grau(x) = 2) e (∃a ∈ Adj(x) : cor(a) = BRANCA)) faça
18                     cor(x) ← CINZA
19                     P ← ⟨chave(x), P⟩;   x ← a
20                 enquanto ((grau(y) = 2) e (∃b ∈ Adj(y) : cor(b) = BRANCA)) faça
21                     cor(y) ← CINZA
22                     P ← ⟨P, chave(y)⟩;   y ← b
23                 se (y ≠ u) então
24                     se ((x,y) ∈ E(G_i)) então
25                         Csc ← Csc ∪ ⟨chave(x), P, chave(y)⟩
26                         grau(x) ← grau(x) − 1;   grau(y) ← grau(y) − 1
27                         se (grau(x) = 2) então
28                             F ← F ∪ {x}
29                         se (grau(y) = 2) então
30                             F ← F ∪ {y}
31                     senão                    /* criamos um novo vértice w. */
32                         Adj(x) ← Adj(x) ∪ {w};   Adj(y) ← Adj(y) ∪ {w}
33                         Adj(w) ← {x,y};   grau(w) ← 2
34                         cor(w) ← BRANCA;   chave(w) ← P
35                 senão
36                     Csc ← Csc ∪ ⟨chave(x), P, chave(y)⟩
37         para cada u ∈ V(G_i) faça
38             se (cor(u) = BRANCA) então
39                 retorna NAO
40 retorna SIM, Csc
```

O Algoritmo 5.2 atribui uma orientação cíclica a um grafo ciclicamente orientável. Para isso, utilizamos uma função booleana chamada *orienta* que atribui o valor 1 para uma orientação no sentido horário e -1 para uma orientação no sentido anti-horário.

Algoritmo 5.2: $AtribuiOrientacaoCiclica(G,B,Csc)$

Entrada: Um grafo ciclicamente orientável G e uma pilha Csc de ciclos sem corda obtida pelo Algoritmo 5.1.

Saída: Uma orientação cíclica para o grafo G.

```
1  inicialize orienta(x,y) ← 0 para todo x,y ∈ V(G)
2  enquanto (Csc ≠ ∅) faça
3      remova um elemento c = ⟨x_1,...,x_t⟩ de Csc
4      x_{t+1} ← x_1
5      i ← 1
6      enquanto (i ≤ t) faça
7          se (orienta(x_i,x_{i+1}) = 0) então
8              orienta(x_i,x_{i+1}) ← 1
9              orienta(x_{i+1},x_i ← −1)
10             i ← i + 1
11         senão
12             se (orienta(x_{i+1},x_i) = 1) então
13                 para cada j ∈ {1,...,t} faça
14                     orienta(x_j,x_{j+1}) ← −1
15                     orienta(x_{j+1},x_j ← 1)
16             senão
17                 para cada j ∈ {i+1,...,t} faça
18                     orienta(x_j,x_{j+1}) ← 1
19                     orienta(x_{j+1},x_j ← −1)
20             i ← t + 1        /* Para evitar repetição na atribuição */
```

O Algoritmo 5.2 atribui, em tempo $O(n^2)$, uma possível orientação aos ciclos sem corda de um grafo classificado como CO pelo Algoritmo 5.1. A orientação é feita de maneira independente para cada ciclo sem corda em Csc.

A orientação é realizada com base nos caminhos encontrados pelo Algoritmo 5.1 e sua atribuição é feita no sentido contrário ao que os ciclos foram encontrados, ou seja, do último ciclo para o primeiro. Isso garantirá que os ciclos não sejam orientados de forma conflitante, pois o caminho que está sendo orientado tem como base a orientação da aresta que possui em comum com o grafo anteriormente orientado.

Nos casos em que o objetivo seja apenas verificar se um grafo está ciclicamente orientado, o Algoritmo 5.1 pode ser facilmente modificado a fim de que se tenha um menor tempo de execução.

A alteração consiste em não mais utilizar os vértices de grau dois, mas sim os vértices com grau de entrada e saída iguais a um. Isso garantirá que uma menor quantidade de vértices seja analisada. Sugerimos o uso de $Adj^{-}(u)$ para os vértices na adjacência de entrada do vértice u e $Adj^{+}(u)$ para os vértices na adjacência de saída. Chamaremos a versão modificada do algoritmo de $CiclosSemCordaGrafosCOd(G)$. Lembramos que no final do algoritmo o grafo é identificado como COd se todos os ciclos sem corda são cicliclicamente orientados.

5.3 Análise dos algoritmos

A corretude do Algoritmo $CiclosSemCordaGrafosCO(G)$ é dividida em duas partes. A primeira, de reconhecimento de grafos CO, segue de Speyer [48]. O teorema abaixo completa a corretude do algoritmo.

Teorema 5.12 *Se um grafo G é CO, então o Algoritmo 5.1 encontra todos os ciclos sem corda de G.*

Prova. Suponha que G é CO. Visto que todos os componentes biconexos G_i de G são CO, podemos assumir que G é biconexo. Denote por G' o grafo obtido no final de uma iteração. No primeiro caso (Linha 24), temos que $G = G' \cup C_k$, como podemos ver na Figura 5.8(a). Todos os ciclos sem corda de G são ciclos sem corda de G' ou igual a C_k, visto que outros ciclos que contêm vértices do caminho P_k terão como corda a aresta (x,y). No segundo caso (Linha 31), temos que G' é essencialmente G, visto que identificamos um novo vértice w como P_k. Isto pode ser visto na Figura 5.9. No último caso (Linha 35), o grafo G é um ciclo no qual é claramente um ciclo sem corda, como podemos ver na Figura 5.8(c). □

Da mesma forma, o Algoritmo 4.3 encontra todos os ciclos sem corda de um grafo CO. Nos Casos 1 e 3 (ver Figura 5.8), o Algoritmo 5.1 encontra o mesmo ciclo que o Algoritmo 4.3. No Caso 2, o Algoritmo 5.1 encontra o ciclo depois que o Algoritmo 4.3. Contudo, existe uma rotulação de grau que faz que o Algoritmo 4.3 encontre os ciclos sem corda na mesma ordem que o Algoritmo 5.1.

O algoritmo para determinar todos os componentes biconexos tem complexidade de tempo $O(n+m)$, como podemos ver na Seção 2.4 e [50]. Baseado na Proposição 5.9, o algoritmo começa testando se G tem no máximo $2 \cdot n - 3$ arestas. Portanto, qualquer computação que leva tempo $O(n+m)$ leva, de fato, tempo $O(n)$.

Nosso algoritmo usa uma função booleana $cor(v)$ que atribui a cor BRANCA ou CINZA para todos os vértices. Os vértices CINZA são aqueles que já foram removidos de G e serão identificados como o novo vértice w ou comporão um novo ciclo sem corda. Se G é CO, então todos os vértices entrarão em algum momento em F e, consequentemente, serão coloridos com CINZA. O algoritmo tem $O(n)$ passos e resolve recursivamente o mesmo problema, usando DFS. O algoritmo DFS tem complexidade de tempo $O(n+m)$, como podemos ver no Capítulo 2. Portanto, o Algoritmo 5.1 tem complexidade de tempo $O(n^2)$, já que $m \in O(n)$ e calcular os componentes biconexos tem complexidade de tempo $O(n^2)$ (Seção 2.4).

CAPÍTULO 6

Companheira Quase-Cartan Positiva

A noção de matrizes quase-Cartan foi introduzida por Barot, Geiss e Zelevinsky [2]. Essas matrizes, simetrizáveis, são associadas a matrizes antissimetrizáveis. Os autores mostraram algumas propriedades dessas matrizes, do ponto de vista matemático. Decidir se existe uma matriz companheira quase-Cartan positiva é parte de um critério proposto pelos autores para decidir se uma álgebra *cluster* é de tipo finito (tem um número finito de variáveis *cluster*), isto é, a matriz associada tem companheira quase-Cartan positiva e o grafo associado é ciclicamente orientado. Pelo critério de Sylvester [8], uma matriz simétrica é positiva se todos os menores principais líderes são determinante positivos. Por definição, uma matriz simetrizável é positiva se sua simetrizada também é. Mostramos na Proposição 2.27 que uma matriz simetrizável é positiva se todos os menores principais também o são.

O capítulo está organizado da seguinte maneira: na Seção 6.1 mostramos algumas características das matrizes companheiras quase-Cartan positivas, estreitamos alguns limites propostos por [2] e propomos maneiras mais fáceis de encontrá-las; na Seção 6.2 conjecturamos que o problema de decidir se existe uma companheira quase-Cartan positiva para grafos gerais é NP-completo e mostramos que ele está na classe de problemas NP; na Seção 6.3 mostramos algumas características das companheiras quase-Cartan associadas a grafos ciclicamente orientáveis e apresentamos um algoritmo para decidir se existe uma companheira quase-Cartan positiva, neste caso; por fim, na Seção 6.4 provamos que decidir se uma álgebra *cluster* é de tipo finito pertence à classe de problemas polinomiais, apresentando um algoritmo com complexidade $O(n^4)$ para tal propósito, onde n é a ordem da matriz relacionada.

6.1 Companheira quase-Cartan positiva

Nesta seção, estudamos as matrizes companheiras quase-Cartan positivas do ponto de vista matemático e computacional e obtemos algumas propriedades e caracterizações. O seguinte teorema nos permite pensar indutivamente no problema de encontrar uma companheira quase-Cartan positiva.

Teorema 6.1 *Uma matriz antissimetrizável B tem uma companheira quase-Cartan positiva se e somente se qualquer submatriz principal de B tem uma companheira quase-Cartan positiva.*

Prova. ($\Rightarrow$) Seja C uma companheira quase-Cartan positiva de B. Por indução, precisamos apenas observar que $C_{[ii]}$ é uma companheira quase-Cartan positiva de $B_{[ii]}$. Pelo Lema 2.30, $C_{[ii]}$ é simetrizável.

Segue do Teorema 2.28 que $C_{[ii]}$ é positiva. Portanto, $C_{[ii]}$ é uma companheira quase-Cartan positiva de $B_{[ii]}$.

($\Leftarrow$) Segue do fato que a matriz B é uma submatriz principal dela mesma. □

O lema seguinte fornece uma condição necessária para uma companheira quase-Cartan ser positiva. Apresentamos as principais ideias da prova original, visto que elas nos serão úteis mais adiante.

Lema 6.2 (Barot, Geiss e Zelevinsky [2]) *Se C é uma matriz quase-Cartan positiva, então*

(a) $0 \leq c_{ij} \cdot c_{ji} \leq 3$ *para quaisquer i, j tal que $i \neq j$;*
(b) $c_{ik} \cdot c_{kj} \cdot c_{ji} = c_{ki} \cdot c_{jk} \cdot c_{ij} \geq 0$ *para quaisquer i, j, k distintos 2 a 2.*

Prova.

(a) Seja $C' = \begin{pmatrix} 2 & c_{ij} \\ c_{ji} & 2 \end{pmatrix}$ uma submatriz principal de C. Visto que C é uma matriz simetrizável, ela é simétrica pelos sinais e $c_{ij} \cdot c_{ji} \geq 0$. Como C é positiva, temos que $det(C') = 4 - c_{ij} \cdot c_{ji} > 0$. Portanto, $c_{ij} \cdot c_{ji} \leq 3$, visto que as entradas são inteiras.

(b) Seja $c_{ik} \cdot c_{kj} \cdot c_{ji} \neq 0$. Como C é simetrizável, temos que $c_{ki} \cdot c_{jk} \cdot c_{ij} = c_{ik} \cdot c_{kj} \cdot c_{ji}$. A condição de positividade para os menores principais de C de ordem 3 nas linhas e colunas i, j, k pode ser reescrita como:

$$c_{ik} \cdot c_{kj} \cdot c_{ji} > c_{ij} \cdot c_{ji} + c_{ik} \cdot c_{ki} + c_{jk} \cdot c_{kj} - 4. \tag{6-1}$$

Visto que $c_{ik} \cdot c_{kj} \cdot c_{ji} \neq 0$, temos que $|c_{st}| \geq 1$ e assim $c_{st} \cdot c_{ts} \geq 1$ para $(s,t) \in \{(i,k),(k,j),(j,i)\}$. Portanto, $c_{ik} \cdot c_{kj} \cdot c_{ji} > 3 - 4 = -1$. Isto nos leva à conclusão.

□

Exemplo 6.3 *Sejam $a, b \in \mathbb{Z}$ e $C = \begin{pmatrix} 2 & a \\ b & 2 \end{pmatrix}$. A matriz C é quase-Cartan se e somente se $a \cdot b \geq 0$. Neste caso, a condição (a) é equivalente à matriz C ser positiva. Logo, C é quase-Cartan positiva se e somente se $(a,b) \in \{(0,0),(1,1),(-1,-1),(1,2),(-1,-2),(2,1),(-2,-1)\}$.*

Para uma matriz quase-Cartan de ordem 3 há três submatrizes principais líderes, que são: a submatriz de ordem 1, que obviamente tem determinante positivo; a submatriz de ordem 2, que é positiva quando $0 \leq c_{ij} \cdot c_{ji} \leq 3$; e, finalmente, a submatriz de ordem 3 que é ela mesma. Logo, para uma matriz de ordem 3, precisamos analisar somente a matriz $C^+ = (c^+_{ij})$ definida por $c^+_{ij} = |b_{ij}|$ para $i \neq j$ e $c^+_{ii} = 2$.

Proposição 6.4 *Seja B uma matriz antissimetrizável de ordem 3. B tem uma companheira quase-Cartan positiva se e somente se a matriz $C^+ = (c^+_{ij})$ é positiva.*

Prova. ($\Rightarrow$) Suponha que B possui uma matriz companheira quase-Cartan positiva C. Logo, $det(C^+) = 8 - 2 \cdot c^+_{jk} \cdot c^+_{kj} - 2 \cdot c^+_{ij} \cdot c^+_{ji} - 2 \cdot c^+_{ki} \cdot c^+_{ik} + 2 \cdot c^+_{ij} \cdot c^+_{jk} \cdot c^+_{ki} = 8 - 2 \cdot c_{jk} \cdot c_{kj} - 2 \cdot c_{ij} \cdot c_{ji} - 2 \cdot c_{ki} \cdot c_{ik} + 2 \cdot |c_{ij} \cdot c_{jk} \cdot c_{ki}| \geq det(C) > 0$. Portanto, C^+ também é uma companheira quase-Cartan positiva de B.

($\Leftarrow$) Visto que C é uma matriz quase-Cartan, segue do Teorema 2.16 que C^+ é uma companheira quase-Cartan de B. □

Nos resultados a seguir, conseguimos um limite mais estreito para as condições do Lema 6.2. A seguir, o caso para $n = 3$.

Lema 6.5 *Seja C uma matriz quase-Cartan positiva de ordem 3.*

(a) Se C é conexa, então $0 \leq c_{ij} \cdot c_{ji} \leq 2$ para todo i, j tal que $i \neq j$.
(b) $0 \leq c_{ik} \cdot c_{kj} \cdot c_{ji} \leq 2$ para quaisquer i, j, k distintos 2 a 2.

Prova. A Proposição 6.4 mostra que $det(C^+) > det(C)$. Sendo assim, C^+ é positiva. Por razões de simplicidade, consideramos que $C = C^+$.

(a) Pelo Lema 6.2, $0 \leq c_{ij} \cdot c_{ji} \leq 3$. Logo, $|c_{ij}| \leq 3$. Suponha, sem perda de generalidade, que $|c_{12}| \cdot |c_{21}| = 3$. Por simetria, suponha que $c_{12} = 3$ e $c_{21} = 1$. Logo, $det(C) = 8 + 2 \cdot c_{12} \cdot c_{23} \cdot c_{31} - 2 \cdot c_{12} \cdot c_{21} - 2 \cdot c_{13} \cdot c_{31} - 2 \cdot c_{23} \cdot c_{32} = 2 + 6 \cdot c_{23} \cdot c_{31} - 2 \cdot c_{13} \cdot c_{31} - 2 \cdot c_{23} \cdot c_{32} > 0$.
Visto que C é conexa, $c_{23} \neq 0$ ou $c_{31} \neq 0$. A positividade de C implica em ambos: $c_{23} \neq 0$ e $c_{31} \neq 0$. Como C é simetrizável, $c_{32} \neq 0$ e $c_{13} \neq 0$.
Como $c_{12} \cdot c_{23} \cdot c_{31} = c_{21} \cdot c_{32} \cdot c_{13}$, $3 \cdot c_{23} \cdot c_{31} = c_{32} \cdot c_{13}$ e $c_{32} = 3$ ou $c_{13} = 3$. Por simetria, suponha que $c_{32} = 3$. Assim, $c_{23} = 1$ visto que $c^2_{13} = c_{13} \cdot c_{31} \leq 3$. Concluímos que $c_{31} = c_{13} = 1$.
Por outro lado, $det(C) = 2 + 6 \cdot c_{23} \cdot c_{31} - 2 \cdot c_{13} \cdot c_{31} - 2 \cdot c_{23} \cdot c_{32} = 2 + 6 - 2 - 6 = 0$.
Isto nos leva à contradição para a positividade de C. Portanto, $c_{ij} \cdot c_{ji} \leq 2$, para todo i, j.

(b) Suponha que $c_{12} \cdot c_{23} \cdot c_{31} = c_{13} \cdot c_{32} \cdot c_{21} \geq 3$ e que $c_{12} \cdot c_{21} = 3$. Pelo item anterior, usando a sua contrapositiva, temos que $c_{23} = 0$ e $c_{31} = 0$, uma contradição com a hipótese. Isto implica que $c_{ij} \cdot c_{ji} \leq 2$ para todo i, j e que $c_{12} \cdot c_{23} \cdot c_{31} \geq 4$. Podemos supor, sem perda de generalidade, que $c_{12} = 2$ e $c_{23} = 2$. Então, $c_{21} = 1$ e $c_{32} = 1$. Logo, $c_{13} = 4$ e $c_{31} \neq 0$, uma contradição ao Lema 6.2. Portanto, $0 \leq c_{12} \cdot c_{23} \cdot c_{31} \leq 2$.

□

Generalizamos o lema anterior para qualquer $n \geq 3$.

Teorema 6.6 *Seja C uma matriz quase-Cartan positiva de ordem n, com* $n \geq 3$.

(a) Se C é conexa, então $0 \leq c_{ij} \cdot c_{ji} \leq 2$ *para todo* i, j *tal que* $i \neq j$.
(b) $0 \leq c_{ik} \cdot c_{kj} \cdot c_{ji} \leq 2$ *para quaisquer* i, j, k *distintos 2 a 2.*

Prova.

(a) Suponha que existem i, j tal que $c_{ij} \cdot c_{ji} \geq 3$. Sem perda de generalidade, suponha que $i = 1$ e $j = 2$. Visto que $n \geq 3$, consideramos $k \geq 3$ e a submatriz principal C' de C, formada de linhas e colunas $1, 2$ e k. Pelo Lema 6.5, C' é desconexa. Isto implica que $c_{1k} = c_{k2} = 0$ e assim $c_{k1} = c_{2k} = 0$. Visto que isto é verdadeiro para todo $k \geq 3$, temos que C é desconexa, uma contradição.
(b) Segue do Lema 6.5, pela consideração da submatriz principal de C, formada pelas linhas e colunas i, j e k.

□

No próximo caso, precisamos novamente analisar apenas a matriz C^+.

Teorema 6.7 *Seja B uma matriz antissimetrizável tal que* $b_{ij} \neq 0$ *para todo* i, j. *B possui uma companheira quase-Cartan positiva C se e somente se* C^+ *é positiva.*

Prova. ($\Rightarrow$) Suponha que existe uma companheira quase-Cartan positiva C. Iremos mostrar por indução que C^+ é positiva. Se C é uma matriz de ordem 2, o resultado é claramente obtido. Primeiro, mostramos que $det(C^+) > 0$. Seja $x_{ij} = sgn(c_{ij})$. Devido ao fato da matriz ser quase-Cartan, $x_{ji} = x_{ij}$ e $x_{ii} = 1$. Como $c_{ij} \neq 0$, temos que $sgn(c_{ij}) \neq 0$. Definimos $x_i = x_{1i}$ para todo i. Nós iremos mostrar, por indução em i, que $x_{ij} = x_i \cdot x_j$, para todo i, j. Claramente, $x_{ii} = x_i^2$.

Visto que $x_1 = 1$, é fácil ver que $x_{1i} = x_1 \cdot x_i$. Suponha que $x_{kj} = x_k \cdot x_j$ para todo $k < i$ e para todo j. Pelo Teorema 6.6, $c_{ij} \cdot c_{jk} \cdot c_{ki} > 0$. Logo, $x_{ij} \cdot x_{jk} \cdot x_{ki} = 1$. Portanto, $x_{ij} = x_{jk} \cdot x_{ki} = x_{kj} \cdot x_{ki} = x_k \cdot x_j \cdot x_k \cdot x_i = x_i \cdot x_j$.

Visto que $x_{ij} = x_i \cdot x_j$ e $c_{ij} = x_{ij} \cdot c^+_{ij}$, temos que $C = XC^+X$, onde X é a matriz diagonal (x_i).

Segue que $det(C) = det(X) \cdot det(C^+) \cdot det(X) = det(C^+)$. Por indução na dimensão de C^+, temos que todos os menores principais líderes de C^+ são positivos.

($\Leftarrow$) Segue do fato que C é uma matriz quase-Cartan e da Proposição 2.16 que C^+ é uma companheira quase-Cartan de B. □

Neste caso $G(B) = K_n$.

O Teorema 6.7 não é válido para todas as matrizes antissimetrizáveis, como podemos ver no exemplo seguinte.

Exemplo 6.8 *Sejam* $B = \begin{pmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ 0 & -1 & -1 & 0 \end{pmatrix}$, $C = \begin{pmatrix} 2 & -1 & 1 & 0 \\ -1 & 2 & 0 & 1 \\ 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 2 \end{pmatrix}$ *e* $C^+ = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \\ 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 2 \end{pmatrix}$. *A matriz companheira quase-Cartan* C^+ *não é positiva, mas C é.*

Para o caso da proposição seguinte, sempre temos uma companheira quase-Cartan positiva.

Proposição 6.9 *Seja B a matriz antissimétrica tal que* $|b_{ij}| = 1$ *para todo* $i \neq j$*, então B tem uma companheira quase-Cartan positiva.*

Prova. Se B é uma matriz de ordem n, então $det(C^+) = n+1$. Claramente, C^+ é positiva. □

Como vimos na Seção 2.6, verificar todas as companheiras quase-Cartan para encontrar uma positiva é um procedimento de complexidade exponencial. Entretanto, Barot, Geiss e Zelevinsky [2] mostraram uma condição de sinais que simplifica o teste da existência de uma companheira quase-Cartan positiva sem ter que atribuir todos os possíveis sinais.

Definição 6.10 (Condição de sinais – Barot, Geiss e Zelevinsky [2]) *Seja B uma matriz antissimetrizável. Uma companheira quase-Cartan C de B satisfaz a condição de sinais se, para todo ciclo sem corda* C_k *em* $G(B)$*, o produto* $\prod_{(i,j)\in C_k} (-c_{ij})$ *sobre todas as arestas de* C_k *é negativo.*

A proposição seguinte mostra que é suficiente verificar a companheira quase-Cartan que satisfaz a condição de sinais para determinar a positividade da matriz.

Proposição 6.11 (Barot, Geiss e Zelevinsky [2]) *Para ser positiva, uma companheira quase-Cartan C de uma matriz antissimetrizável B deve satisfazer a condição de sinais.*

Uma matriz quase-Cartan que satisfaz a condição de sinais não é necessariamente positiva. No Exemplo 2.19, todas as companheiras quase-Cartan satisfazem a condição de sinais, mas nenhuma é positiva. Por outro lado, como podemos ver no exemplo seguinte, podemos ter mais de uma companheira quase-Cartan positiva.

Exemplo 6.12 *Seja* $B = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$ *uma matriz antissimetrizável. Temos que* $\begin{pmatrix} 2 & -1 & 1 \\ -1 & 2 & -1 \\ 1 & -1 & 2 \end{pmatrix}$, $\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$, $\begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & 1 \\ -1 & 1 & 2 \end{pmatrix}$ *e* $\begin{pmatrix} 2 & 1 & -1 \\ 1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix}$ *são companheiras quase-Cartan positivas de B.*

Na Proposição 6.4 e no Teorema 6.7 a matriz C^+ sempre satisfaz a condição de sinais.

É necessário que as duas condições no critério (d) do Teorema 3.17 sejam satisfeitas para termos uma álgebra *cluster* de tipo finito. De fato, o caso de ordem 4 da Proposição 6.9 corresponde ao grafo K_4, que não é ciclicamente orientável e no Exemplo 2.19, não existe companheira quase-Cartan positiva mas o grafo é trivialmente ciclicamente orientável, já que é uma árvore.

6.2 A existência de companheira quase-Cartan positiva pertence à classe NP

Nesta seção, desenvolvemos dois algoritmos. O primeiro decide, com complexidade de tempo $O(n^4)$, se uma matriz simetrizável é positiva ou não. Ele é usado para provar que o problema de decidir se uma matriz antissimetrizável tem uma companheira quase-Cartan positiva está na classe NP. O segundo é um algoritmo de complexidade de tempo exponencial para encontrar uma companheira quase-Cartan, se existir, de uma matriz antissimetrizável dada. Dado o caráter combinatório da busca pela companheira quase-Cartan positiva, para grafos gerais temos a conjectura seguinte.

Conjectura 1 *O problema de decidir se existe uma companheira quase-Cartan positiva pertence à classe NP-completa.*

Apresentamos a seguir o Algoritmo 6.1, baseado na Proposição 2.27, que tem complexidade de tempo $O(n^4)$ e decide se uma matriz C é positiva. Ele pode ser usado como verificador para o problema da conjectura acima. Logo, este problema pertence à classe de problemas NP. Para mais informações a respeito das classes P, NP e NP-completa, veja o Apêndice A e [10, 46].

Algoritmo 6.1: *Positiva*(C)

Entrada: Uma matriz simetrizável C.

Saída: A resposta se a matriz é positiva ou não.

```
para cada submatriz principal líder C' de C
faça
    se (det(C') ≤ 0) então
        retorna NÃO
retorna SIM
```

Para a conjectura anterior, elaboramos um algoritmo de complexidade de tempo exponencial que encontra uma companheira quase-Cartan positiva para uma matriz antissimetrizável B.

Algoritmo 6.2: *CompanheiraQuaseCartan*(B)

Entrada: Uma matriz antissimetrizável B.

Saída: Uma companheira quase-Cartan positiva C, se existir.

```
para cada i ∈ {1,...,n} faça
    c_ii ← 2
para cada x ∈ {(x_ij) | x_ij ∈ {−1,1} e i < j} faça
    para cada i ∈ {1,...,n} faça
        para cada j ∈ {i+1,...,n} faça
            c_ij ← x_ij · |b_ij|
            c_ji ← x_ij · |b_ji|
    se (Positiva(C)) então
        retorna C
retorna "Não existe companheira quase-Cartan positiva de B"
```

6.3 Companheira quase-Cartan positiva associada a grafos ciclicamente orientáveis

Nesta seção, mostramos que encontrar uma companheira quase-Cartan positiva associada a um grafo ciclicamente orientado pertence à classe P de problemas polino-

miais. Neste caso, a matriz antissimetrizável sempre tem pelo menos uma companheira quase-Cartan que satisfaz a condição de sinais.

Proposição 6.13 (Barot, Geiss e Zelevinsky [2]) *Seja B uma matriz antissimetrizável. Se $G(B)$ é ciclicamente orientável, então B tem uma companheira quase-Cartan (não necessariamente positiva) que satisfaz a condição de sinais da Proposição 6.10; além disso, tal companheira quase-Cartan é única para as mudanças simultâneas de sinais nas linhas e colunas.*

Se $G(B)$ não é ciclicamente orientável (CO), então não necessariamente existe uma companheira quase-Cartan satisfazendo a condição de sinais. Podemos ver isto claramente na Figura 5.4. No caso em que o grafo associado é CO, é suficiente verificar apenas uma companheira quase-Cartan que satisfaça a condição de sinais.

Proposição 6.14 *Seja C uma companheira quase-Cartan de uma matriz antissimetrizável B que satisfaz a condição de sinais. Se $G(B)$ é ciclicamente orientável, então B tem uma companheira quase-Cartan positiva se e somente se C é positiva.*

Prova. Suponha que $G(B)$ seja ciclicamente orientável. Claramente, se C é positiva, então B tem uma companheira quase-Cartan positiva. Por outro lado, se B tem uma companheira quase-Cartan positiva, digamos C', então pela Proposição 6.11, C' satisfaz a condição de sinais. Pela Proposição 6.13, C é obtida de C' pelas mudanças simultâneas de sinais nas linhas e colunas. Portanto, existe uma matriz diagonal $X = (x_{ij})$ com $x_{ii} \in \{-1,1\}$ tal que $C = XC'X$. Visto que $det(C) = det(X) \cdot det(C') \cdot det(X)$ e $det(X) \in \{-1,1\}$. Temos que $det(C) = det(C')$. Da mesma forma, todos os menores principais de C são os menores principais de C'. O resultado segue do Teorema 2.28. □

Devido à proposição anterior, elaboramos o Algoritmo 6.3 (*Companheira Positiva*(G,B,Csc)) para grafos CO que verifica se uma matriz antissimetrizável tem uma companheira quase-Cartan positiva. Ele é baseado na ideia de encontrar uma companheira quase-Cartan C que satisfaz a condição de sinais e verificar a positividade de C.

No Algoritmo 6.3 a pilha *Csc* contém todos os ciclos sem corda obtidos pelo Algoritmo 5.1 (*CiclosSemCordaCO*(G)) e o conjunto T contém todas as pontes de G, ou seja, todos os componentes biconexos simples de G.

Algoritmo 6.3: *CompanheiraPositiva*(G,B,Csc)

Entrada: Um grafo ciclicamente orientado G, uma matriz antissimetrizável B associada a G, uma pilha Csc de ciclos sem corda e um conjunto T de componentes biconexos simples de G.

Saída: A resposta se existe ou não uma companheira quase-Cartan positiva de B.

```
1  inicialize sgn(x_i, x_j) ← 0 para todo i, j ∈ {1,2,...,n}
2  inicialize sgn(aresta) ← 1 para todas arestas em T
3  enquanto (Csc ≠ ∅) faça
4      remova um elemento c de Csc                    /* c = ⟨x_1,...,x_t⟩ */
5      x_{t+1} ← x_1
6      I ← 0
7      prod ← 1
8      para cada i ∈ {1,...,t} faça
9          se (sgn(x_i, x_{i+1}) ≠ 0) então
10             prod ← −prod · sgn(x_i, x_{i+1})
11         senão
12             se (I = 0) então
13                 I ← i
14             senão
15                 sgn(x_i, x_{i+1}) ← −1
16     sgn(x_I, x_{I+1}) ← prod
17 inicialize c_ij ← |b_ij| · sgn(x_i, x_j) para todo i, j ∈ {1,2,...,n}
18 inicialize c_ii ← 2 para todo i ∈ {1,2,...,n}
19 se (Positiva(C)) então
20     retorna SIM
21 senão
22     retorna NÃO
```

Vimos no Corolário 5.11 que a cardinalidade do conjunto Csc de ciclos sem corda é no máximo n. Portanto, a complexidade de tempo do Algoritmo 6.3 é $O(n^4)$, devido à Linha 19. O restante do algoritmo é $O(n^2)$. Para provar a corretude do Algoritmo 6.3, primeiro mostramos que a função de sinais sgn é bem definida.

Proposição 6.15 *A função sgn é bem definida, isto é,* $sgn(aresta) \in \{1,-1\}$ *para toda* $aresta \in E(G)$.

Prova. Os números de linha nesta prova se referem às linhas do Algoritmo 6.3. Primeiro, provamos que na Linha 16 temos $I \neq 0$. Podemos assumir que G é um grafo biconexo, visto que o algoritmo atua em cada componente biconexo. O primeiro ciclo a ser

processado é um ciclo sem sinal de aresta definido. Portanto, a primeira aresta não satisfaz a condição da Linha 9 e executa a Linha 13 ($I = 1$). Para quaisquer outros ciclos C_k, temos que $G = G' \cup C_k$ e somente os sinais das arestas de G' são definidos. Portanto, a primeira ou segunda aresta de C_k não tem sinal definido ($I = 1$ ou $I = 2$).

As arestas de G são pontes, isto é, pertencem a T ou fazem parte de um ciclo sem corda. Se G é uma ponte, então o sinal da aresta recebe 1 na Linha 2. Se não, recebe -1 na Linha 15 ou "prod" na Linha 16. Portanto, a função *sgn* é bem definida. □

Mostramos em seguida que a construção de C leva a uma companheira quase-Cartan que satisfaz a condição de sinais.

Teorema 6.16 *A matriz C definida abaixo satisfaz a condição de sinais.*

$$c_{ij} = \begin{cases} 2, & \text{se } i = j; \\ sgn(x_i, x_j) \cdot |b_{ij}|, & \text{caso contrário.} \end{cases}$$

Prova. Relembre que o conjunto *Csc* é composto de todos os ciclos sem corda de G. Segue da demonstração da Proposição 6.15 que o sinal do produto $\prod_{(i,j) \in C_k} (-c_{ij})$ de cada ciclo C_k é igual a $-prod^2$. □

Teorema 6.17 *O Algoritmo 6.3 é correto.*

Prova. Segue do Teorema 6.16 e da Proposição 6.14. □

Exemplo 6.18 *Seja B uma matriz antissimetrizável de ordem n, com as entradas acima da diagonal principal dadas por:*

$$b_{ij} = \begin{cases} -1, & \text{se } j - i = 1, \\ 1, & \text{se } j - i = 2, \\ 0, & \text{se } j - i > 2. \end{cases}$$

O grafo $G(B)$ tem $n-2$ ciclos sem corda de tamanho 3. Eles são formados por todas as triplas de índices consecutivos e todos são ciclicamente orientados.

Apresentamos um exemplo com $n = 4$: $B = \begin{pmatrix} 0 & -1 & 1 & 0 \\ 1 & 0 & -1 & 1 \\ -1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \end{pmatrix}$.

O grafo da matriz antissimetrizável B é apresentado na Figura 6.1. Os ciclos sem corda são $\langle 2,4,3\rangle$ *e* $\langle 2,1,3\rangle$*, que são ciclicamente orientados. Além disso, eles cumprem a condição de sinais.*

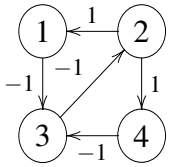


Figura 6.1: *Exemplo de um grafo com* 2 *ciclos sem corda e com uma atribuição de sinais dada pelo algoritmo.*

Veja a matriz companheira quase-Cartan positiva associada à atribuição de sinais da figura anterior:

$$C = \begin{pmatrix} 2 & 1 & -1 & 0 \\ 1 & 2 & -1 & 1 \\ -1 & -1 & 2 & -1 \\ 0 & 1 & -1 & 2 \end{pmatrix}$$

6.4 Reconhecimento polinomial de álgebras cluster de tipo finito

Uma vez que o Algoritmo 5.1 ($CiclosSemCordaGrafosCO(G)$) modificado para a versão $CiclosSemCordaGrafosCOd(G)$ verifica se um grafo está ou não orientado de forma cíclica e o Algoritmo 6.3 ($CompanheiraPositiva(G,B,S)$) verifica se uma matriz tem ou não uma companheira quase-Cartan positiva, ambos com complexidade de tempo polinomial, segue do Teorema 3.17 que decidir se uma álgebra *cluster* é de tipo finito pertence à classe *P* de problemas polinomiais.

Algoritmo 6.4: $AlgebraClusterTipoFinito(B,G)$

Entrada: Uma matriz antissimétrica B, que define uma álgebra *cluster* $\mathcal{A}(\mathcal{B})$, e seu respectivo grafo G.

Saída: A resposta se a álgebra *cluster* é de tipo finito ou não.

```
1 COd,Csc ← CiclosSemCordaGrafosCOd(G)
2 se (COd = SIM) então
3 |   CompanheiraPositiva(G,B,S)
4 senão
5 |   retorna NÃO
```

Baseado nos algoritmos anteriores, temos que a complexidade de tempo do Algoritmo 6.4 é $O(n^4)$. A corretude do Algoritmo 6.4 segue diretamente da corretude dos Algoritmos 5.1 e 6.3.

CAPÍTULO 7

Conclusões

Neste trabalho, investigamos o problema de decidir se uma álgebra *cluster* é de tipo finito. Usando apenas a definição, o problema é difícil de ser resolvido, visto que devem ser realizadas todas as mutações na álgebra até que as variáveis *cluster* comecem a se repetir. Isso pode levar a um processo infinito, caso a álgebra *cluster* não seja de tipo finito. Barot, Geiss e Zelevinsky [2] propuseram o seguinte critério: uma álgebra *cluster* $\mathcal{A}(B)$ é de tipo finito se e somente se todos os ciclos sem corda no grafo associado à matriz antissimetrizável B é ciclicamente orientado e existe uma companheira quase-Cartan positiva de B.

Devido à primeira parte do critério, surgiu o interesse de enumerar todos os ciclos sem corda de um grafo qualquer. No Capítulo 4, apresentamos um algoritmo que realiza esta tarefa. Deste capítulo, elaboramos o artigo intitulado "Efficient enumeration of chordless cycles" [14]. Também desenvolvemos uma versão paralela intitulada "A GPU-based parallel algorithm for enumerating all chordless cycles in graphs" [32].

Utilizamos as propriedades de grafos ciclicamente orientáveis propostas por Gurvich [26] e Speyer [45] para elaborar um algoritmo que detecta se o grafo é ciclicamente orientável. Em seguida, provamos que a quantidade de ciclos sem corda em um grafo ciclicamente orientável é polinomial e que o algoritmo anterior os enumera. Estes resultados estão no Capítulo 5 e em "Polynomial enumeration of chordless cycles on cyclically orientable graphs" [6].

Para a segunda parte do critério, sobre decidir se uma matriz tem uma companheira quase-Cartan positiva, trabalhamos em algumas propriedades das matrizes simetrizáveis e antissimetrizáveis, em um algoritmo para encontrar a matriz simetrizante e um algoritmo para verificar se uma matriz é positiva. Esses resultados podem ser encontrados nos Capítulos 2 e 6 e no artigo intitulado "Algorithms and properties for positive symmetrizable matrices" [13].

Dada uma matriz antissimetrizável B qualquer, não existe apenas uma companheira quase-Cartan positiva, mas encontrá-la é muito difícil visto que deve-se tentar todas as combinações de sinais possíveis e verificar se os menores principais são positivos. De fato, a quantidade de companheiras quase-Cartan é exponencial. Apesar disso, mostramos

que podemos fazer isso em tempo polinomial para grafos ciclicamente orientáveis.

Por fim, no Capítulo 6, mostramos como decidir se uma álgebra *cluster* é de tipo finito em tempo polinomial. Este resultado está no artigo intitulado “Polynomial recognition of cluster algebras of finite type” [12].

Durante o doutorado, na disciplina Computação Paralela, foi estudado o problema de encontrar o fecho transitivo de um grafo. Este resultado pode ser verificado no artigo intitulado “Implementações paralelas para fecho transitivo” [11].

Existem alguns problemas em aberto que podem ser explorados após o término do doutorado, como a nossa conjectura de que o problema de encontrar uma companheira quase-Cartan C para uma matriz antissimetrizável B pertence à classe de problemas NP-completa. Nós provamos que o problema pertence à classe de problemas NP e reduzimos o espaço de busca devido às características que fornecemos para as companheiras quase-Cartan positivas.

Outro problema interessante é estudar grafos ciclicamente orientáveis e a quantidade de ciclos sem corda em classes de grafos específicas, tais como grafos planares e produtos.

Referências Bibliográficas

[1] ASSEM, I.; SIMSON, D.; SKOWROŃSKI, A. **Elements of representation theory of associative algebras**, volume 65. London Mathematical Society Student Texts, Cambridge University Press, Techniques of Representation Theory edition, 2006.

[2] BAROT, M.; GEISS, C.; ZELEVINSKY, A. **Cluster algebras of finite type and positive symmetrizable matrices**. *Journal of the London Mathematical Society*, 73:545–564, 2006.

[3] BIRMELÉ, E.; FERREIRA, R.; GROSSI, R.; MARINO, A.; PISANTI, N.; RIZZI, R.; SACOMOTO, G. **Optimal listing of cycles and st-paths in undirected graphs**. In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, p. 1884–1896. SIAM, 2013.

[4] BISDORFF, R. **On enumerating chordless circuits in directed graphs**. Disponível em: `http://sma.uni.lu/bisdorff/ChordlessCircuits/documents/chordlessCircuits.pdf`. Acesso em: 10 ago. 2015, 2010.

[5] CASTONGUAY, D.; NOVOA, C. **Uma Introdução à Álgebra Cluster**. *Workshop Sul Americano de Representações de Álgebras*, 2008.

[6] CASTONGUAY, D.; DIAS, E. S. **Polynomial enumeration of chordless cycles on cyclically orientable graphs**. *arXiv*, Disponível em: `http://arxiv.org/abs/1505.02829`. Submetido. Acesso em: 10 ago. 2015, 2015.

[7] CHANDRASEKHARAN, N.; LASKSHMANAN, V.; MEDIDI, M. **Efficient parallel algorithms for finding chordless cycles in graphs**. *Parallel Processing Letters*, 3(2):165–170, 1993.

[8] CHEN, W.-K. **Theory and design of broadband matching networks**. Pergamon Press Ltd., 1976.

[9] COLBOURN, C.; MCKAY, B. **A corretion to Colbourn's paper on the complexity of matrix symmetrizability**. *Information Processing Letters*, 11:96–97, 1980.

[10] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos - teoria e prática**. Campus, 3 edition, 2012.

[11] DE AQUINO GOMES, R.; DIAS, E. S.; SANTANA, M. R. C.; CÁCERES, E. N.; MARTINS, W. S. **Implementações paralelas para fecho transitivo**. *CLAIO-SOBRAPO*, p. 4058–4069, 2012.

[12] DIAS, E. S.; CASTONGUAY, D. **Polynomial recognition of cluster algebras of finite type**. *arXiv*, Disponível em: `http://arxiv.org/abs/1507.03844`. Submetido. Acesso em: 10 ago. 2015, 2015.

[13] DIAS, E. S.; CASTONGUAY, D.; DOURADO, M. C. **Algorithms and properties for positive symmetrizable matrices**. *arXiv*, Disponível em: `http://arxiv.org/abs/1503.03468`. Submetido. Acesso em: 10 ago. 2015, 2015.

[14] DIAS, E. S.; CASTONGUAY, D.; LONGO, H. J.; JRADI, W. A. R. **Efficient enumeration of chordless cycles**. *arXiv*, Disponível em: `http://arxiv.org/abs/1309.1051`. Submetido. Acesso em: 10 ago. 2015, 2013.

[15] DOGRUSÖZ, U.; KRISHNAMOORTHY, M. **Cycle vector space algorithms for enumerating all cycles of a planar graph**. *Journal of Parallel Algorithms and Applications*, p. 1–14, 1995.

[16] FOMIN, S.; ZELEVINSKY, A. **Cluster algebras I: foundations**. *Journal of the American Mathematical Society*, 15:497–529, 2002.

[17] FOMIN, S.; ZELEVINSKY, A. **Cluster algebras II : finite type classification**. *Inventiones Mathematicae*, 154:63–121, 2003.

[18] FOMIN, S. **Cluster algebras portal**, 2015. Disponível em: `http://www.math.lsa.umich.edu/~fomin/cluster.html`. Acesso em: 10 ago. 2015.

[19] GAREY, M. R.; JOHNSON, D. S. **Computers and intractability - a guide to the theory of NP-completeness**, volume 1. Freeman, 1979.

[20] GEKHTMAN, M.; SHAPIRO, M.; VAINSHTEIN, A. **Cluster algebras and poisson geometry**, volume 167. American Mathematical Society – Mathematical Surveys and Monographs, 2010.

[21] GENTLE, J. E. **Matrix Algebra: Theory, Computations and Applications in Statistics**, volume Springer Texts in Statistics. Springer, 2010.

[22] GLEISS, P. M. **Short cycles: minimum cycle bases of graphs from chemistry and biochemistry**. PhD thesis, Universität Wien, 2001.

[23] GOLUMBIC, M. C. **Algorithmic graph theory and perfect graphs**, volume Annals of Discrete Mathematics, Vol 57. North-Holland Publishing Co., Amsterdam, The Netherlands, 2 edition, 2004.

[24] GONÇALVES, A. **Introdução à Álgebra**, volume Coleção Projeto Euclides. Associação Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, 5 edition, 2013.

[25] GRATZ, S.; GRABOWSKI, J. **Cluster algebras of infinite rank**. *Journal of the London Mathematical Society*, 2:337–363, 2014.

[26] GURVICH, V. **On cyclically orientable graphs**. *Discrete Mathematics*, 308:129–135, 2008.

[27] HAAS, R.; HOFFMANN, M. **Chordless paths through three vertices**. *Theoretical Computer Science*, 351:360–371, 2006.

[28] HARISH, P.; NARAYANAN, P. **Accelerating large graph algorithms on the GPU using CUDA**. In: *Proceedings of the 14th International Conference on High Performance Computing, HiPC' 07*, p. 197–208. Springer-Verlag, 2007.

[29] HAYWARD, R. **Weakly triangulated graphs**. *Journal of Combinatorial Theory, Series B*, 39:200–209, 1985.

[30] HAYWARD, R. **Two classes of perfect graphs**. PhD thesis, School of Computer Science, McGill Univ., 1986.

[31] JOHNSON, D. B. **Finding All the Elementary Circuits of a Directed Graph**. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[32] JRADI, W. A. R.; DIAS, E. S.; CASTONGUAY, D.; LONGO, H. J.; DO NASCIMENTO, H. A. D. **A GPU-based parallel algorithm for enumerating all chordless cycles in graphs**. *arXiv*, Disponível em: `http://arxiv.org/abs/1410.4876v2`. Submetido. Acesso em: 10 ago. 2015, 2014.

[33] KAC, V. G. **Infinite dimensional Lie algebras**. Cambridge University Press, 3 edition, 1990.

[34] KAPOOR, S.; RAMESH, H. **An algorithm for enumerating all spanning trees of a directed graph**. *Algorithmica*, 27(2):120–130, 2000.

[35] LIMA, E. L. **Álgebra Linear**, volume Coleção Matemática Universitária. Associação Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, 6 edition, 2003.

[36] LIU, H.; WANG, J. **A new way to enumerate cycles in graph**. In: *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, p. 57–59, 2006.

[37] LOIZOU, G.; THANISCH, P. **Enumerating the cycles of a digraph: a new preprocessing strategy**. *Information Sciences*, 27:163–182, 1982.

[38] MAKINO, K.; UNO, T. **New algorithms for enumerating all maximal cliques**. *Lecture Notes in Computational Science, SWAT 2004*, 3111:260–272, 2004.

[39] MUSIKER, G.; SCHIFFLER, R.; WILLIAMS, L. **Positivity for cluster algebras from surfaces**. *Advances in Mathematics*, 227:2241–2308, 2004.

[40] NIKOLOPOULOS, S. D.; PALIOS, L. **Detecting holes and antiholes in graphs**. *Algorithmica*, 47:119–138, 2007.

[41] PFALTZ, J. **Chordless cycles in networks**. In: *Proceedings of ICDE Workshop 2013*, p. 223–228. IEEE, 2013.

[42] READ, R.; TARJAN, R. **Bounds on backtrack algorithms for listing cycles, paths and spanning trees**. *Networks*, 5:237–252, 1975.

[43] SANKAR, K.; SARAD, A. V. **A time and memory efficient way to enumerate cycles in a graph**. In: *Proceedings of ICIAS 2007*, p. 498–500. IEEE, 2007.

[44] SEVEN, A. I. **Recognizing cluster algebras of finite type**. *The Electronic Journal of Combinatorics*, 14:1–35, 2007.

[45] SIMÕES, R. C. G. **Cluster-Tilting Theory**. Master's thesis, Departamento de Matemática, Universidade de Lisboa, 2008.

[46] SIPSER, M. **Introduction to the Theory of Computation**. Thomson, 2 edition, 2006.

[47] SOKHN, N.; BALTENSPERGER, R.; BERSIER, L.; HENNEBERT, J.; NITSCHE, U. **Identification of chordless cycles in ecological networks**. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, COMPLEX 2012*, 126:316–324, 2013.

[48] SPEYER, D. E. **Cyclically orientable graphs**, Fevereiro, 2008. Disponível em: `http://arxiv.org/pdf/math/0511233v1.pdf`. Acesso em: 10 ago. 2015.

[49] SPINRAD, J. **Finding large holes**. *Information Processing Letters*, 39:227–229, 1991.

[50] SZWARCFITER, J. L. **Grafos e algoritmos computacionais**. Rio de Janeiro, Campus, 2 edition, 1988.

[51] TAMASSIA, R.; GOODRICH, M. T. **Estrutura de Dados e Algoritmos em Java**. Porto Alegre, Ed. Bookman, 4 edition, 2007.

[52] TARJAN, R. E. **Depth first search an linear graph algorithms**. *SIAM Journal on Computing*, 1:146–160, 1972.

[53] TARJAN, R. E. **Enumeration of the elementary circuits of a directed graph**. *SIAM Journal on Computing*, 2(3):211–216, 1973.

[54] TIERNAN, J. C. **An efficient search algorithm to find the elementary circuits of a graph**. *Communications of ACM*, 13(12):722–726, 1970.

[55] TOMITA, E.; TANAKA, A.; TAKAHASHI, H. **The worst-case time complexity for generating all maximal cliques and computational experiments**. *Theoretical Computer Science*, 363:28–42, 2006.

[56] UNO, T.; SATOH, H. **An efficient algorithm for enumerating chordless cycles and chordless paths**, 2014. Disponível em: `http://arxiv.org/pdf/1404.7610v1.pdf`. Acesso em: 10 ago. 2015.

[57] WILD, M. **Generating all cycles, chordless cycles, and hamiltonian cycles with the principle of exclusion**. *Journal of Discrete Algorithms*, 6(1):93–102, 2008.

[58] WILSON, R. J.; WATKINS, J. J. **Graphs: An Introductory Approach**. Wiley, Michigan University, 1990.

[59] ZELEVINSKY, A. **What is a cluster algebras?** *Notices of the American Mathematical Society*, 54(11):1494–1495, Dezembro, 2007.

APÊNDICE A

Definições Complementares

Neste apêndice, fornecemos algumas definições adicionais que algumas vezes são assumidas como de conhecimento geral ou conhecimento básico a respeito do tema.

A.1 Sobre grafos

Começamos com as definições de algumas propriedades mais elementares de grafos. Quando não for explicitado, trataremos de grafos não orientados.

Definição A.1 *O **grau** de um vértice é igual ao número de arestas que estão conectadas a ele (Figura A.1).*

(a) *Grau um.* (b) *Grau dois.* (c) *Grau três.* (d) *Grau n.*

Figura A.1: *A figuras de (a) a (d) representam o grau do vértice a.*

Definição A.2 *Em um grafo orientado, **grau de entrada** de um vértice, denotado por $g^-(v)$, é igual ao número de arestas que chegam a ele.*

Definição A.3 *Em um grafo orientado, **grau de saída** de um vértice, denotado por $g^+(v)$, é igual ao número de arestas que saem dele.*

Definição A.4 *Um **laço** é uma aresta do tipo $e = (a,a)$, ou seja, que relaciona um vértice a ele próprio.*

Um laço contribui com 2 para o grau de um vértice em um grafo não orientado e com 1 para o grau de entrada e 1 para o grau de saída em um grafo orientado.

Definição A.5 *Arestas **múltiplas** ou **paralelas** são duas ou mais arestas que possuem os mesmos vértices como extremidade.*

Definição A.6 *Um grafo é **completo** quando existe uma aresta entre cada par de seus vértices. Neste caso, utilizamos a notação K_n para designar um grafo completo com n vértices (Figura A.2).*

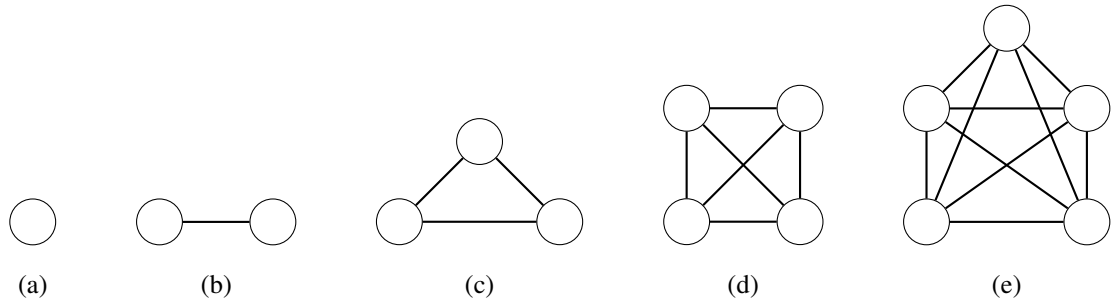


Figura A.2: *Grafos completos: K_1, K_2, K_3, K_4 e K_5.*

Definição A.7 *Um grafo H é **subgrafo** de um grafo G se o seu conjunto de vértices é um subconjunto do conjunto de vértices de G e o seu conjunto de arestas é um subconjunto do conjunto de arestas de G (Figura A.3).*

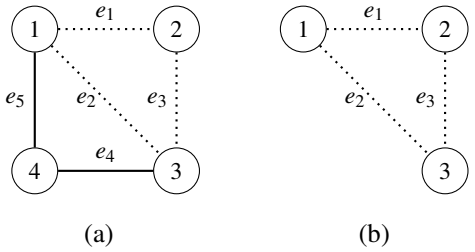


Figura A.3: *O grafo (b) é subgrafo de (a).*

Definição A.8 *Um subgrafo é **máximo** se não existe outro subgrafo de maior cardinalidade de vértices com a mesma propriedade em análise. De forma análoga, um subgrafo é **mínimo** se não existe outro subgrafo de menor cardinalidade de vértices com igual propriedade.*

Definição A.9 *Um subgrafo **induzido** é composto por um subconjunto de vértices de um grafo G, juntamente com todas as arestas que possuem esses vértices como extremidade. Como exemplo, temos a ilustração de um subgrafo induzido pelo subconjunto de vértices $\{1,2,3,5\}$ do grafo completo K_5 (Figura A.4).*

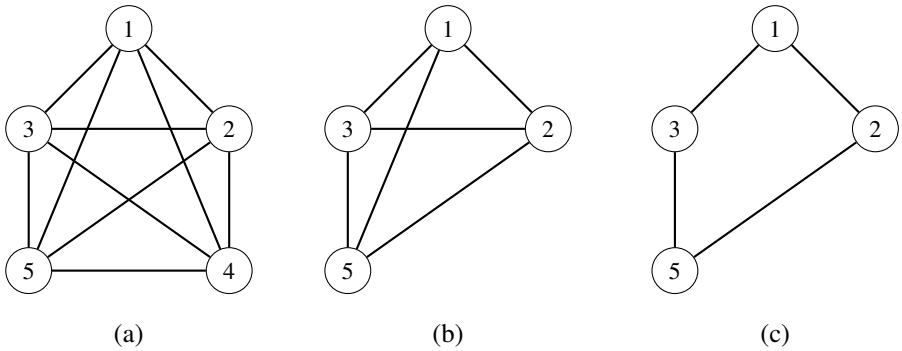


Figura A.4: *O grafos (b) e (c) são, respectivamente, exemplos de de subgrafos induzido e não induzido do grafo (a).*

Definição A.10 *Dois grafos, G_1 e G_2, com $|V(G_1)| = |V(G_2)| = n$, são denominados* ***isomorfos*** *quando existir uma função $f : V(G_1) \to V(G_2)$ em que cada elemento do seu domínio (Figura A.5(a)) está associado a um único elemento do contradomínio (Figura A.5(b)), caracterizando assim uma função bijetora, tal que $(v,w) \in E(G_1)$, se e somente se $(f(v), f(w)) \in E(G_2)$, para todo $v,w \in V(G_1)$.*

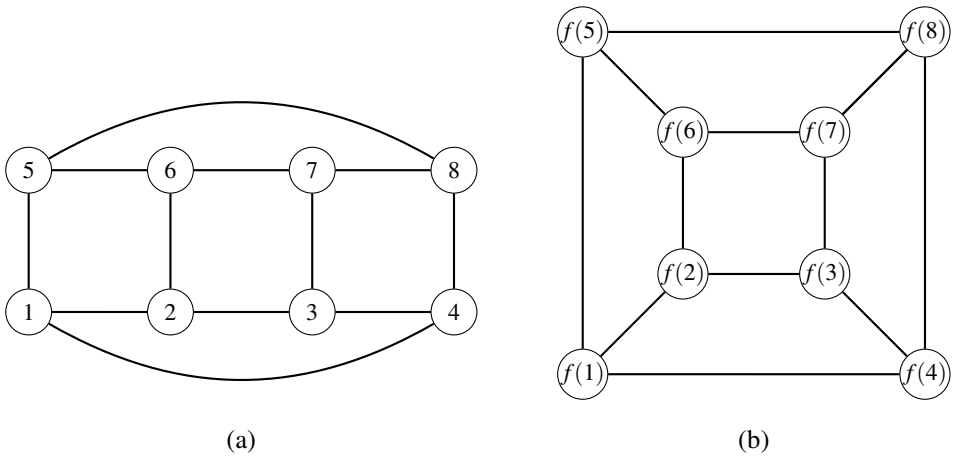


Figura A.5: *Os grafos (a) e (b) são isomorfos entre si.*

Definição A.11 *Um* ***grafo de interseção*** *é um grafo que representa o padrão de interseções de uma família de conjuntos. Qualquer grafo pode ser representado como um grafo de interseção, mas algumas classes importantes de grafos especiais podem ser definidas pelos tipos de conjuntos que são utilizados para formar uma representação da interseção deles.*

Definição A.12 *Um* ***grafo de intervalo*** *é o grafo de interseção de uma família de intervalos na linha real. Ele tem um vértice para cada intervalo na família, e uma aresta entre cada par de vértices correspondem a intervalos que se cruzam.*

Todo grafo de intervalo é um grafo cordal, pois os intervalos podem ser representados por caminhos que se sobrepõem e os caminhos são subárvores de uma árvore.

Definição A.13 *Um subgrafo de um grafo conexo G é uma* ***árvore geradora*** *se ele é uma árvore contendo todos os vértices de G (Figura A.6).*

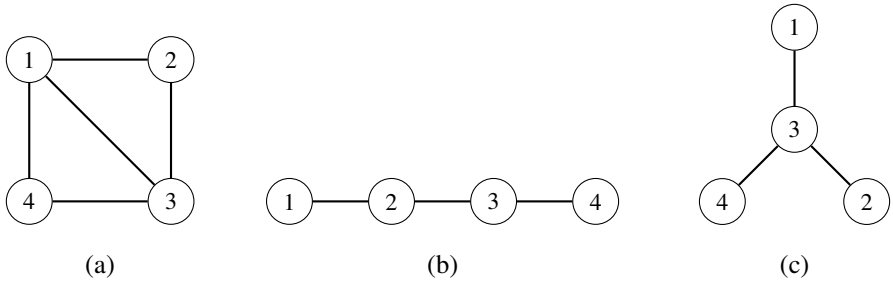


Figura A.6: *Os subgrafos (b) e (c) são exemplos de árvore geradora para o grafo (a).*

Definição A.14 *Um grafo G é* ***bipartido*** *se seus vértices podem ser separados em dois conjuntos disjuntos de tal forma que todas as arestas do grafo conectam vértices de conjuntos distintos.*

No caso da Figura A.7, os vértices podem ser divididos nos conjuntos $V_1 = \{a,b,c\}$ e $V_2 = \{d,e,f\}$. Como todos os vértices de V_1 estão ligados a todos os vértices de V_2, então temos um **grafo bipartido completo** $K_{n,m}$, sendo n e m as cardinalidades dos conjuntos V_1 e V_2, respectivamente.

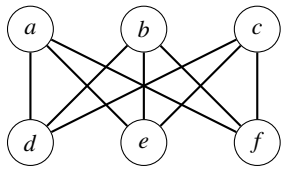


Figura A.7: *Grafo bipartido completo* $K_{3,3}$.

Definição A.15 *Sejam G_1 e G_2 dois grafos, o* ***produto Cartesiano*** *$G_1 \times G_2$ é o grafo contendo o conjunto de vértices $V(G_1 \times G_2) = V(G_1) \times V(G_2)$, e dois vértices (v_1,v_2) e (u_1,u_2) de $G_1 \times G_2$ são adjacentes se $[v_2 = u_2$ e $(v_1,u_1) \in E(G_1)]$ ou $[v_1 = u_1$ e $(v_2,u_2) \in E(G_2)]$.*

A.2 Complexidade computacional

Para as definições e notações utilizadas nesta seção, utilizamos como texto base os livros de Sipser [46] e Cormen et al. [10].

A **Teoria da Complexidade** tem como objetivo a classificação de problemas de acordo com o tempo e espaço necessários para encontrar sua solução. Em geral, uma questão muito importante é o modelo utilizado para medir o tempo. No entanto,

em modelos determinísticos, os requisitos de tempo não diferem muito e, desta forma, a escolha do modelo não é crucial, já que o sistema de classificação não é muito sensível a diferenças relativamente pequenas na complexidade.

De maneira específica, a complexidade computacional estabelece limites (inferior e superior) no número de passos necessários para resolver um problema computacional. Utilizando-se de qualquer algoritmo, ela identifica problemas computacionais potencialmente difíceis, identifica traços comuns entre tais problemas e acumula evidências sobre a dificuldade de sua resolução.

Uma maneira de medir a eficiência de um algoritmo é através de sua complexidade. Segundo Szwarcfiter [50], "um algoritmo é eficiente precisamente quando a sua complexidade for um polinômio no tamanho de sua entrada".

Devido ao fato de que o tempo exato de execução de um algoritmo frequentemente é uma expressão complexa, usualmente apenas o estimamos. Em uma forma conveniente de estimativa, chamada *análise assintótica*, buscamos entender o tempo de execução do algoritmo quando ele é executado sobre entradas grandes. Assim, consideramos somente o termo de mais alta ordem da expressão para o tempo de execução do algoritmo, desconsiderando todos coeficientes e os outros termos de mais baixa ordem.

Exemplo A.16 *A função* $f(n) = 8 \cdot n^2 + 5 \cdot n + 24$ *tem três termos e o termo de mais alta ordem é* $8 \cdot n^2$*. Desconsiderando o coeficiente 8, dizemos que* f *é assintoticamente no máximo* n^2*. A* **notação assintótica** *para descrever esse relacionamento é* $f(n) \in O(n^2)$.

Definição A.17 *Sejam* f *e* g *duas funções* $f, g : \mathbb{N} \to \mathbb{R}^+$*. Dizemos que* $f(n) \in O(g(n))$ *se existem inteiros positivos* c *e* n_0 *tais que, para todo inteiro* $n \geq n_0$*,* $f(n) \leq c \cdot g(n)$.

Quando $f(n) \in O(g(n))$, dizemos que $g(n)$ é um limitante superior para $f(n)$, ou mais precisamente que $g(n)$ é um limitante superior assintótico para $f(n)$, para enfatizar que estamos suprimindo fatores constantes. Intuitivamente, $f(n) \in O(g(n))$ significa que f é menor ou igual a g se desconsiderarmos diferenças até um fator constante.

Exemplo A.18 *Considere a função* $f_1(n) = 5 \cdot n^3 + 2 \cdot n^2 + 22 \cdot n + 6$*. O termo de mais alta ordem é* $5 \cdot n^3$*. Desconsiderando o coeficiente 5, temos que* $f_1(n) \in O(n^3)$*. Podemos verificar se esse resultado satisfaz a definição formal atribuindo 6 para* c *e 10 para* n_0*. Então,* $f_1(n) = 5 \cdot n^3 + 2 \cdot n^2 + 22 \cdot n + 6 \leq 6 \cdot n^3$ *para todo* $n \geq 6$.

Além disso, $f_1(n) \in O(n^4)$ *porque* n^4 *é maior que* n^3 *e continua sendo um limitante superior assindético sobre* f_1*. Entretanto,* $f_1(n)$ *nunca será* $O(n^2)$*, independente de quais valores sejam atribuídos a* c *e a* n_0.

Dizemos que um algoritmo é $\Theta(g(n)$ se e somente se $f(n) \in O(g(n))$ e $g(n) \in O(f(n))$. Essa notação é utilizada na análise do caso médio.

É natural questionar se todos os problemas podem ser resolvidos em tempo polinomial. A resposta é não. Por exemplo, existem problemas, como o famoso "Problema da parada (halting)" de Turing, que não podem ser resolvidos por qualquer computador, não importa quanto tempo seja fornecido. Também existem problemas que podem ser resolvidos, mas não no tempo $O(n^k)$ para qualquer constante k. Em geral, consideramos problemas que podem ser resolvidos por algoritmo de tempo polinomial como sendo *tratáveis*, e os problemas que exigem tempo superpolinomial como *intratáveis*, ou difíceis.

Problemas algorítmicos são aqueles formados por uma *instância*, que representa o conjunto de dados, e uma *questão*, que representa o objetivo a ser alcançado. A instância inclui todos os dados necessários para sua formulação e compreensão, enquanto a questão caracteriza o problema para o qual devemos fornecer uma resposta.

Existem classes gerais nas quais os problemas algorítmicos podem estar inseridos. Falaremos sobre três delas: a classe dos problemas de decisão, localização e otimização.

Problemas de decisão são aqueles que exigem uma resposta afirmativa ou negativa para uma questão, ou seja, a resposta é simplesmente "sim" ou "não". Em um problema de localização, o objetivo é identificar uma estrutura que satisfaça um conjunto de propriedades fornecidas. Por fim, um problema de otimização é aquele que possui como objetivo a satisfação de determinados critérios de otimização.

Em geral, existe um relacionamento entre as classes de problemas, sendo que um problema de decisão é menos difícil do que um problema de localização que, por sua vez, é menos difícil que um problema de otimização. Segundo Cormen et al. [10], é simples reformular um problema de otimização como um problema de decisão não mais difícil que o primeiro.

Para um entendimento efetivo de algumas definições fornecidas na tese, definiremos as classes P, NP e NPC (NP-completa).

Antes de serem apresentadas estas definições, é importante formalizar a noção de problemas que podem ser resolvidos em *tempo polinomial*. Em geral, esses problemas são considerados tratáveis de acordo com os três argumentos a seguir:

(a) Uma vez que um algoritmo de tempo polinomial para um problema é descoberto, algoritmos mais eficientes frequentemente o seguem. Isto significa que um problema resolvido por um algoritmo com complexidade $O(n^{100})$ deveria ser considerado intratável. No entanto, há uma quantidade ínfima de problemas práticos aos quais esta complexidade se aplica e, para estes problemas, é bastante provável que um algoritmo com menor tempo de execução logo seja descoberto.
(b) Um problema que pode ser resolvido em tempo polinomial em um modelo pode ser resolvido em tempo polinomial de forma similar em outro modelo.

(c) A classe de problemas que podem ser resolvidos em tempo polinomial têm propriedades de fechamento interessantes, pois os polinômios são fechados sob a adição, a multiplicação e a composição. Por exemplo, se a saída de um algoritmo de tempo polinomial é alimentada na entrada de outro, o algoritmo composto é polinomial.

Para os propósitos desta tese, diferenças polinomiais no tempo de execução de algoritmos são consideradas pequenas, enquanto que diferenças exponenciais são consideradas grandes.

Algoritmos de *tempo exponencial* surgem quando resolvemos problemas por meio de busca através de um espaço de soluções, chamado *busca por força-bruta*. Por exemplo, uma maneira de fatorar um número em seus primos constituintes é buscar através de seus potenciais divisores. O tamanho do espaço de busca é exponencial, portanto essa busca usa tempo exponencial. Às vezes, a busca por força-bruta pode ser evitada por meio de um entendimento mais profundo de um problema, que pode revelar um algoritmo de tempo polinomial de maior utilidade.

O número de passos que um algoritmo usa sobre uma entrada específica pode depender de vários parâmetros. Por exemplo, se a entrada é um grafo, o número de passos pode depender do número de nós, do número de arestas, do grau máximo do grafo ou alguma combinação desses e/outros fatores.

Por motivos de simplicidade, computamos o tempo de execução de um algoritmo puramente como uma função do comprimento da cadeia que representa a entrada e não consideramos qualquer outro parâmetro.

Na análise do pior caso (resp. melhor caso), consideramos o maior tempo (resp. menor tempo) de execução de todas as entradas de um comprimento específico. Na análise do caso médio, consideramos a média de todos os tempos de execução de entradas de um comprimento específico.

A.3 Caráter NP-completo e as classes P e NP e NPC

Ao longo desta seção, serão mencionadas três classes de problemas: P, NP e NPC, sendo esta última classe a de problemas NP-completos.

Em geral, para classificar um problema como pertencente à classe P, NP ou NPC é analisado o problema de decisão associado a ele. Isto ocorre devido ao fato de que esta classe é a mais simples dentre as três citadas anteriormente. Além disso, qualquer prova de que um problema de decisão seja intratável pode ser estendida aos demais casos.

A classe P é aquela cujas linguagens são decidíveis em tempo polinomial sobre uma máquina de Turing determinística de uma única fita. Portanto, podemos definir a classe P como o conjunto de problemas de decisão que possuem algoritmos com complexidade de tempo polinomial que o resolvam.

Essa classe desempenha um papel central na teoria da complexidade e é importante porque

(a) P é invariante para todos os modelos de computação que são polinomialmente equivalentes a uma máquina de Turing determinística de uma única fita; e
(b) P corresponde aproximadamente à classe de problemas que são solúveis realisticamente em um computador.

Observação A.19 *Não é porque ainda não se conhece um algoritmo polinomial que resolve um determinado problema que podemos afirmar que ele não está na classe dos problemas P.*

Um problema pertence à classe de problemas NP se existe uma máquina de Turing não determinística de uma única fita que o decide em tempo polinomial. O termo NP vem do acrônimo *Non-deterministic Polynomial*.

De acordo com Szwarcfiter [50], *"Define-se a classe NP como sendo aquela que compreende todos os problemas de decisão D, tais que existe uma justificativa à resposta SIM para D, cujo passo de reconhecimento pode ser realizado por um algoritmo polinomial no tamanho da entrada de D"*.

Segundo Cormen et al. [10], *"A classe NP consiste dos problemas que são verificáveis em tempo polinomial"*, ou seja, se for fornecido um certificado de uma solução, é possível verificar sua validade em tempo polinomial de acordo com o tamanho da entrada para o problema.

Podemos perceber, através desta definição, que qualquer problema na classe P pertence a NP, pois este pode ser resolvido em tempo polinomial e o algoritmo verificador pode ser uma alteração do algoritmo que o resolve.

Um problema está na classe NPC - e vamos nos referir a ele como um problema NP-completo - se ele está em NP e é tão difícil quanto qualquer problema em NP.

Existem algumas técnicas para mostrar que um problema é NP-completo. Essa demonstração é uma declaração sobre a dificuldade para resolver o problema evidenciando o quanto ele é difícil e não o quanto é fácil.

Segundo Garey e Johnson [19], a principal técnica usada para demonstrar que dois problemas são relacionados é através da redução de um para outro, utilizando uma transformação construtiva que mapeia qualquer instância do primeiro problema em uma instância equivalente do segundo. Tal transformação provê recursos para converter qualquer algoritmo que resolva o segundo problema em um algoritmo correspondente para resolver o primeiro.

De maneira simplificada, o método da redutibilidade consiste na realização da transformação, em tempo polinomial, da saída de um problema na entrada de outro. De

forma geral, um problema está na classe NPC se ele está em NP e existe uma redução de um problema NPC conhecido a ele.

Devido à redutibilidade, se for encontrada uma solução polinomial para qualquer problema NP-completo, então todo problema em NP e, consequentemente, todo problema em NP-completo pode ser resolvido em tempo polinomial. A maioria dos teóricos da Ciência da Computação acredita que isto não acontece, visto que existe uma grande quantidade de problemas nesta faixa e até hoje não encontrou-se nenhum algoritmo polinomial para resolvê-los.

A.4 Sobre álgebra

Agora, apresentamos algumas definições consideradas básicas ou elementares em álgebra.

Definição A.20 *Um* **anel** *A é um conjunto munido de duas operações $A \times A \to A$, denotado, respectivamente, por $+$ e $\cdot$ chamadas, respectivamente, por adição e multiplicação do anel, tais que:*

(a) o conjunto A é um grupo abeliano em relação à adição, isto é:

- $(\forall a,b,c \in A)(a+(b+c)=(a+b)+c)$;
- $(\forall a,b \in A)(a+b=b+a)$;
- *existe elemento neutro para essa adição. Ele será indicado por 0_A ou apenas 0, quando não houver possibilidade de confusão: é o zero do anel. Portanto, para todo $a \in A$, temos que $a+0=a$; e*
- *todo elemento de A admite um simétrico aditivo, ou seja, para todo $a \in A$ existe um elemento em A, indicado por $(-a)$, de forma que $a+(-a)=0$.*

(b) a multiplicação é distributiva em relação à adição: $(\forall a,b,c \in A)(a \cdot (b+c) = a \cdot b + a \cdot c$ e $(a+b) \cdot c = a \cdot c + b \cdot c)$; e

(c) a multiplicação é associativa: $(\forall a,b,c \in A)(a \cdot (b \cdot c)) = (a \cdot b) \cdot c$ e existe elemento neutro para a multiplicação. Ele será indicado por 1_K ou apenas 1, quando não houver possibilidade de confusão: é a unidade do anel. Portanto, para todo $a \in K$, temos que $a \cdot 1 = 1 \cdot a = a$.

A terceira condição corresponde ao anel associativo com unidade que consideramos aqui.

Exemplo A.21 *Alguns exemplos de aneis:*

- $(\mathbb{Z},+,\cdot)$ *é um anel, anel dos inteiros.*

- *Anel das matrizes:* $(M_n(\mathbb{Z}),+,\cdot)$.
- *Anel dos racionais:* $(\mathbb{Q},+,\cdot)$.
- *Anel dos reais:* $(\mathbb{R},+,\cdot)$.
- *Anel dos complexos:* $(\mathbb{C},+,\cdot)$.

Definição A.22 *Dizemos que um* **anel** $(A,+,\cdot)$ *é* ***comutativo*** *se e somente se a multiplicação é comutativa, isto é,* $x\cdot y = y\cdot x, \forall x,y \in A$.

Definição A.23 *Um anel comutativo K recebe o nome de* ***corpo*** *se todo elemento não nulo de K admite simétrico multiplicativo, ou seja,* $(\forall x \in K)(x \neq 0 \rightarrow \exists y \in K \mid x\cdot y = 1)$.

Exemplo A.24 *Os anéis* $\mathbb{Q}$, $\mathbb{R}$ *e* $\mathbb{C}$ *são corpos, mas o anel* $\mathbb{Z}$ *não é, visto que somente o* 1 *e o* -1 *admitem simétrico multiplicativo.*

Definição A.25 *Sendo K um corpo, um* k**-espaço vetorial** V *é um conjunto munido de duas operações: a primeira* $V\times V \rightarrow V$, *denotada por* $+$ *e chamada de adição e a segunda* $K\times V \rightarrow V$, *denotada por* $\cdot$ *e chamada de multiplicação externa tais que:*

(a) V é um grupo abeliano em relação à adição, isto é:

- $(\forall x,y,z \in V)(x+(y+z) = (x+y)+z)$;
- $(\forall x,y \in V)(x+y = y+x)$;
- *existe elemento neutro para essa adição. Ele será indicado por* 0_V *ou apenas* 0, *quando não houver possibilidade de confusão: é o zero do anel. Portanto, para todo* $x \in V$, *temos que* $x+0 = x$; *e*
- *todo elemento de V admite um simétrico aditivo, ou seja, para todo* $x \in V$ *existe um elemento em V, indicado por* $(-x)$, *de forma que* $x+(-x) = 0$.

(b) a multiplicação externa é distributiva em relação à adição: $(\forall a,b \in K \text{ e } x,y \in V)(a\cdot(x+y) = a\cdot x + a\cdot y \text{ e } (a+b)\cdot x = a\cdot x + b\cdot x)$; *e*

(c) a multiplicação externa é associativa: $(\forall a,b \in K \text{ e } x \in V)(a\cdot(b\cdot x) = (a\cdot b)\cdot x)$ *e* $1\cdot x = x$.

Definiremos apenas álgebras sobre corpo.

Definição A.26 *Sendo K um corpo, uma* K**-álgebra** A *é um conjunto munido de duas operações internas* $A\times A \rightarrow A$, *denotadas, respectivamente, por* $+$ *e* $\cdot$ *chamadas, respectivamente, de adição e multiplicação e munido também de uma multiplicação externa* $K\times V \rightarrow V$, *denotada por* $\cdot$, *tais que:*

(a) A munido da adição e da multiplicação externa é um K-espaço vetorial.

(b) A munido das operações internas é um anel; e

(c) temos a seguinte propriedade: $(\forall a,b \in A \text{ e } x \in K)(x\cdot(a\cdot b) = a\cdot(x\cdot b) = (x\cdot a)\cdot b$.

Definição A.27 *Uma* **álgebra** *é dita* **comutativa** *se ela é um anel comutativo.*

Para maiores informações, veja [24] e [35].

Índice Remissivo